

Oxygen XML Developer plugin 13.2.0

Contents

Chapter 1: Introduction.....	11
Chapter 2: Installation.....	13
Installation Requirements.....	14
Platform Requirements.....	14
Operating System.....	14
Environment Requirements.....	14
Installation Instructions.....	14
Obtaining and Registering a License Key.....	14
Named User License Registration.....	15
Named User License Registration with Text File.....	16
Named User License Registration with XML File.....	16
How Floating (Concurrent) Licenses Work.....	16
License Registration with an Activation Code.....	21
Uninstalling the Application.....	21
Uninstalling the Eclipse plugin.....	21
Performance Problems.....	21
External Processes.....	21
Chapter 3: Getting Started.....	23
Getting Help.....	24
Supported Types of Documents.....	24
Perspectives.....	24
Oxygen XML Developer plugin XML Perspective	25
XSLT Debugger Perspective	29
XQuery Debugger Perspective	30
Oxygen XML Developer plugin Database Perspective	31
Chapter 4: Editing Documents.....	35
Working with Unicode.....	36
Opening and Saving Unicode Documents.....	36
Opening and Closing Documents.....	36
Creating New Documents.....	36
Saving Documents.....	40
Opening and Saving Remote Documents via FTP/SFTP	40
Opening the Current Document in System Application.....	44
Closing Documents.....	44

Viewing File Properties.....	44
Grouping Documents in XML Projects.....	44
Creating a New Project.....	44
Editing XML Documents.....	46
Associate a Schema to a Document.....	46
Streamline with Content Completion.....	49
Validating XML Documents.....	57
Document Navigation.....	65
Large Documents.....	68
Working with XML Catalogs.....	71
Converting Between Schema Languages.....	72
Formatting and Indenting Documents (Pretty Print).....	74
Viewing Status Information.....	74
XML Editor Specific Actions.....	75
Editing XHTML Documents.....	78
Editing XML Schemas.....	78
XML Schema Text Editor.....	78
XML Schema Diagram Editor.....	79
Contextual Editing.....	107
Create an XML Schema From a Relational Database Table.....	107
Generate Sample XML Files.....	107
XML Schema Regular Expressions Builder.....	111
Generating Documentation for an XML Schema.....	112
Searching and Refactoring Actions.....	119
Search and Refactor Operations Scope.....	121
Resource Hierarchy / Dependencies View.....	122
Component Dependencies View.....	123
Highlight Component Occurrences.....	124
XML Schema Quick Assist Support.....	125
Editing Relax NG Schemas.....	125
Relax NG Schema Diagram.....	125
Relax NG Editor Specific Actions.....	129
Searching and Refactoring Actions.....	130
Resource Hierarchy/Dependencies View.....	130
Component Dependencies View.....	131
RNG Quick Assist Support.....	132
Configuring a Custom Datatype Library for a RELAX NG Schema.....	133
Editing NVDL Schemas.....	133
NVDL Schema Diagram.....	133
NVDL Editor Specific Actions.....	135
Searching and Refactoring Actions.....	135
Component Dependencies View.....	135
Editing XSLT Stylesheets.....	136
Validating XSLT Stylesheets.....	136
Contextual Editing.....	137

Syntax Highlight.....	137
Content Completion in XSLT Stylesheets.....	137
The XSLT/XQuery Input View.....	142
The XSLT Outline View.....	144
XSLT Stylesheet Documentation Support.....	146
Generating Documentation for an XSLT Stylesheet.....	146
Finding XSLT References and Declarations.....	154
Highlight Component Occurrences.....	154
XSLT Refactoring Actions.....	154
Resource Hierarchy/Dependencies View.....	155
Component Dependencies View.....	157
XSLT Quick Assist Support.....	158
Editing XQuery Documents.....	158
XQuery Outline View.....	158
Folding in XQuery Documents.....	159
Generating HTML Documentation for an XQuery Document.....	160
Editing CSS Stylesheets.....	161
Validating CSS Stylesheets.....	161
Content Completion in CSS Stylesheets.....	162
CSS Outline View.....	163
Folding in CSS Stylesheets.....	163
Formatting and Indenting CSS Stylesheets (Pretty Print).....	163
Other CSS Editing Actions.....	163
Editing JSON Documents.....	163
JSON Editor Text Mode.....	164
JSON Editor Grid Mode.....	165
Validating JSON Documents.....	165
Convert XML to JSON.....	166
Editing XProc Scripts.....	166
Editing Schematron Schemas.....	167
Combined RELAX NG / W3C XML Schemas and Schematron Schema.....	168
Validate an XML Document.....	168
Handling Read-Only Files.....	169

Chapter 5: Predefined Document Types.....171

Document Type.....	172
The DocBook 4 Document Type.....	172
Transformation Scenarios.....	172
Templates.....	172
Inserting olink Links in Docbook Documents.....	172
The DocBook 5 Document Type.....	175
Transformation Scenarios.....	175
Templates.....	176
Inserting olink Links in Docbook Documents.....	176

The DocBook Targetset Document Type.....	179
Templates.....	179
The DITA Topics Document Type.....	179
Transformation Scenarios.....	179
Templates.....	179
The DITA Map Document Type.....	180
Transformation Scenarios.....	180
Templates.....	181
The XHTML Document Type.....	181
Transformation Scenarios.....	181
Templates.....	181
The TEI ODD Document Type.....	181
Transformation Scenarios.....	182
Templates.....	182
The TEI P4 Document Type.....	182
Transformation Scenarios.....	182
Templates.....	183
The TEI P5 Document Type.....	183
Transformation Scenarios.....	183
Templates.....	183
The EPUB Document Type.....	183
Chapter 6: Grid Editor.....	185
Layouts: Grid and Tree.....	186
Navigating the Grid.....	186
Specific Grid Actions.....	187
Sorting a Table Column.....	187
Inserting a Row in a Table.....	187
Inserting a Column in a Table.....	187
Clearing the Content of a Column.....	187
Adding Nodes.....	187
Duplicating Nodes.....	187
Refresh Layout.....	188
Start Editing a Cell Value.....	188
Stop Editing a Cell Value.....	188
Drag and Drop in the Grid Editor.....	188
Copy and Paste in the Grid Editor.....	188
Bidirectional Text Support in the Grid Editor.....	190
Chapter 7: Transforming Documents.....	191
Output Formats.....	192
Transformation Scenario.....	193
Batch Transformation.....	193

Built-in Transformation Scenarios.....	193
Defining a New Transformation Scenario.....	194
Transformation Scenarios View.....	204
XSLT Processors.....	205
Supported XSLT Processors.....	205
Configuring Custom XSLT Processors.....	206
Configuring the XSLT Processor Extensions Paths.....	206
XSL-FO Processors.....	207
The Built-in XSL-FO Processor.....	207
Add a Font to the Built-in FOP - The Simple Version.....	207
Add a Font to the Built-in FOP.....	208
XProc Transformations.....	211
XProc Transformation Scenario.....	211
Integration of an External XProc Engine.....	212
Chapter 8: Querying Documents.....	213
Running XPath Expressions.....	214
What is XPath.....	214
Oxygen's XPath Console.....	214
Working with XQuery.....	217
What is XQuery.....	217
Syntax Highlight and Content Completion.....	218
XQuery Outline View.....	218
The XQuery Input View.....	219
XQuery Validation.....	221
Other XQuery Editing Actions.....	222
Transforming XML Documents Using XQuery.....	222
Chapter 9: Debugging XSLT Stylesheets and XQuery Documents.....	227
Overview.....	228
Layout.....	228
Control Toolbar.....	229
Information View.....	230
Multiple Output Documents in XSLT 2.0.....	239
Working with the XSLT / XQuery Debugger.....	239
Steps in a Typical Debug Process.....	239
Using Breakpoints.....	240
Determining What XSLT / XQuery Expression Generated Particular Output.....	240
Debugging Java Extensions.....	242
Supported Processors for XSLT / XQuery Debugging.....	242
Chapter 10: Profiling XSLT Stylesheets and XQuery Documents.....	243
Overview.....	244

Viewing Profiling Information.....	244
Invocation Tree View.....	244
Hotspots View.....	244
Working with XSLT/XQuery Profiler.....	245
Chapter 11: Working with Archives.....	247
Browsing and Modifying Archive Structure.....	248
Working with EPUB.....	249
Create an EPUB.....	249
Publish to EPUB.....	250
Editing Files From Archives.....	250
Chapter 12: Working with Databases.....	251
Relational Database Support.....	252
Configuring Database Data Sources.....	252
Configuring Database Connections.....	258
Resource Management.....	263
SQL Execution Support.....	268
Native XML Database (NXD) Support.....	270
Configuring Database Data Sources.....	271
Configuring Database Connections.....	273
Data Source Explorer View.....	276
XQuery and Databases.....	285
Build Queries With Drag and Drop From Data Source Explorer View.....	285
XQuery Transformation.....	286
XQuery Database Debugging.....	288
WebDAV Connection.....	289
How to Configure a WebDAV Connection.....	289
WebDAV Connection Actions.....	289
Chapter 13: Importing Data.....	291
Introduction.....	292
Import from Database.....	292
Import Table Content as XML Document.....	292
Convert Table Structure to XML Schema.....	295
Import from MS Excel Files.....	295
Import from HTML Files.....	295
Import from Text Files.....	296
Chapter 14: Content Management System (CMS) Integration.....	299
Integration with Documentum (CMS).....	300
Configure Connection to Documentum Server.....	300

Documentum (CMS) Actions in the Data Source Explorer View.....	301
Transformations on DITA Content from Documentum (CMS).....	305

Chapter 15: Composing Web Service Calls.....307

Overview.....	308
Composing a SOAP Request.....	308
Testing Remote WSDL Files.....	310
The UDDI Registry Browser.....	310
Generate WSDL Documentation.....	311

Chapter 16: Digital Signatures.....313

Overview.....	314
Canonicalizing Files.....	315
Certificates.....	316
Signing Files.....	316
Verifying the Signature.....	317

Chapter 17: Text Editor Specific Actions.....319

Finding and Replacing Text in the Current File.....	320
The Find All Elements / Attributes Dialog.....	320
Spell Checking.....	321
Spell Checking Dictionaries.....	322
Learned Words.....	323
Ignored Words.....	323
Automatic Spell Check.....	323
Spell Checking in Multiple Files.....	323

Chapter 18: Configuring the Application.....325

Configuring Options.....	326
Customized Default Options.....	326
Importing / Exporting Global Options.....	326
Preferences.....	327
Oxygen XML Developer plugin License.....	328
Global.....	328
Fonts.....	328
Document Type Association.....	328
Editor.....	329
CSS Validator.....	342
XML.....	343
Data Sources.....	360
Archive.....	364
Custom Editor Variables.....	365

Network Connections	365
Certificates.....	366
XML Structure Outline.....	367
Scenarios Management.....	368
View.....	368
Reset Global Options.....	369
Scenarios Management.....	369
Editor Variables.....	369
Custom Editor Variables.....	371
Chapter 19: Common Problems.....	373
XSLT Debugger Is Very Slow.....	374
Syntax Highlight Not Available in Eclipse Plugin.....	374
Problem Report Submitted on the Technical Support Form.....	374
Signature verification failed error on open or edit a resource from Documentum.....	375
Chapter 20: Terms.....	377

Chapter 1

Introduction

Welcome to the User Manual of Oxygen XML Developer plugin 13.2.0 .
This user guide is focused mainly at describing features, functionality and application interface to help you get started in no time.

Chapter

2

Installation

Topics:

- [Installation Requirements](#)
- [Installation Instructions](#)
- [Obtaining and Registering a License Key](#)
- [Uninstalling the Application](#)
- [Performance Problems](#)

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

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

Installation Requirements

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

Platform Requirements

The run-time requirements of the application are:

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

Operating System

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

Environment Requirements

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

Tools

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

Java Virtual Machine Prerequisites

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

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

Installation Instructions

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

Obtaining and Registering a License Key

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

For demonstration and evaluation purposes a time limited license is available upon request from the [Oxygen XML website](#) . This license is supplied at no cost for a period of 30 days from date of issue. During this period the software is fully

functional, enabling you to test all its aspects. Thereafter, the software is disabled and a permanent license must be purchased in order to use it. For special circumstances, if a trial period of greater than 30 days is required, please contact support@oxygenxml.com .

For definitions and legal details of the license types, consult the End User License Agreement received with the license key. It is also available at http://www.oxygenxml.com/eula_developer.html .

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

- using the application's Register dialog (the most common case) to register a named-user based or concurrent license;
- storing the license key into a file (either text or xml) that is copied into the application's install folder.

When started, the application looks for a valid license key in the following location, in this order:

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

Named User License Registration

1. Save a backup copy of the message containing the new license key.
2. Start the application.
3. Copy to the clipboard the license text as explained in the message.
4. If this is a new install of the application then it will display automatically the registration dialog when it is started. In the case you already used the application and obtained a new license, go to **Window > Preferences > oXygen, Register** button.

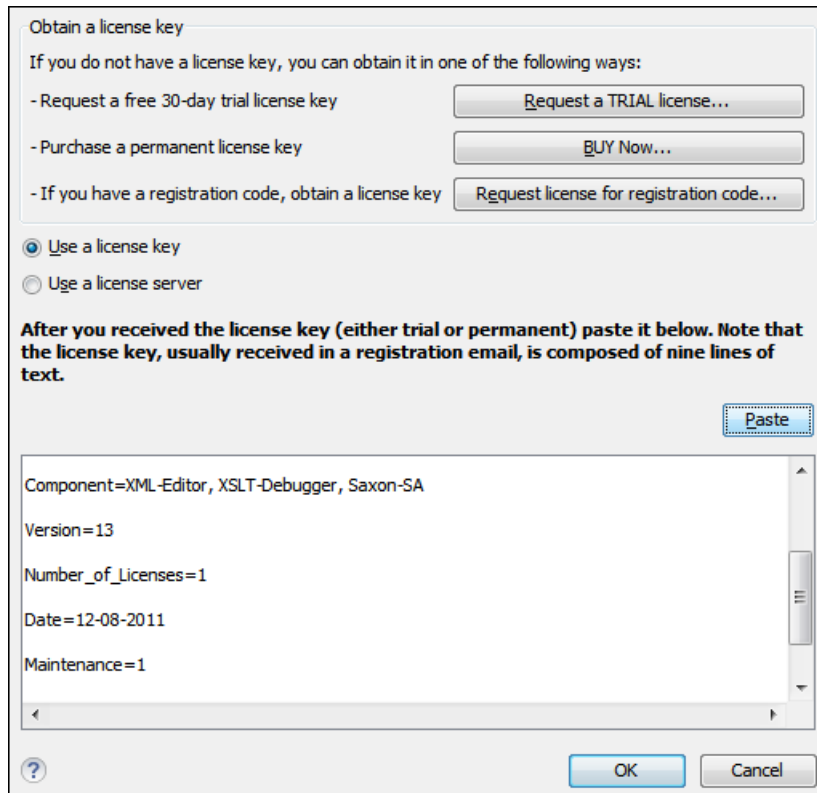


Figure 1: Registration Dialog

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

7. Press the OK button.

Named User License Registration with Text File

1. Save the license key in a file named `licensekey.txt`.
2. Copy the file in the application install folder or in the `lib` subfolder of the install folder.
3. Start the application.


Named User License Registration with XML File

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

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

How Floating (Concurrent) Licenses Work

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

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

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

- if you plan to use the application on machines running in the same local network, Oxygen XML Developer plugin can manage the licenses usage by itself. Different running instances of the application communicate between them. The registration procedure requires you to paste the license key in the license registration dialog. See [Named User License Registration](#) procedure for more details.
- if you plan to use the application on machines running in different network segments, then you must use a Oxygen XML Developer plugin floating license server. A floating license server can be installed either as a Java servlet or as a standalone process.

Setting up a Floating License Server Running as a Java Servlet

Setting up the floating license servlet.

Apache Tomcat 5.5 or higher is necessary. You can get it from: <http://tomcat.apache.org/>

1. Download the license servlet **Web ARchive (.war)** from one of the download URLs included in the registration email message.
2. Go to the Tomcat Web Application Manager page. In the **WAR file to deploy** section choose the WAR file and then press the **Deploy** button. The *oXygen License Servlet* should be up and running, but there is no licensing information set.
3. To set the license key, log on the deployment machine, and go to the Tomcat installation folder (usually `/usr/local/tomcat`). Then go to the `webapps/oXygenLicenseServlet/WEB-INF/license/` folder and create a new file called `license.txt`. Copy the license text that was sent to you via e-mail into this file and save it.
4. It is recommended to password protect your pages using a Tomcat Realm. Please refer to the Tomcat Documentation for detailed info, like the [Realm Configuration HOW-TO - Memory Based Realm section](#).

5. Once you have defined a realm resource, you have to edit `webapps/oxygenLicenseServlet/WEB-INF/web.xml` file to configure user access rights on the license server. Note that Tomcat's standard security roles are used, i.e.: **standard** for licensing and **admin** or **manager** for the license usage report page.
6. By default, the license server is logging its activity in `/usr/local/tomcat/logs/oxygenLicenseServlet.log` file. To change the log file location, edit the `log4j.appender.R2.File` property from the `/usr/local/tomcat/webapps/oxygenLicenseServlet/WEB-INF/lib/log4j.properties` configuration file.
7. Restart *oxygen License Servlet* from the Tomcat Web Application Manager page.

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

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


Report Page

You can access an activity report at

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

It displays in real time the following information:

- **License load** - a graphical indicator that shows how many licenses are available. When the indicator turns red, there are no more licenses available.
- **Floating license server status** - general information about the license server status like:
 - server start time
 - license count
 - rejected and acknowledged requests
 - average usage time
 - license refresh and timeout intervals
 - location of the license key
 - server version
- **License key information** - license key data:
 - licensed product
 - registration name
 - company name
 - license category
 - number of floating users
 - Maintenance Pack validity
- **Current license usage** - lists all currently acknowledged users:
 - user name
 - date and time when the license was granted
 - name and IP address of the computer where Oxygen XML Developer plugin runs
 - MAC address of the computer where Oxygen XML Developer plugin runs

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

Setting up a Floating License Server Running as a Standalone Process

Setting up the floating license server.

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

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

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

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

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

3. Start the server using the startup script.

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

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

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

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

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

```
licenseServer.bat myLicenseDir 54321
```


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

Install the License Server as a Windows Service

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

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

- `installWindowsService.bat [serviceName]` - install the server as a Windows service with the name `serviceName`. The parameters for the license key folder and the server port can be set in the `oxygenLicenseServer.vmoptions` file.
- `startWindowsService.bat [serviceName]` - start the Windows service.
- `stopWindowsService.bat [serviceName]` - stop the Windows service.
- `uninstallWindowsService.bat [serviceName]` - uninstall the Windows service.

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

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

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

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

👉 **Note:**

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

Common Problems

Here are the common problems that may appear when setting up a floating license server running as a standalone process.

Windows service reports Incorrect Function when started

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

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

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

When started, the Windows service reports Error 1067: The process terminated unexpectedly.

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

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

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

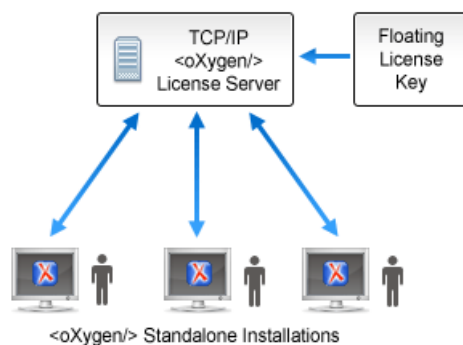


Figure 2: Floating License Server Running as a Standalone Process

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

If the maximum number of available licenses was not exceeded a license key is received from the floating license server and registered in Oxygen XML Developer plugin . The license details are displayed in the **About** dialog opened from the **Help** menu. If the maximum number of licenses was exceeded a warning dialog pops up letting you know about the

problem. The message contains information about the users who requested and successfully received the floating licenses before exceeding the maximum number of licenses.

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

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

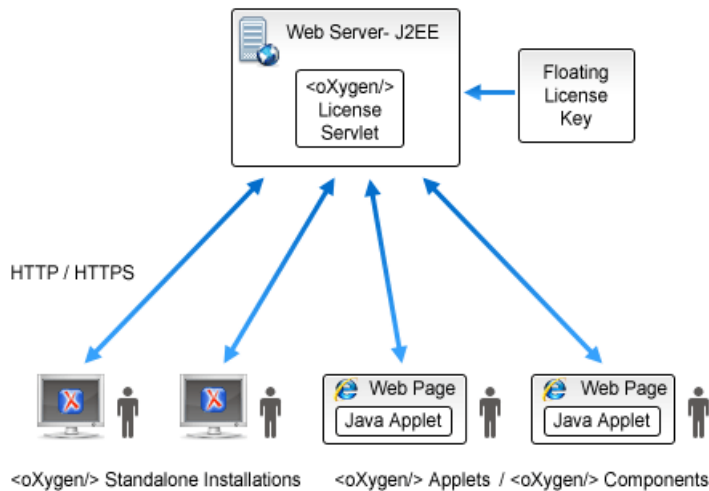



Figure 3: Floating License Server Running as a Servlet

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

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

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

Release a Floating License

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

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

5. Press the **OK** button of the dialog.

License Registration with an Activation Code

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

Uninstalling the Application

This section contains uninstallation procedures.

Uninstalling the Eclipse plugin



Caution:

The following procedure will remove the Oxygen XML 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 from the list of plugins.
3. Choose **Uninstall**.
4. Accept the Eclipse restart.
5. If you want to remove also the user preferences that were configured in the **Preferences** dialog you must remove the folder `%APPDATA%\com.oxygenxml.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.

Performance Problems

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

External Processes

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

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

Chapter

3

Getting Started

Topics:

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

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

Getting Help

Online help is available at any time while working in Oxygen XML Developer plugin by going to menu **Help > Help Contents > Oxygen plugin**

Supported Types of Documents

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

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

Perspectives

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

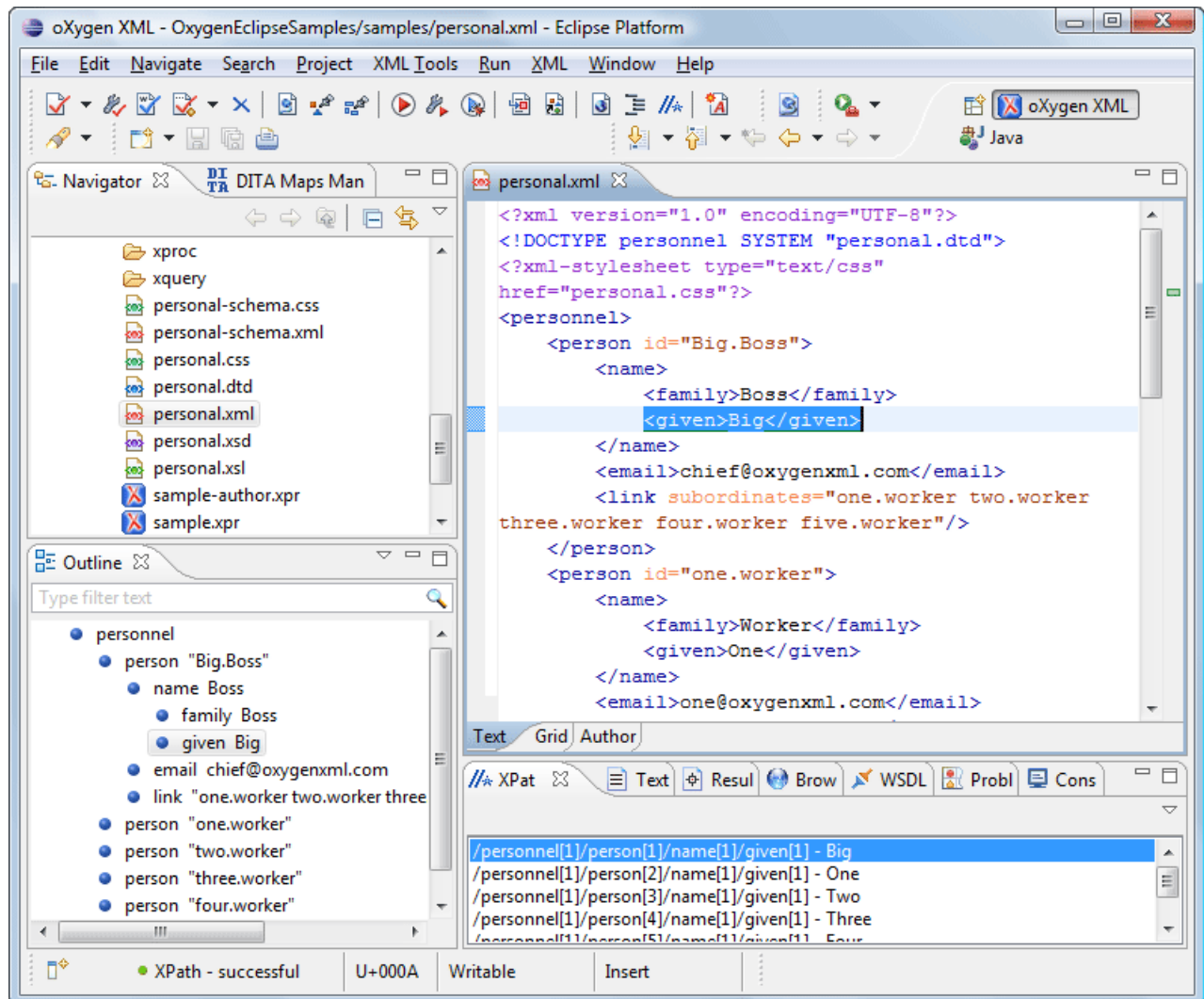
In Oxygen XML Developer plugin you can work with documents in one of the perspectives:

<i>Editor perspective</i>	Documents editing is supported by specialized and synchronized editors and views.
<i>XSLT Debugger perspective</i>	XSLT stylesheets can be debugged by tracing their execution step by step.
<i>XQuery Debugger perspective</i>	XQuery transforms can be debugged by tracing their execution step by step.
<i>Database perspective</i>	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

The Oxygen XML Developer plugin XML perspective is used for editing the content of your XML documents.

Figure 4: Oxygen XML Developer plugin XML perspective



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

This perspective organizes the workspace in the following sections:

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 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 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 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 tagless editor).

The Outline View

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

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

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

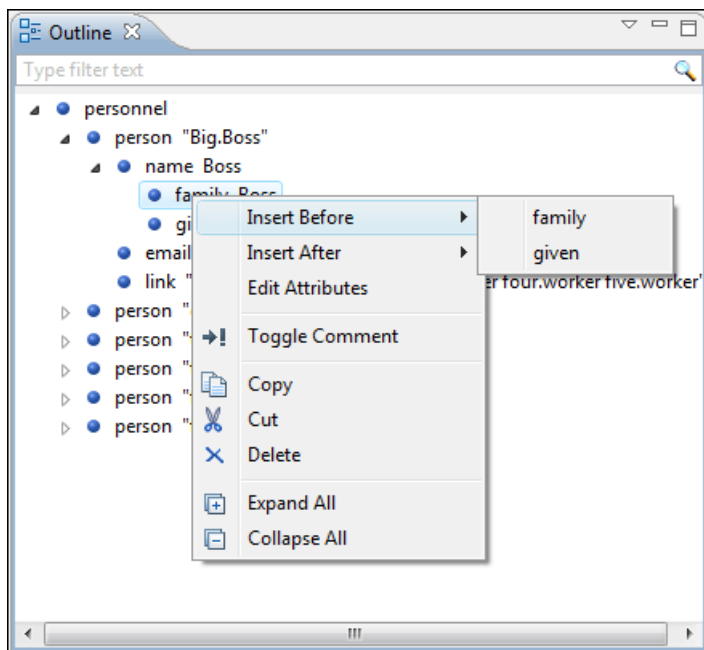


Figure 5: The Outline View

The Oxygen XML 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.

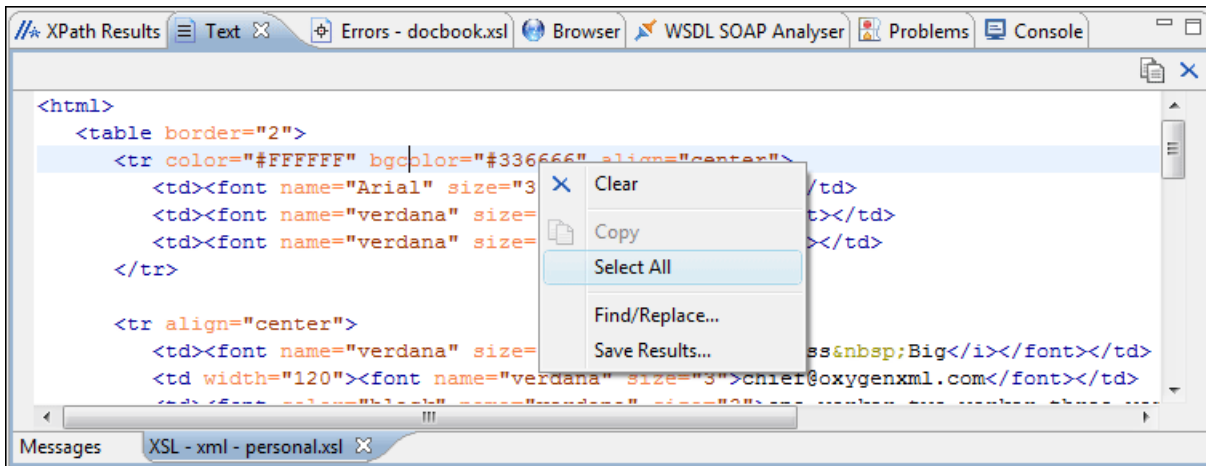


Figure 6: The Text View

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.

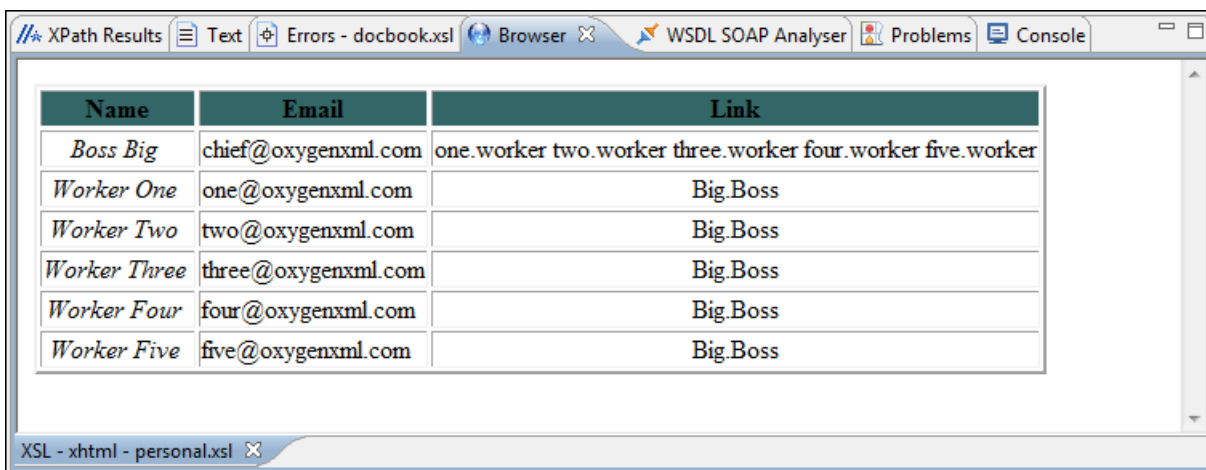


Figure 7: The Browser View

The Results View

The Results View displays the messages generated as a result of user actions like validations, transformations, search operations etc. Each message is a link to the location related to the event that triggered the message. Double clicking on 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](#)
- [Check Spelling in Files action](#)
- [Search References action](#)
- [XPath expression results](#)
- [SQL results](#)

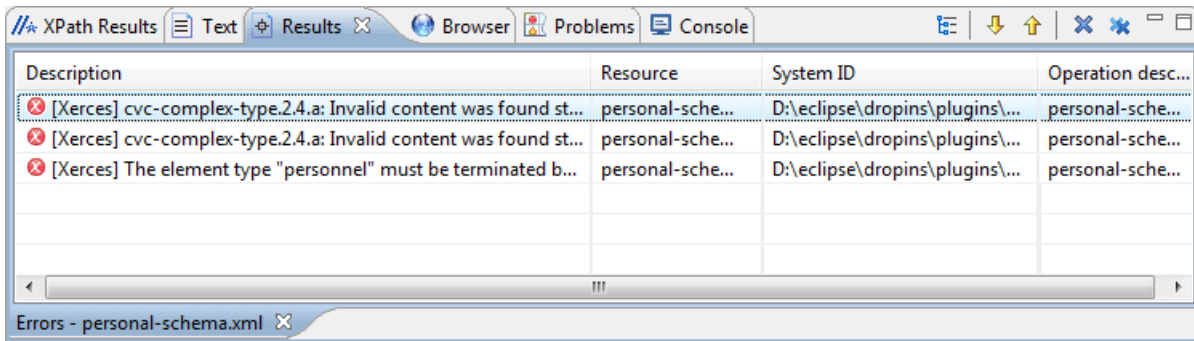


Figure 8: Errors View

The actions available on the toolbar of this view are:

- remove actions: **Remove selected**, **Remove all** - these actions reduce the number of the messages from the view by removing them. It is useful when the view is cluttered by messages that are not important.

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, etc),
 - 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

The Oxygen XML Developer plugin XPath Results view is automatically showed in the views pane of the Eclipse window to display XPath results.

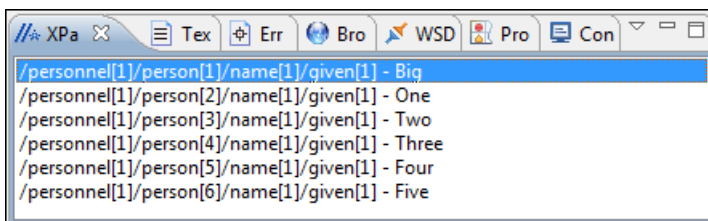
















Figure 9: The XPath Results View

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
-  - The Python documents icon
-  - The CSS documents icon
-

XSLT Debugger Perspective

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

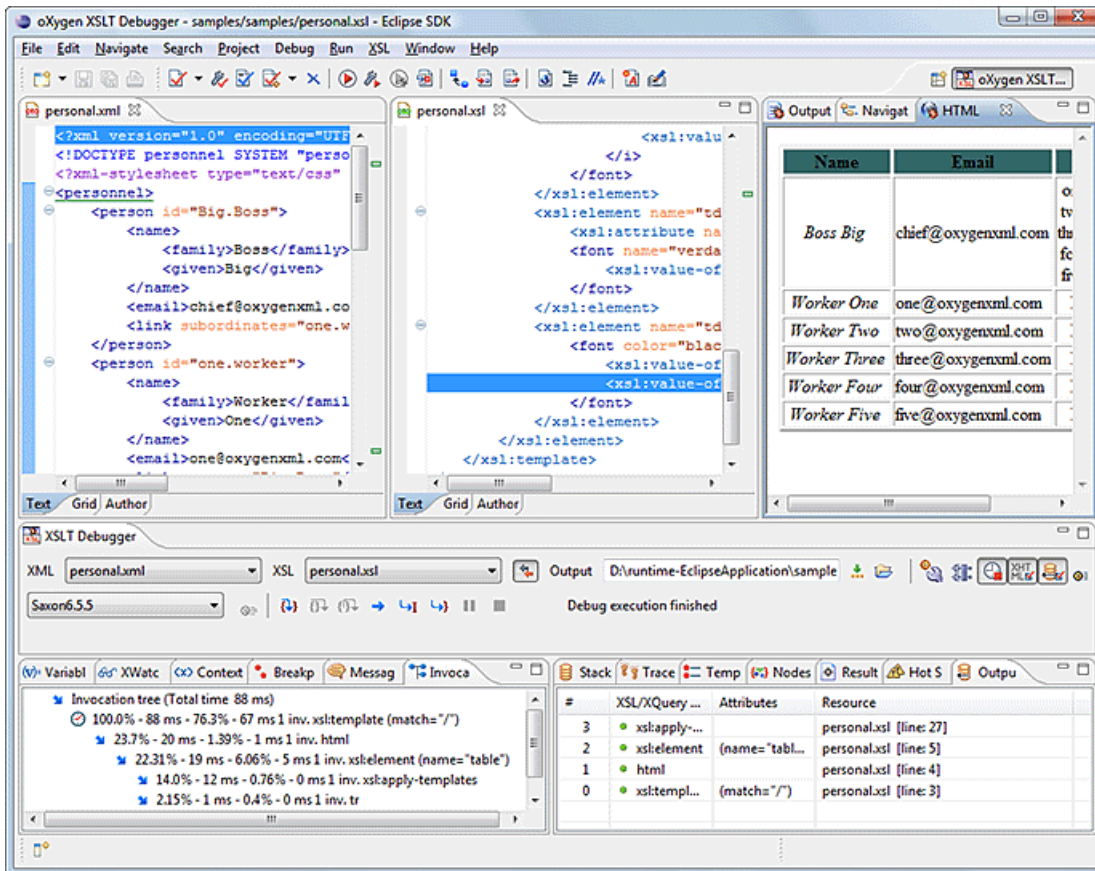


Figure 10: Oxygen XML Developer plugin - XSLT Debugger perspective

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

XQuery Debugger Perspective

The XQuery Debugger perspective is similar to *the XSLT Debugger perspective*. It is used to detect problems in an XQuery transformation process by executing the process step by step in a controlled environment and inspecting the information provided in the special views. The workspace is organized as follows:

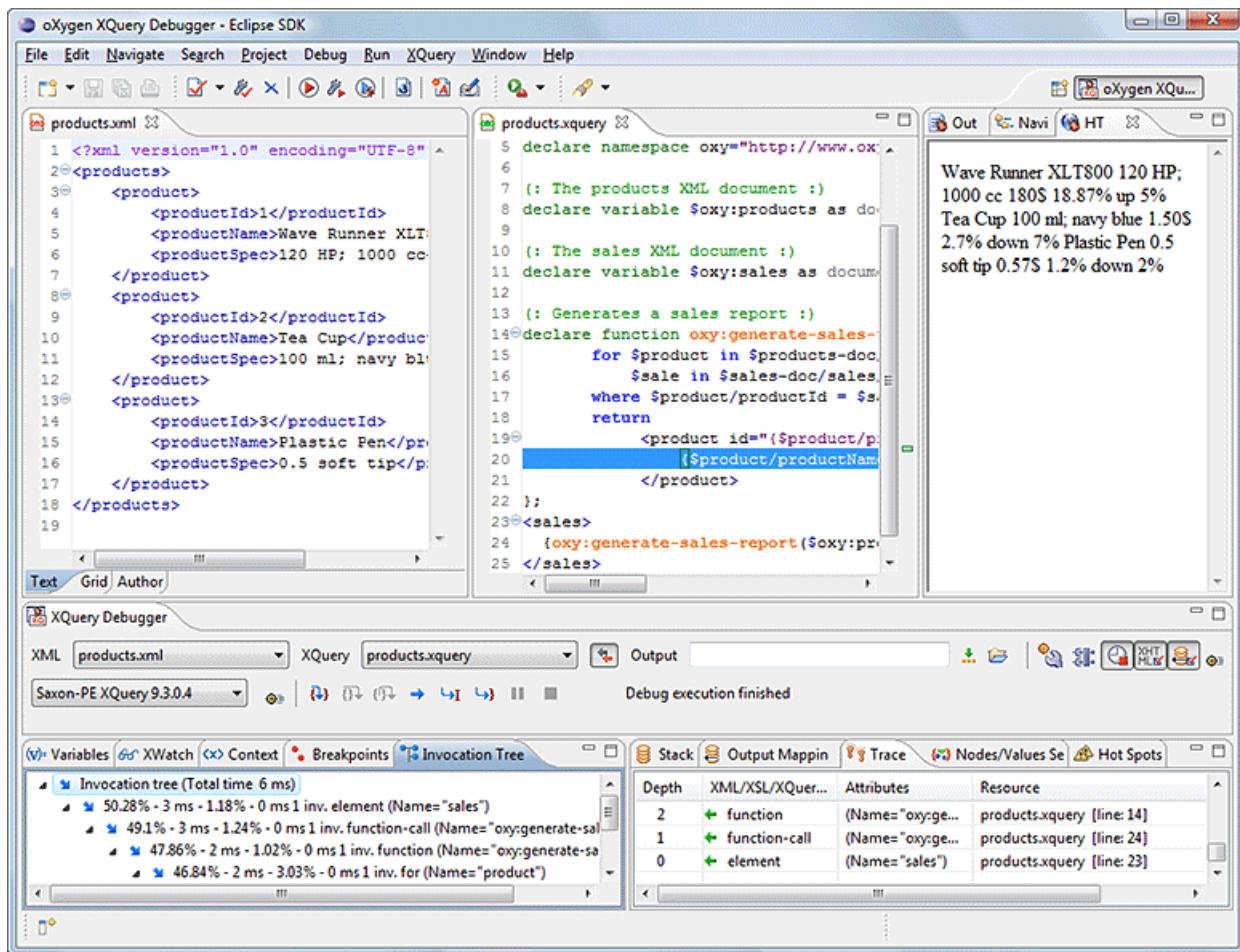


Figure 11: Oxygen XML Developer plugin XQuery Debugger perspective

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

Oxygen XML Developer plugin Database Perspective

The **Database** perspective is similar to the **Editor** perspective. It allows you to manage a database, offering support for browsing multiple connections at the same time, relational and native XML databases, SQL execution, XQuery execution and data export to XML.

This perspective offers database specific support for:

- Oracle Berkeley DB XML Database
- eXist XML Database
- IBM DB2 (Enterprise edition only)
- JDBC-ODBC Bridge

- MarkLogic (Enterprise edition only, XQuery support only)
- Microsoft SQL Server 2005 and Microsoft SQL Server 2008 (Enterprise edition only)
- MySQL
- Oracle 11g (Enterprise edition only)
- PostgreSQL 8.3 (Enterprise edition only)
- 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.

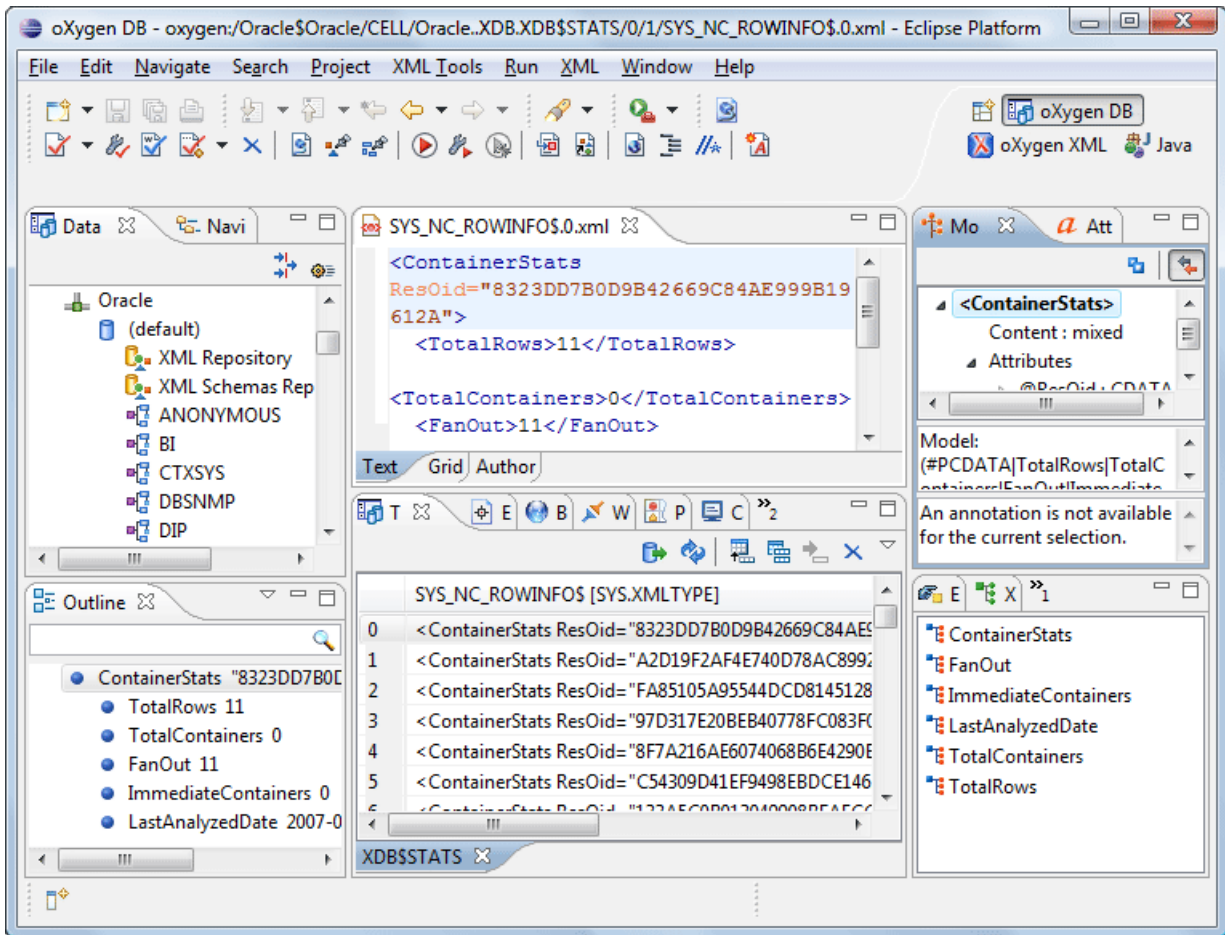


Figure 12: Database perspective

- 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 Documents

Topics:

- [Working with Unicode](#)
- [Opening and Closing Documents](#)
- [Grouping Documents in XML Projects](#)
- [Editing XML Documents](#)
- [Editing XML Schemas](#)
- [Editing Relax NG Schemas](#)
- [Editing NVDL Schemas](#)
- [Editing XSLT Stylesheets](#)
- [Editing XQuery Documents](#)
- [Editing CSS Stylesheets](#)
- [Editing JSON Documents](#)
- [Editing XProc Scripts](#)
- [Editing Schematron Schemas](#)
- [Handling Read-Only Files](#)


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

Working with Unicode

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

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

As a modern XML Editor, Oxygen 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 16bit 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: Windows XP/2003:** You can enable support for CJK (Chinese, Japanese, Korean) languages from **Control Panel / Regional and Language Options / Languages / Install files for East Asian languages**.

Opening and Saving Unicode Documents

On loading documents of the type XML, XSL, XSD, and DTD, Oxygen 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 Oxygen XML Developer plugin when the document is edited and saved.

Opening and Closing Documents

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


Creating New Documents

This section details the procedures available for creating new documents.

Oxygen XML Developer plugin Plugin Wizard for New Document

The New wizard only creates a skeleton document containing the document prolog, a root element and possibly other child elements depending on the options specific for each schema type. If you need to generate full and valid XML instance documents based on an XML Schema schema, you should use the [XML instance generation tool](#) instead.

The Oxygen XML Developer plugin installs a series of Eclipse wizards for easy creation of new documents. Using these wizards you let Oxygen XML complete details like 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. Select **File > New > -> Other (Ctrl+N) > oXygen** or press the  **New** toolbar button. 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 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 may 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 **Customize** was clicked, the following dialog is opened. Depending on the selected document type, different properties can be set:

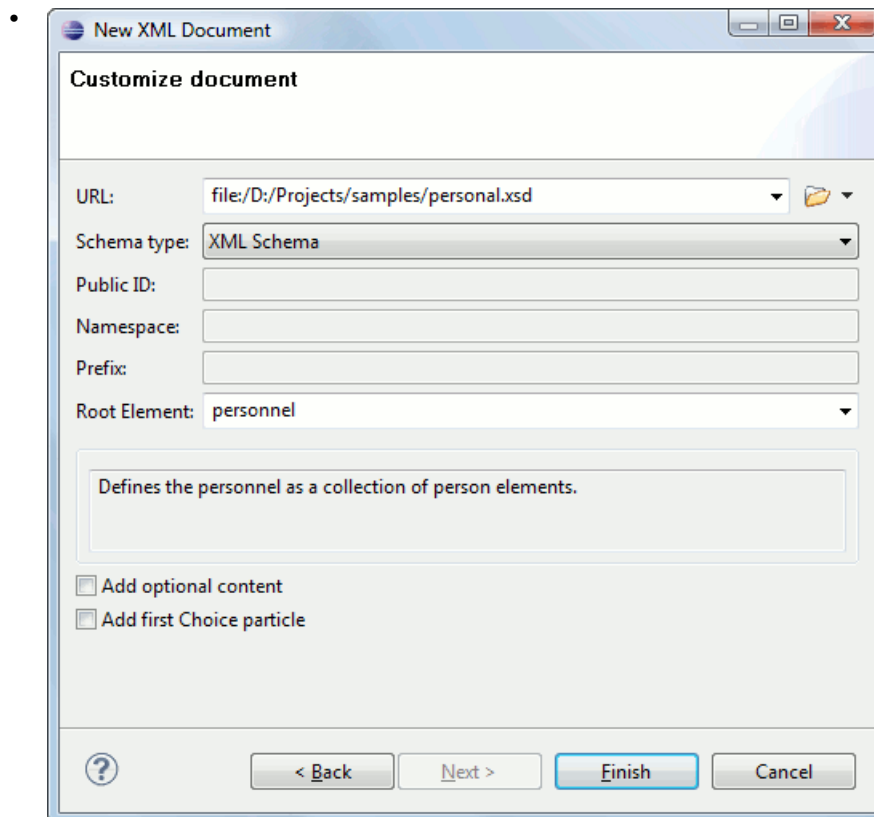


Figure 13: New XML Document Dialog

- **Schema URL** - Path to the schema file. When a file is selected, Oxygen XML Developer plugin analyzes its content and tries to fill-in the rest of the dialog;
- **Schema type** - The following options are available: XML Schema, DTD, RelaxNG XML syntax, RelaxNG compact syntax, and NVDL;
- **Public ID** - Specifies the PUBLIC identifier declared in the document prolog;
- **Namespace** - The document namespace;
- **Prefix** - The prefix for the namespace of the document root;
- **Root Element** - Populated with elements defined in the specified schema, enables selection of the element to be used as document root;

- **Description** - Shows a small description of the selected document root;
- **Add optional content** - When selected, the elements and attributes that are defined in the XML Schema as optional, are generated in the skeleton XML document;
- **Add first Choice particle** - When selected, the first element of an *xs:choice* schema element is generated in the skeleton XML document created in a new editor panel when the **OK** button is pressed.

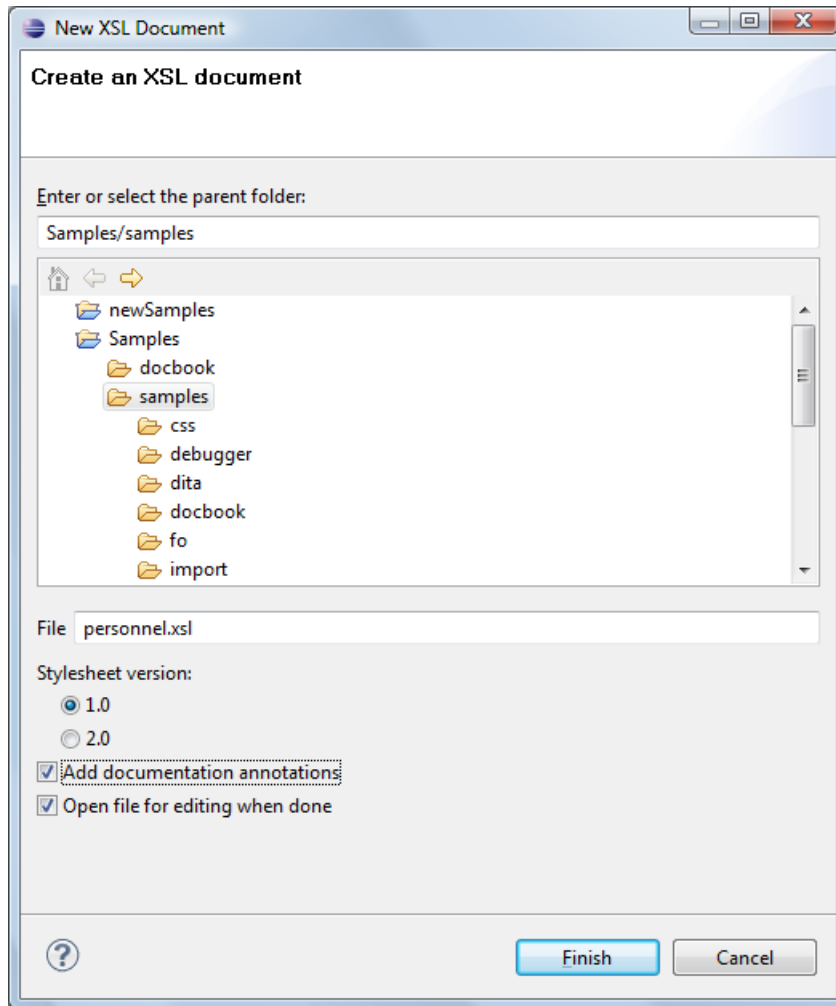


Figure 14: New XSL Document Dialog

- **Stylesheet version** - Stylesheet version number. Possible options: 1.0 and 2.0;
- **Add documentation annotations** - Adds annotation for XSL components.

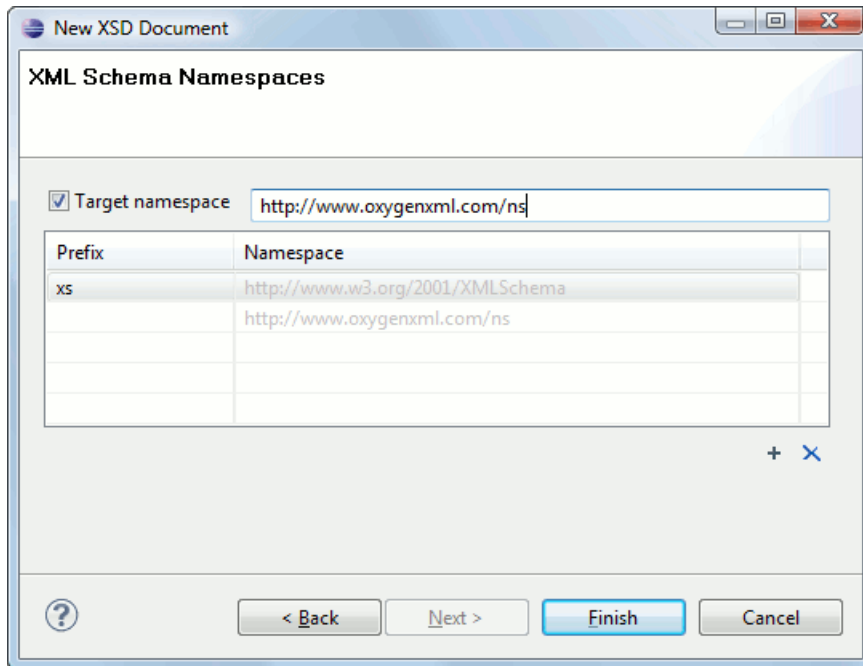


Figure 15: New XML Schema Document Dialog

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

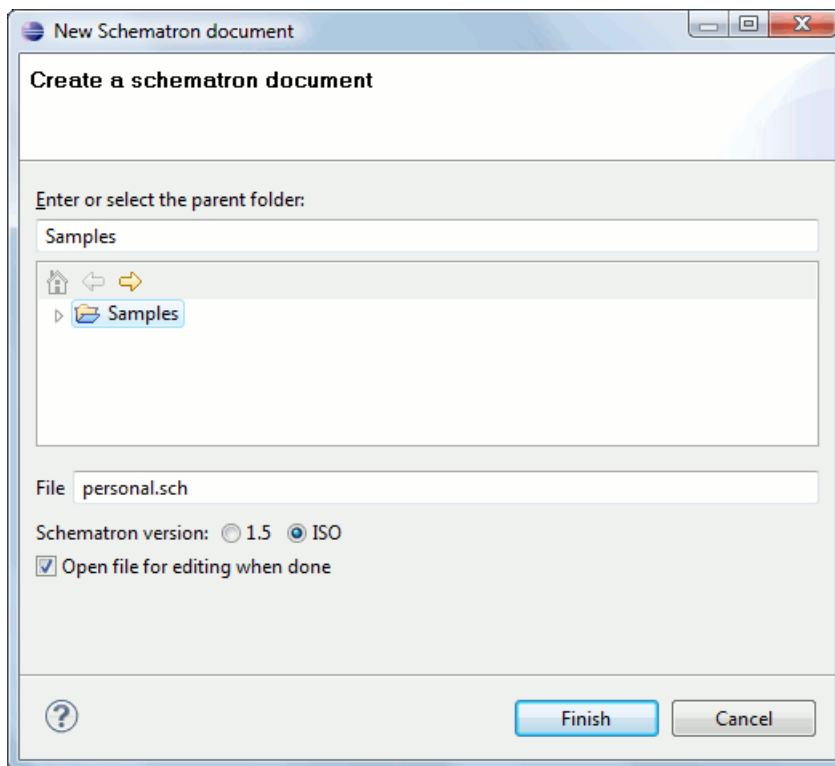


Figure 16: New Schematron Document Dialog

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

Creating Documents Based on Templates

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

The list of templates presented in the dialog includes:

- Document Types templates - Templates supplied with the defined document types.
 - User defined templates - The user can add template files in the `templates` folder of the Oxygen XML Developer plugin install directory. Also in the option page **Window > Preferences > oXygen > Editor > Templates > Document Templates** can be specified a custom templates folder to be scanned.
1. Go to menu **File > New > Other > oXygen > New From Templates**.
 2. Select a document type.
 3. Type a name for the new document and press the **Next** button.
 4. Press the **Finish** button.

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

Document Templates

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

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

Saving Documents

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


- **File > Save > (Ctrl+S)**.
- **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. The remote files can be edited exactly as the local ones, for example they can be added to a project, and can be subject to XSL and FO transformations.

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

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  button and then used in the *Data Source Explorer* view.

The Dialog

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

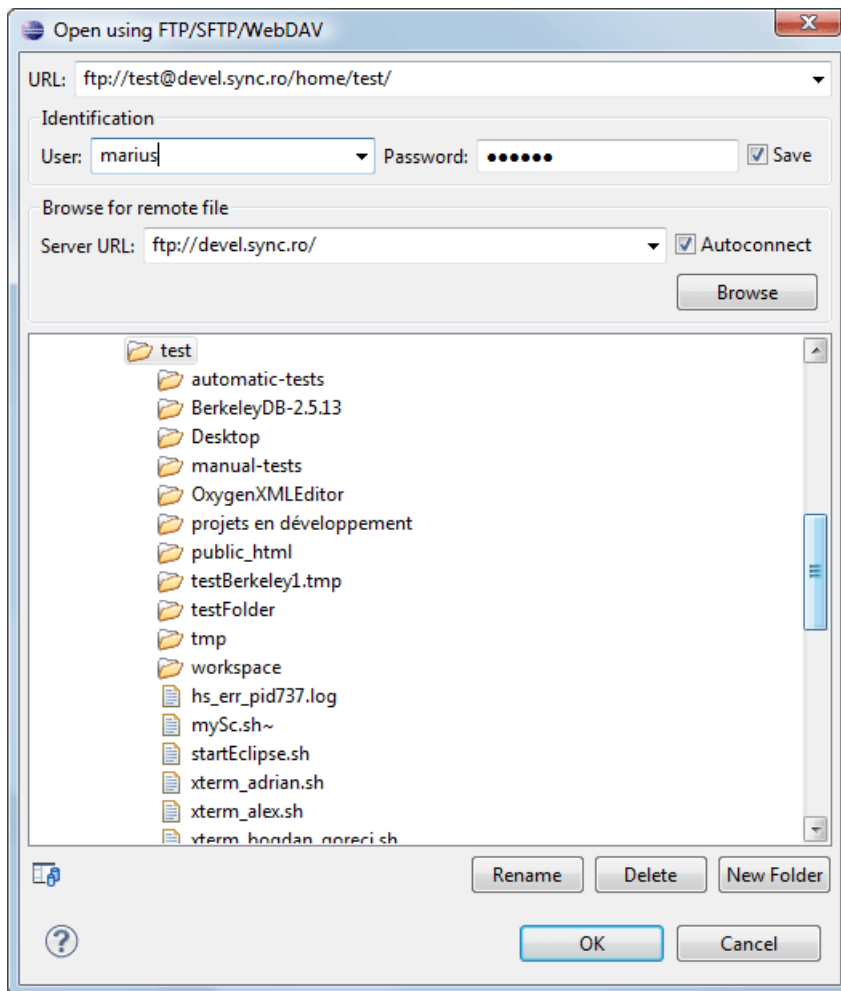


Figure 17: Open URL dialog

The displayed dialog is composed of several parts:


- The editable combo box, in which it can be specified directly the URL to be opened or saved.

 **Tip:**

You can type in here an URL like `ftp://anonymous@some.site/home/test.xml` if the file is accessible through anonymous FTP.

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

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

 **Note:**

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

- The *Browse for remote file* section contains the server combo and the **Autoconnect** check box. Into the server combo it may be specified the protocol , the name or IP of the server .

**Tip:****Server URLs**

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

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

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

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

Changing File Permissions on a Remote FTP Server

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

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

WebDAV over HTTPS

If you want to access a WebDAV repository across an insecure network, Oxygen XML 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.

How to Add a HTTPS Server Certificate to Oxygen XML Developer plugin

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

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

If this is your first visit to the repository it will be displayed a security alert stating that the security certificate presented by the server is not trusted.

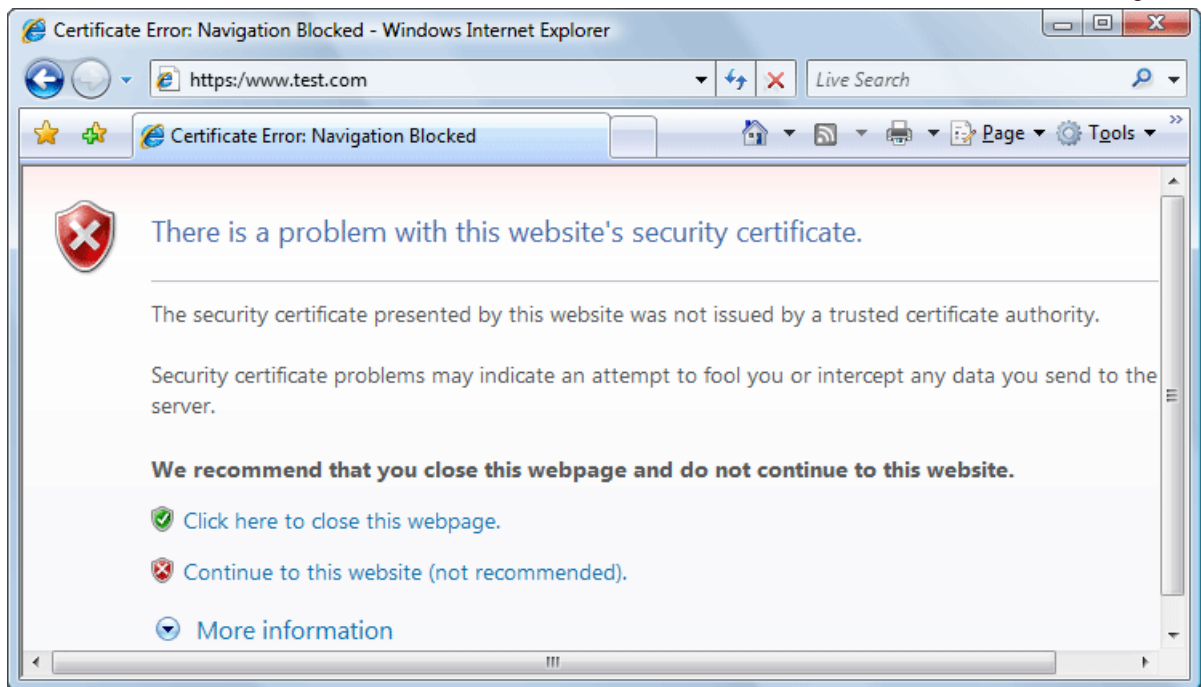


Figure 18: Security alert - untrusted certificate

- b) Go to menu **Tools > Internet Options**.
Internet Options dialog is opened.
 - c) Select **Security** tab.
 - d) Select **Trusted sites** icon.
 - e) Press **Sites** button.
This will open **Trusted sites** dialog.
 - f) Add repository URL to **Websites** list.
 - g) Close **Trusted sites** dialog and **Internet Options** dialog.
 - h) Try again to connect to the same repository URL in Internet Explorer.
The same error page as above will be displayed.
 - i) Select **Continue to this website** option.
A clickable area with a red icon and text **Certificate Error** is added to Internet Explorer address bar.
 - j) Click on **Certificate Error** area.
A dialog containing **View certificates** link is displayed.
 - k) Click on **View certificates** link.
Certificate dialog is displayed.
 - l) Select **Details** tab of **Certificate** dialog.
 - m) Press **Copy to File** button.
Certificate Export Wizard is started.
 - n) Follow indications of wizard for DER encoded binary X.509 certificate. Save certificate to local file server .cer.
2. Import the local file into the JRE running Oxygen XML Developer plugin .
- a) Open a text-mode console.
 - b) Go to the `lib/security` subfolder of your JRE directory, that is of the directory where it is installed the JRE running Oxygen XML Developer plugin . You find the home folder of the JRE in the `java.home` property that is displayed in the About dialog tab.
 - c) Run the following command:

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

The `local-file.cer` file contains the server certificate, created during the previous step. **keytool** requires a password before adding the certificate to the JRE keystore. The default password is *changeit*. If somebody changed the default password then he is the only one who can perform the import. As a workaround you can delete the `cacerts` file, re-type the command and enter as password any combination of at least 6 characters. This will set the password for future operations with the key store.

3. Restart Oxygen XML Developer plugin .

Opening the Current Document in System Application

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

Closing Documents

To close documents use one of the following methods:

- Go to menu **File > Close (Ctrl+F4)** : Closes only the selected tab. All other tab instances remain opened.
- Go to menu **File > Close All (Ctrl+Shift+F4)**: Closes all open documents. If a document is modified or has no file, a prompt to save, not to save, or cancel the save operation is displayed.
- Select the item **Close** from the contextual menu of an editor tab: Closes the selected editor.
- Select the item **Close Other Files** from the contextual menu of an editor tab: Closes the other files except the selected tab.
- Select the item **Close All** from the contextual menu of an editor tab: Closes all open editors within the panel.

Viewing File Properties

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

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

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



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

Grouping Documents in XML Projects

This section explains how to create and work with projects.

Creating a New Project




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

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

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

Creating New Project Items

A series of actions are available in the contextual menu:

- **New > File** - Creates a new file and adds it to the project structure.
-  **Add Folder** - Adds a link to a physical folder, whose name and content mirror a real folder existing in the file system on disk. The icon of this action is different on Mac OS X () as the standard folder icon on Mac OS X is not the usual one from Windows and Unix/Linux systems.
- **New > Logical Folder** - Creates a logical folder in the tree structure (the icon is a magenta folder on Mac OS X - )
- **New > Logical Folders from Web** - Replicates the structure of a remote folder accessible over FTP/SFTP/WebDAV, as a structure of logical folders. The newly created logical folders contain the file structure of the folder it points to.
- **New > Project** - Creates a new project, after closing the current project and all open files.

Managing Project Content


Validate Files

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

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

Applying Transformation Scenarios

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

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

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

Other Context-Dependent Actions

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

- **Open with** - Open selected file with one of internal tools: , , , WSDL/SOAP Analyzer, , .
- **Open All Files** - Action available only when at least one folder is selected. Opens in the editor view all files contained by the selected resources.

Create an Oxygen XML Developer plugin XML Project

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

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.

Supported Schema Types for XML Documents


The supported schema types are:

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


Setting a Default Schema

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

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

 **Note:** If a DTD schema is specified in the document, the content completion for Author mode is based on this schema (even if there is already one detected from the validation scenario);

- detected from the document type that matches the edited document - Each document type available in *Document Type Association* preferences page contains a set of rules for associating a schema with the current document.

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

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

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

**Important:**

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

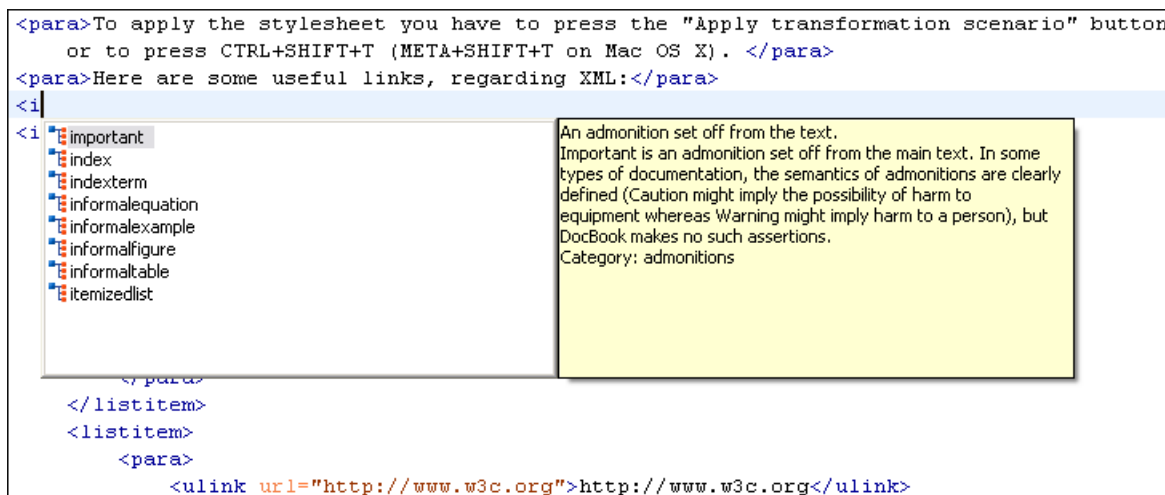


Figure 19: Content Completion Driven by DocBook DTD

Making the Schema Association Explicit in the XML Instance Document

The schema used by the *content completion* assistant and *document validation* engine can be associated with the document using the **Associate Schema** action. For most of the schema types, it uses *the xml-mode 1 processing instruction*, the exceptions being:

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

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

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

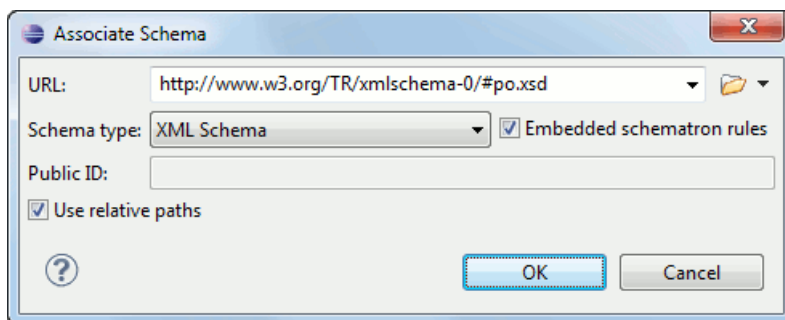


Figure 20: The Associate Schema Dialog

The following options are available:

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

- **Public ID** - Specify a public ID if you have selected a DTD.
- **Embedded schematron rules** - if you have selected XML Schema or Relax NG schemas with embedded Schematron rules, enable this option.
- **Use relative paths** - enable this option if the XML instance document and the associated schema contain relative paths. The location of the schema file is inserted in the XML instance document as a relative file path. This practice allows you, for example, to share these documents with other users, without running into problems caused by different project locations on physical disk.

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

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

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

Associating a Schema With the Namespace of the Root Element

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

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

The xml-model Processing Instruction

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


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

It is available in the *content completion* assistant, before XML document root element and has the following attributes:

- `href` - schema file location. Mandatory attribute.
- `type` - content type of schema. Optional attribute with the following possible values:
 - for DTD the recommended value is `application/xml-dtd`;
 - for W3C XML Schema the recommended value is `application/xml` or can be left unspecified;
 - for RELAX NG the recommended value is `application/xml` or can be left unspecified;
 - for RELAX NG - compact syntax the recommended value is `application/relax-ng-compact-syntax`;
 - for Schematron the recommended value is `application/xml` or can be left unspecified;
 - for NVDL the recommended value is `application/xml` or can be left unspecified.
- `schematypens` - namespace of schema language of referenced schema with the following possible values:
 - for DTD - not specified;
 - for W3C XML Schema the recommended value is `http://www.w3.org/2001/XMLSchema`;
 - for RELAX NG the recommended value is `http://relaxng.org/ns/structure/1.0`;
 - for RELAX NG - not specified;
 - for Schematron the recommended value is `http://purl.oclc.org/dsdl/schematron`;
 - for NVDL the recommended value is `http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0`.
- `phase` - phase name of validation function in Schematron schema. Optional attribute.
- `title` - title for associated schema. Optional attribute.

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

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

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

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 menu **XML > Learn Structure (Ctrl+Shift+L)**.
The **Learn Structure** action reads the mark-up structure of the current document. The **Learn completed** message is displayed in the application's status bar when the action is finished.
3. Go to menu **XML > Save Structure (Ctrl+Shift+S)**. Enter the DTD file path.
4. Press the *Save* button.

Streamline with Content Completion

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

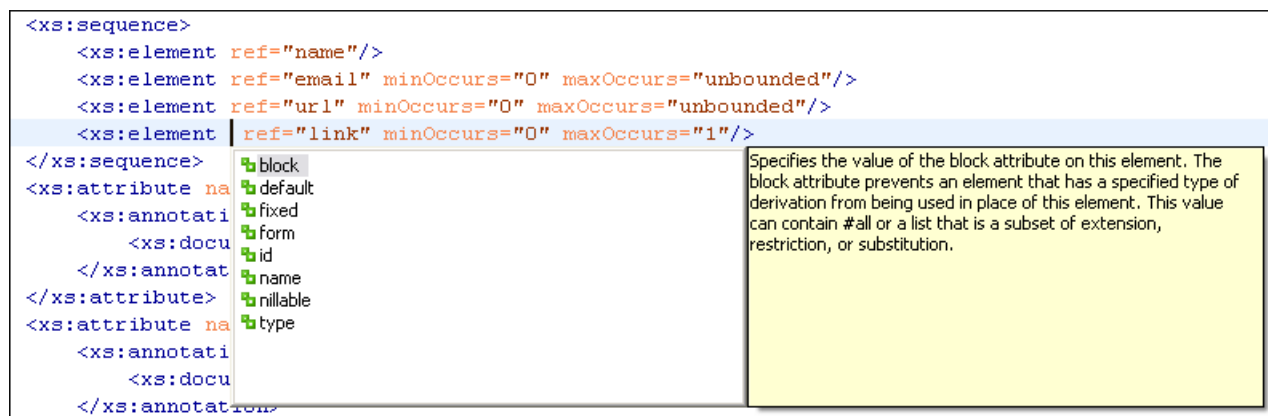


Figure 21: Content Completion Assistant


Oxygen XML Developer plugin logs the URL of the detected schema in the [Status view](#).

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

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

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

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

 **Note:** When the DTD, XML Schema or RELAX NG schema specifies required child elements for the newly added element, they will be inserted automatically only if the Add Element Content option (found in **Preferences > Editor > Content Completion** options page) is enabled. The Content Completion assistant can also add optional content and first choice particle, as specified in the DTD or XML Schema or RELAX NG schema, for the element if these two options are enabled.

After inserting the element, the cursor will be positioned:

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

The content assistant can be started at any time by pressing CTRL+Space. The effect is that the context-sensitive list of proposals will be shown in the caret's current position if element, attribute or attribute value insertion makes sense. The Content Completion assistant is displayed:

- anywhere within a tag name or at the beginning of a tag name in an XML document, XML Schema, DTD or Relax NG (full or compact syntax) schema;
- anywhere within an attribute name or at the beginning of an attribute name in any XML document with an associated schema;
- within attribute values or at the beginning of attribute values in XML documents where lists of possible values have been defined for that element in the schema associated with the document

The items that populate the Content Completion assistant are dependent on the element structure specified in the DTD, XML Schema, Relax NG (full or compact syntax) schema, or NVDL schema associated to the edited document.

The number and type of elements displayed by the assistant is dependent on the cursor's current position in the structured document. The child elements displayed within a given element are defined by the structure of the specified DTD, XML Schema, Relax NG (full or compact syntax) schema, or NVDL schema. All elements that can't be child elements of the current element according to the specified schema are not displayed.

If only one element name must be displayed by the content assistant then the assistant is not displayed anymore but this only option is automatically inserted in the document at the current cursor position.

A schema may declare certain attributes as ID or IDREF/IDREFS. When the document is validated, oXygen XML 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 treated as ID attributes and collected and displayed by the Content Completion assistant as possible values for *anyURI* attributes defined in the schema of the edited document. This works only for XML Schema and Relax NG schemas.

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

The operation of the Content Completion assistant is configured by the options available in the options group called [Content Completion](#).

Set Schema for Content Completion

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

- the schema specified explicitly in the document. In this case Oxygen 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>
    <value>value1</value>
    <value>value2</value>
    <value>value3</value>
  </choice>
</element>
```

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

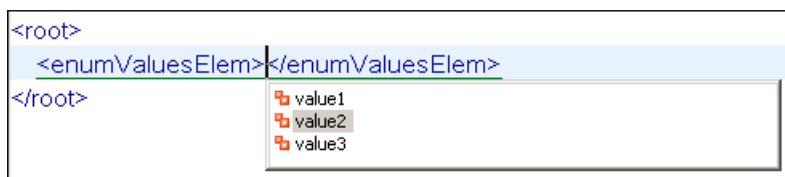


Figure 22: Content Completion assistant - element values in Relax NG documents

Schema Annotations

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

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

```
<xs:annotation>
  <xs:documentation>
```

```

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

```

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

When editing a Schematron schema the content completion assistant displays XSLT 1.0 functions and optionally XSLT 2.0 functions in the attributes *path*, *select*, *context*, *subject*, *test* depending on *the Schematron options* that are set in Preferences. If the Saxon 6.5.5 namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.3.0.5 namespace is declared in the Schematron schema (`xmlns:saxon="http://saxon.sf.net/"`) the content completion displays also the XSLT Saxon extension functions as in the following figure:

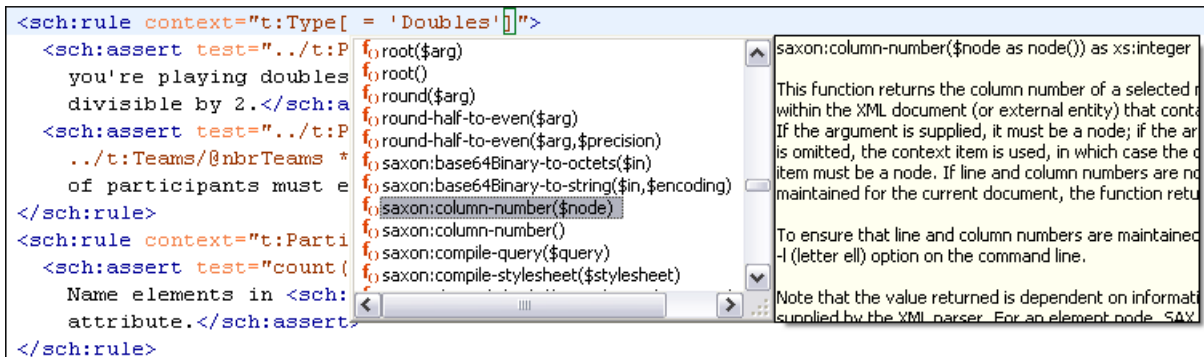


Figure 23: XSLT extension functions in Schematron schemas documents

In a Relax NG schema any element outside the Relax NG namespace (<http://relaxng.org/ns/structure/1.0>) is handled as annotation and the text content is displayed in the annotation window together with the content completion window:

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

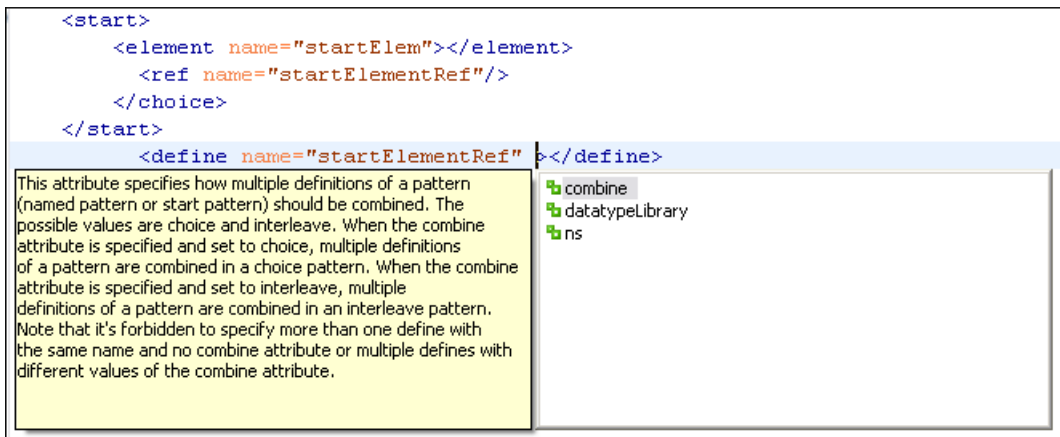


Figure 24: Schema annotations displayed at Content Completion

The following HTML tags are recognized inside the text content of an XML Schema annotation: `p`, `br`, `ul`, `li`. They are rendered as in an HTML document loaded in a web browser: `p` begins a new paragraph, `br` breaks the current line, `ul` encloses a list of items, `li` encloses an item of the list.

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

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

Content Completion Helper Views

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

The Model View

The Model view presents the structure of the current 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 > Model view**

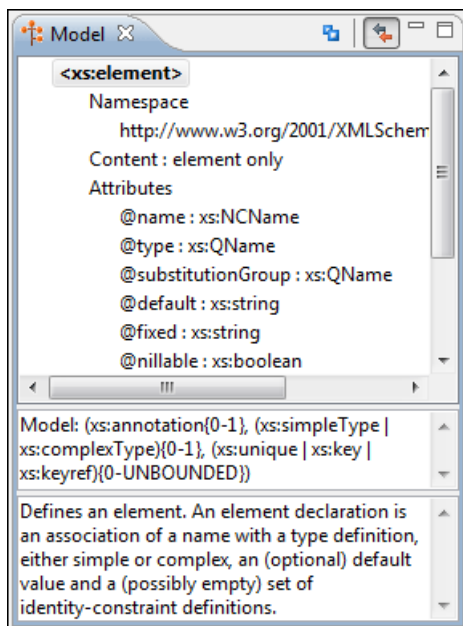


Figure 25: The Model View

The Element Structure Panel

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

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

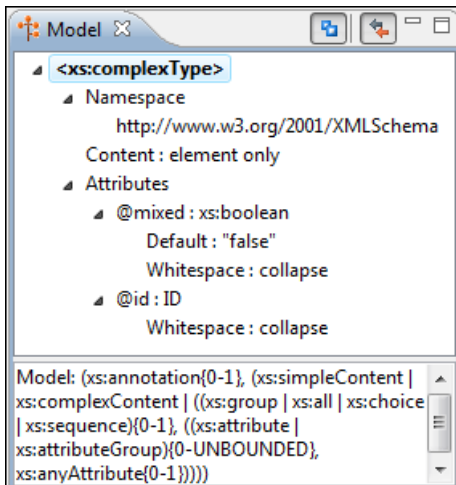


Figure 26: The Element Structure Panel

The Annotation Panel

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

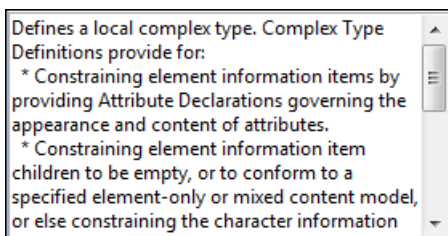


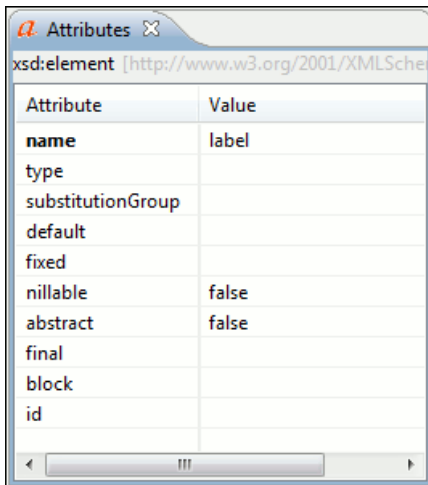
Figure 27: The Annotation panel

The Attributes View

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

The attributes present in the document are painted in the **Attributes View** with a bold font. 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 where 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.



Attribute	Value
name	label
type	
substitutionGroup	
default	
fixed	
nillable	false
abstract	false
final	
block	
id	

Figure 28: The Attributes View

The Elements View

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

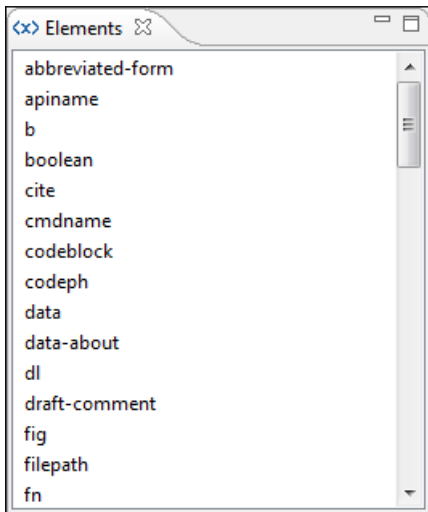


Figure 29: The Elements View

The Entities View

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

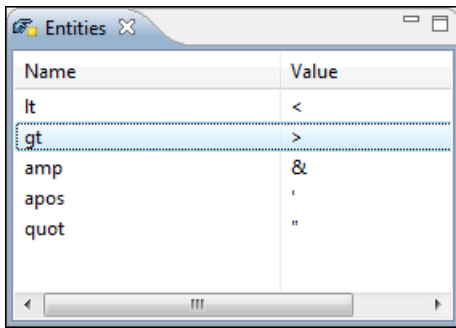


Figure 30: The Entities View

Code Templates

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

Oxygen XML Developer plugin comes with a lot of predefined code templates but you can *define* your own code templates for any type of editor. For more details see the [example for XSLT editor code templates](#).

To obtain the template list you can use the Content Completion on request shortcut key (usually CTRL-SPACE) or the Code Templates on request shortcut key (CTRL-SHIFT-SPACE). The first shortcut displays the code templates in the same [content completion list with elements from the schema of the document](#). The second shortcut displays only the code templates and is the default shortcut of the action **Document > Content Completion > Show Code Templates**.

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

- **`\${caret}** - The position where the caret is inserted. This variable can be used in a or in a
- **`\${selection}** - The XML content of the current selection in the editor panel. This variable can be used in a
- **`\${ask('user-message', param-type, 'default-value' ?)}** - To prompt for values at runtime, use the *ask('user-message', param-type, 'default-value' ?)* editor variable. The following parameters can be set:
 - 'user-message' - the actual message to be displayed. Note the quotes that enclose the message.
 - param-type - optional parameter. Can have one of the following values:
 - url - input is considered to be an URL. Oxygen XML Developer plugin checks that the URL is valid before passing it to the transformation.
 - password - input characters are hidden.
 - generic - the input is treated as generic text that requires no special handling.
 - 'default-value' - optional parameter. Provides a default value in the input text box.

Examples:

- ``${ask('message')}` - Only the message displayed for the user is specified.
 - ``${ask('message', generic, 'default')}` - 'message' will be displayed for the user, the type is not specified (the default is string), the default value will be 'default'.
 - ``${ask('message', password)}` - 'message' will be displayed for the user, the characters typed will be replaced with a circle character.
 - ``${ask('message', password, 'default')}` - Same as above, default value will be 'default'.
 - ``${ask('message', url)}` - 'message' will be displayed for the user, the type of parameter will be URL.
 - ``${ask('message', url, 'default')}` - Same as above, default value will be 'default'.
- **`\${timeStamp}** - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
 - **`\${uuid}** - Universally unique identifier.
 - **`\${id}** - Application-level unique identifier.

- `${cfn}` - Current file name without extension and without parent folder.
- `${cfne}` - Current file name with extension.
- `${cf}` - Current file as file path, that is the absolute file path of the current edited document.
- `${cfd}` - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- `${frameworksDir}` - The path (as file path) of the `frameworks` subfolder of the Oxygen XML Developer plugin installation folder.
- `${pd}` - Current project folder as file path.
- `${oxygenInstallDir}` - Oxygen XML Developer plugin installation folder as file path.
- `${homeDir}` - The path (as file path) of the user home folder.
- `${pn}` - Current project name.
- `${env(VAR_NAME)}` - Value of the `VAR_NAME` environment variable.
- `${system(var.name)}` - Value of the `var.name` system variable.
- `${date(pattern)}` - Current date. Follows the given pattern. Example: yyyy-MM-dd.

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.

If you select menu **Document > Validate > Check Well-Formedness** (**Alt+Shift+W W (Cmd+Alt+W W on Mac)**) or click the toolbar button  **Check Well-Formedness** Oxygen XML Developer plugin checks if your document is *Namespace Well-Formed XML*. If any error is found the result is returned to the message panel. Each error is one record in the result list and is accompanied by an error message. Clicking the record will open the document containing the error and highlight its approximate location.

A not Well-Formed XML Document

```
<root><tag></root>
```

When **Check Well-Formedness** is performed the following error is raised:

```
The element type "tag" must be terminated by the matching end-tag
"/tag>"
```

To resolve the error, click in the result list record which will locate and highlight the errors approximate position. Identify which start tag is missing an end tag and insert `</tag>`.

A not namespace-wellformed document

```
<x:y></x:y>
```

When **Check document form** is performed the following error is raised:


```
Element or attribute do not match QName production:
QName ::= (NCName ':' )?NCName.
```

A not namespace-valid document

```
<x:y></x:y>
```

When **Check document form** is performed the following error is raised:


```
The prefix "x" for element "x:y" is not bound.
```

Also the files contained in the current project and selected with the mouse in *the Project view* can be checked for well-formedness with one action available on the popup menu of the Project view in the **Validate** submenu:  **Check Well-Formedness**.

Validating XML Documents Against a Schema

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

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

The Oxygen XML Developer plugin  **Validate document** function ensures that your document is compliant with the rules defined by an associated DTD, XML Schema, Relax NG, or Schematron schema. XML Schema or Relax NG Schema can embed Schematron rules. For Schematron validations, it is possible to select the validation phase.

Marking Validation Errors and Warnings

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

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

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

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

Status messages from every validation action are logged into the [Console view](#).

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

Validation Example - A DocBook Validation Error

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
"http://www.docbook.org/xml/4.4/docbookx.dtd">
<article>
  <title>Article Title</title>
  <sect1>
    <title>Section1 Title</title>
    <itemizedlist>
      <listitem>
        <link>a link here</link>
      </listitem>
    </itemizedlist>
  </sect1>
</article>
```


The **Validate Document** action will return the following error:

Unexpected element "link". The content of the parent element type must match "(calloutlist|glosslist|bibliolist|itemizedlist|orderedlist|segmentedlist|simplelist|variablelist|caution|important|note|tip|warning|literallayout|programlisting|programlistingco|screen|screenco|screenshot|synopsis|cmdsynopsis|funcsynopsis|classsynopsis|fieldsynopsis|constructorsynopsis|destructorsynopsis|methodsynopsis|formalpara|para|simpara|address|blockquote|graphic|graphicco|mediaobject|mediaobjectco|informalequation|informalexample|informalfigure|informaltable|equation|example|figure|table|msgset|procedure|sidebar|qandaset|task|anchor|bridgehead|remark|highlights|abstract|authorblurb|epigraph|indexterm|beginpage)+".

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

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

Caching the Schema Used for Validation

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

Automatic Validation

Oxygen 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*.

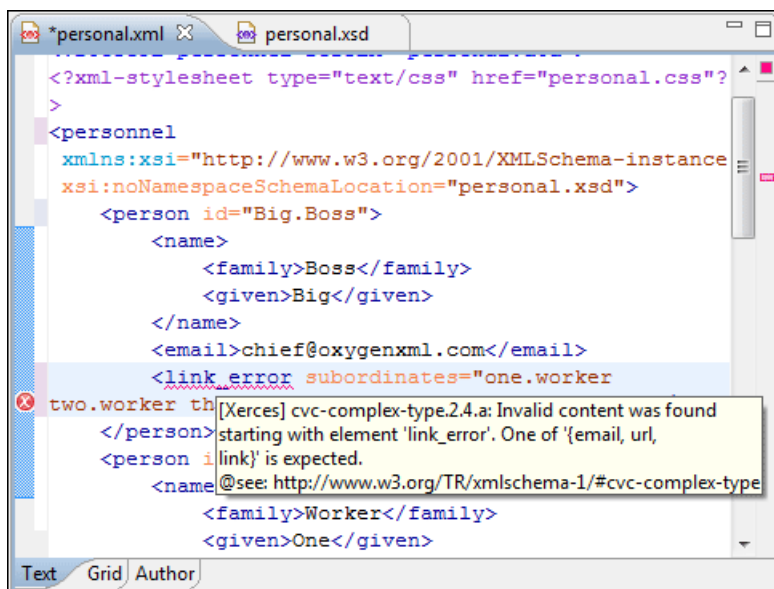


Figure 31: Automatic Validation on the Edited Document

Custom Validators

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

Some validators are configured by default but they are third party processors which do not support the *output message format* of Oxygen XML 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 (`--xinclue`) are enabled by default in the preconfigured LIBXML validator. The `--postvalid` parameter

is also set by default which allows LIBXML to validate correctly the main document even if the XInclude fragments contain IDREFS to ID's located in other fragments.

For validation against an external DTD specified by URI in the XML document, the parameter `--dtdvalid ${ds}` must be added manually to the DTD validation command line. `${ds}` represents the detected DTD declaration in the XML document.



Caution: Known problem: file paths containing spaces are not handled correctly in the LIBXML processor. For example the built-in XML catalog files of the predefined document types (DocBook, TEI, DITA, etc) are not handled by LIBXML if Oxygen 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 Mac OS X if the full path to the LIBXML executable file is not specified in the **Executable path** text field, some errors may occur on validation against a W3C XML Schema like:

Unimplemented block at ... xmlschema.c

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

- **Saxon SA** - Included in Oxygen 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 specification or according to the W3C XML Schema 1.1 specification. 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-install-folder]/xsv` installation folder. If it is installed in a different folder the predefined executable path must be *corrected in Preferences*. It is associated to XML Editor and XSD Editor. It is able to validate the edited document against XML Schema or a custom schema type.
- **SQC (Schema Quality Checker from IBM)** - Not included in Oxygen XML 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-install-folder]/sqc`. If it is installed in a different folder the predefined executable path and working directory must be *corrected in the Preferences page*. It is associated to XSD Editor.

Linked Output Messages of an External Engine

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

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

- Description: message content (the string *Description:* followed by the content of the message that will be displayed in the output view).

Validation Scenario

A complex XML document is usually split in smaller interrelated modules which do not make much sense individually and which cannot be validated in isolation due to interdependencies with the other modules. A mechanism is needed to set the main module of the document which in fact must be validated when an imported module needs to be checked for errors.

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

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

A second benefit of a validation scenario is that the stylesheet can be validated with several engines to make sure that it can be used in different environments with the same results. For example an XSLT stylesheet needs to be applied with Saxon 6.5, Xalan and MSXML 4.0 in different production systems.

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

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

How to Create a Validation Scenario

Follow these steps for creating a validation scenario:

1. Open the **Configure Validation Scenario** dialog from menu **XML** or from the toolbar of the Oxygen XML Developer plugin plugin.

The following dialog is displayed. It contains the following types of scenarios:

- **Default validation** scenario validates the input using Oxygen XML Developer plugin default validation options that apply to the type of the current document;
- **Predefined** scenarios are organized in categories depending on the type of file they apply to and can be easily identified by a yellow key icon that marks them as *read-only*. If the predefined scenario is the framework's default scenario its name is written in bold font. If you try to edit one of these scenarios, Oxygen 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.

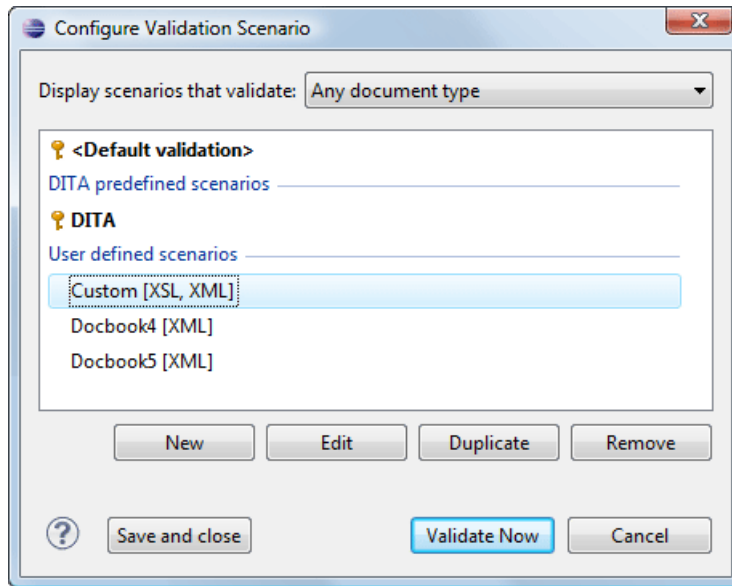


Figure 32: Configure Validation Scenario

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

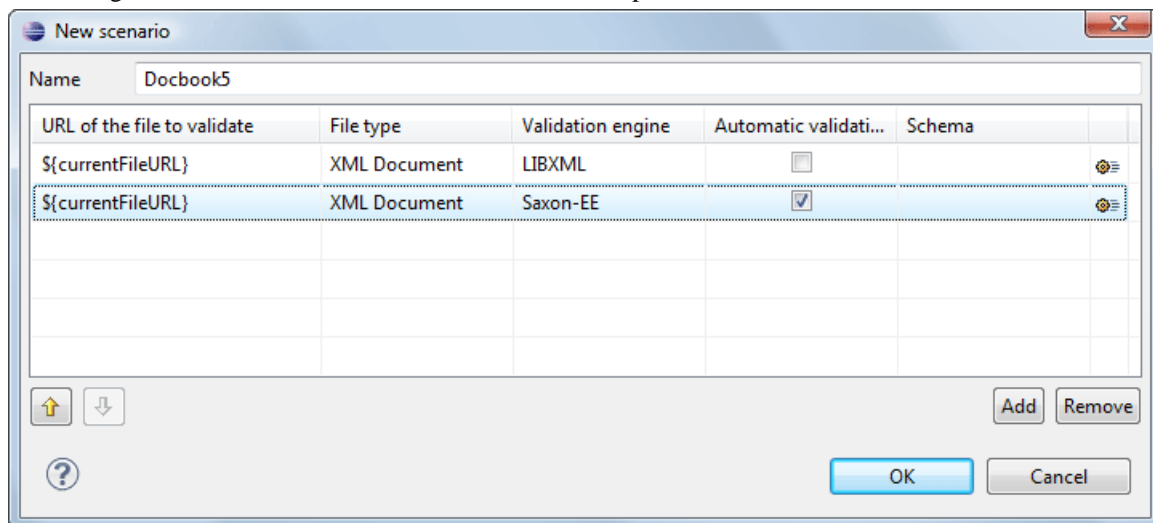


Figure 33: Add / Edit a Validation Unit

The table holds the following information:

- **URL of the file to validate** - The URL of the main module which includes the current module. It is also the entry module of the validation process when the current one is validated.
- **File type** - The type of the document validated in the current validation unit. Oxygen 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 means that the validation is done by the default engine set in Preferences pages for the type of the current document (XML document, XML Schema, XSLT stylesheet, XQuery file, etc) instead of a validation scenario.
- **Automatic validation** - If this option is checked, then the validation operation defined by this row of the table is applied also by If the **Automatic validation** feature is then this option does not take effect as the Preference setting has higher priority.

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

4. Edit the URL of the main validation module.

Specify the URL of the main module:

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

```

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

```

Figure 34: Insert an Editor Variable

5. Select the type of the validated document.

Note that it determines the list of possible validation engines.





6. Select the validation engine.

7. Select the **Automatic validation** option if you want to validate the current unit when

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

Validation Actions in the User Interface

Use one of the actions for validating the current document:

- Select menu **XML > Validate Document (Alt+Shift+V V) ((Cmd+Alt+V V on Mac OS))** or click the button  **Validate Document** available in the **Validate** toolbar. This action returns an error list in the message panel. Mark-up of current document is checked to conform with the specified DTD, XML Schema or Relax NG schema rules. It caches the schema and the next execution of the action uses the cached schema.
- Select menu **XML > Reset Cache and Validate** or click the button  **Reset Cache and Validate** available in the **Validate** toolbar to reset the cache with the schema and validate the document. This action also parses again the XML catalogs and reset the schema used for content completion. It returns an error list in the message panel. Mark-up of current document is checked to conform with the specified DTD, XML Schema or Relax NG schema rules.
- Select menu **XML > Validate with (Alt+Shift+V E) ((Cmd+Alt+V E on Mac OS))** or click the button  **Validate with** available in the **Validate** toolbar. This action can be used to validate the current document using a selectable schema (XML Schema, DTD, Relax NG, NVDL, Schematron schema). It returns an error list in the message panel. Mark-up of current document is checked to conform with the specified schema rules.
- Select submenu **Batch Validation > Validate** in the contextual menu of **Navigator** or **Package Explorer** view, to validate all selected files with their declared schemas.
- Select submenu **Batch Validation > Validate With ...** of the contextual menu of **Navigator** or **Package Explorer** view, to select a schema and validate all selected files with that schema.
- Select menu **XML > Clear Validation Markers (Alt+Shift+V X) ((Cmd+Alt+V X on Mac OS))** or click the toolbar button  **Clear Validation Markers** to clear the error markers added to the **Problems** view at the last validation of the current edited document.

- Select the submenu **Batch Validation > Configure Validation Scenario ...** of the contextual menu of **Navigator** or **Package Explorer** view, to configure and apply a validation scenario in one action to all the selected files in the **Navigator** or **Package Explorer** view.

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

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

Resolving References to Remote Schemas with an XML Catalog

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

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

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

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

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

```
<topic xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1">
```

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

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

Document Navigation

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

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.

```






+ <person id="Big.Boss">
- <person id="one.worker">
  <name>
    <family>Worker</family>
    <given>One</given>
  </name>
  <email>one@oxygenxml.com</email>
  <link manager="Big.Boss"/>
</person>
+ <person id="two.worker">
- <person id="three.worker">
  <name>
    <family>Worker</family>
    <given>Three</given>
  </name>
  <email>three@oxygenxml.com</email>
  <link manager="Big.Boss"/>
</person>
- <person id="four.worker">

```

Figure 35: 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+)** > **Document > Folding >**  **Close Other Folds (Ctrl+NumPad+)** - Folds all the elements except the current element.
- **Document > Folding >**  **Collapse Child Folds (Ctrl+Decimal) (Ctrl+NumPad+-) ((Cmd+NumPad+- on Mac OS))** - Folds the elements indented with one level inside the current element.
- **Document > Folding >**  **Expand Child Folds (Ctrl+NumPad++) ((Cmd+NumPad++))** - Unfolds all child elements of the currently selected element.
- **Document > Folding >**  **Expand All (Ctrl+NumPad+*) ((Cmd+NumPad+* on Mac OS))** - Unfolds all elements in the current document.
- **Document > Folding >**  **Toggle Fold (Alt+Shift+Y) ((Cmd+Alt+Y on Mac OS))** - Toggles the state of the current fold.

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

Outline View

The Outline view offers the following functionality:

- [XML Document Overview](#) on page 67
- [Outline Specific Actions](#) on page 67
- [Modification Follow-up](#) on page 68
- [Document Structure Change](#) on page 68
- [Document Tag Selection](#) on page 68

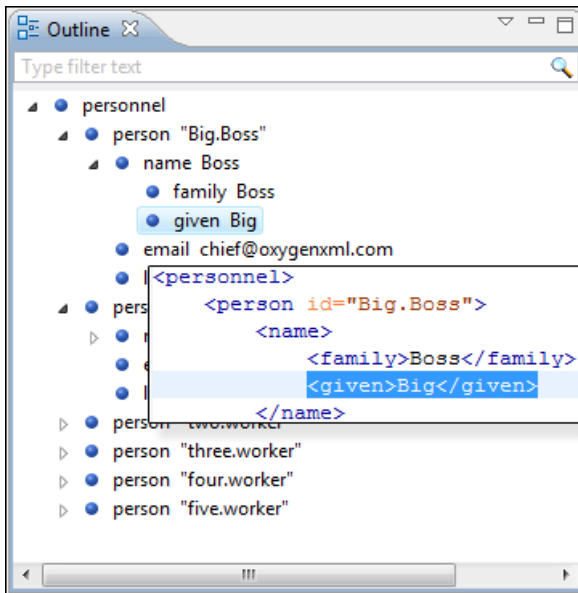


Figure 36: The Outline View

XML Document Overview






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

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

Document errors (such as an element inserted in an invalid position, or a wrong attribute name, or a missing required attribute value) are highlighted in the **Outline** tree. An easy-to-spot exclamation mark sign is used as element icon, a red underline decorates the element name and value and a tooltip provides more information about the nature of the error.

Outline Specific Actions

The following actions are available in the :

-  **Flat presentation mode of the filtered results** - when active, the application flattens the filtered result elements to a single level.
-  **Show comments and processing instructions** - show/hide comments and processing instructions in the **Outline** view.
-  **Show text** - show/hide additional text content for the displayed elements.
-  **Show attributes** - show/hide attribute values for the displayed elements. The displayed attribute values can be changed from [the Outline preferences panel](#).
-  **Configure displayed attributes** - displays the [XML Structured Outline preferences page](#).

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

Modification Follow-up

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

Document Structure Change

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

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

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

The Popup Menu of the Outline Tree

The *Append Child*, *Insert Before* and *Insert After* submenus of the outline tree popup menu allow to quickly insert new tags in the document at the place of the element currently selected in the Outline tree. The *Append Child* submenu lists the names of all the elements which are allowed by the schema associated with the current document as child of the current element. The *Insert Before* and *Insert After* submenus of the Outline tree popup menu list the elements which are allowed by the schema associated with the current document as siblings of the current element inserted immediately before respectively after the current element.

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

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

The *Cut*, *Copy* and *Delete* items of the popup menu execute

Document Tag Selection

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

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

Large Documents

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

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

Including Document Parts with DTD Entities

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

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

A master document looks like this:

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

The referred document looks like this:

```
<section> ... here comes the section content ... </section>
```



Note:

The indicated DTD and the element names (*section*, *chapter*) are used here only for illustrating the inclusion mechanism. You can use any DTD and element names you need.

At a certain point in the master document there can be inserted the section *testing.xml* entity:

```
... &testing; ...
```

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

Including Document Parts with XInclude

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

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

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

Include a chapter in an article using XInclude

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

Chapter file (`introduction.xml`) looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd">
<chapter>
  <title>Getting started</title>
  <section>
    <title>Section title</title>
    <para>Para text</para>
  </section>
</chapter>
```

Main article file looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.docbook.org/xml/4.3/docbookx.dtd"
[ <!ENTITY % xinclude SYSTEM
"../frameworks/docbook/dtd/xinclude.mod">
%xinclude;
]>
<article>
  <title>Install guide</title>
  <para>This is the install guide.</para>
  <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
             href="introduction.dita">
    <xi:fallback>
      <para>
        <emphasis>FIXME: MISSING XINCLUDE CONTENT</emphasis>
      </para>
    </xi:fallback>
  </xi:include>
</article>
```

In this example the following is of note:

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

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

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

and the a.xml file is:

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

after resolving the XPointer reference the document is:

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

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

Working with XML Catalogs

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

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

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

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

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

```
<topic xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1">
```

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

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

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

An XML Catalog file can be created quickly in Oxygen 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](#).

[User preferences related to XML Catalogs](#) can be configured from **Window > Preferences ... > oXygen > XML > XML Catalog**

XML Catalog


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

Important:

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

Converting Between Schema Languages

The **Generate/Convert Schema** allows you to convert a DTD or Relax NG (full or compact syntax) schema or a set of XML files to an equivalent XML Schema, DTD or Relax NG (full or compact syntax) schema. Where perfect equivalence is not possible due to limitations of the target language Oxygen XML Developer plugin will generate an approximation of the source schema.

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

A schema being edited can be converted with just one click on a toolbar button if that schema can be the subject of a supported conversion.

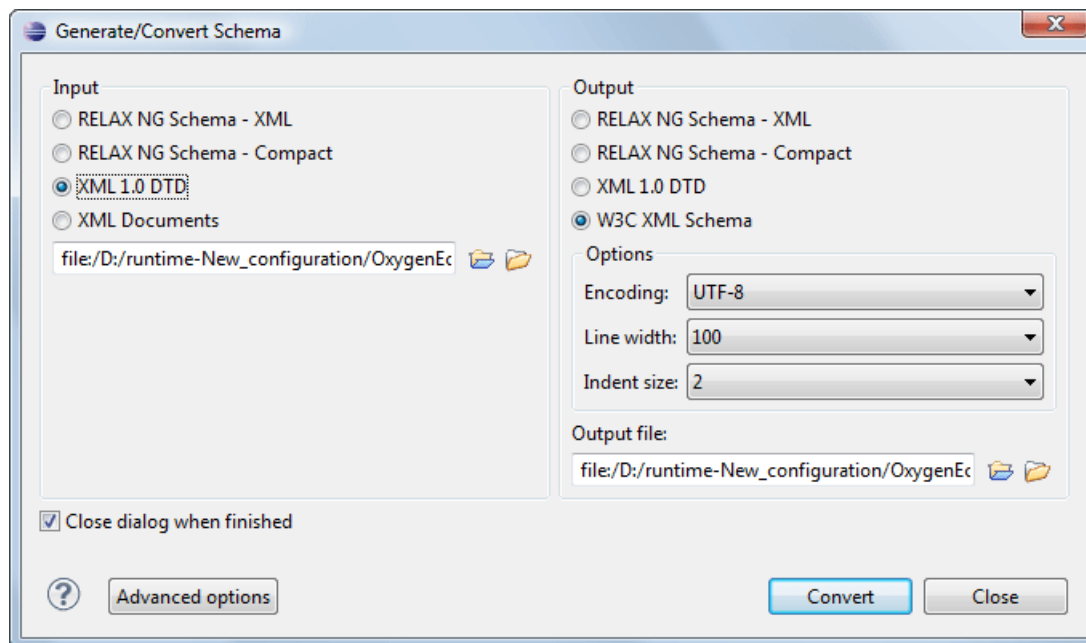


Figure 37: Convert a Schema to Other Schema Language

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

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

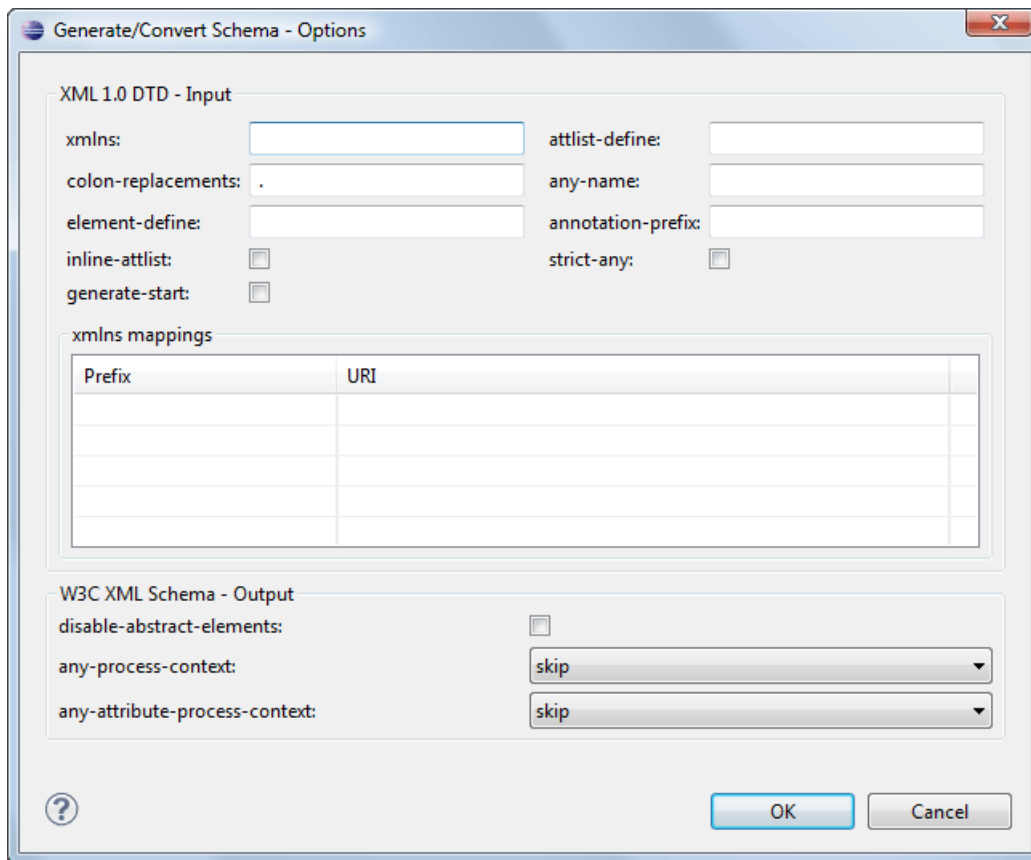


Figure 38: Convert a Schema to Other Schema Language - Advanced Options

For the **Input** panel:

- **xmlns** field - Specifies the default namespace, that is the namespace used for unqualified element names;
- **xmlns** table - Each row specifies in the prefix used for a namespace in the input schema;
- **colon-replacement** - Replaces colons in element names with the specified chars when constructing the names of definitions used to represent the element declarations and attribute list declarations in the DTD;
- **element-define** - Specifies how to construct the name of the definition representing an element declaration from the name of the element. The specified value must contain exactly one percent character. This percent character is replaced by the name of element (after colon replacement) and the result is used as the name of the definition;
- **inline-attlist** - Instructs the application not to generate definitions for attribute list declarations, but instead move attributes declared in attribute list declarations into the definitions generated for element declarations. This is the default behavior when the output language is XSD;
- **attlist-define** - Specifies how to construct the name of the definition representing an attribute list declaration from the name of the element. The specified value must contain exactly one percent character. This percent character is replaced by the name of element (after colon replacement) and the result is used as the name of the definition;
- **any-name** - Specifies the name of the definition generated for the content of elements declared in the DTD as having a content model of ANY;
- **strict-any** - Preserves the exact semantics of ANY content models by using an explicit choice of references to all declared elements. By default, Trang uses a wildcard that allows any element;
- **generate-start** - Specifies whether Trang should generate a start element. DTD's do not indicate what elements are allowed as document elements. Trang assumes that all elements that are defined but never referenced are allowed as document elements;
- **annotation-prefix** - Default values are represented using an annotation attribute *prefix:defaultValue* where prefix is the specified value and is bound to <http://relaxng.org/ns/compatibility/annotations/1.0> as defined by the RELAX NG DTD Compatibility Committee Specification. By default, Trang will use a for prefix unless that conflicts with a prefix used in the DTD.

For the **Output** panel:

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

Formatting and Indenting Documents (Pretty Print)

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

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

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

To change the indenting of the current selected text see the *Indent selection* action.

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

XML elements can be excepted from the reformatting performed by the pretty-print operation by including them in the *Preserve space elements (XPath)* list. That means that when the *Format and Indent* (pretty-print) action encounters in the document an element with the name contained in this list, the whitespace is preserved inside that element. This is useful when most of the elements must be reformatted with the exception of a few ones which are listed here.

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

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

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

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

Viewing Status Information

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

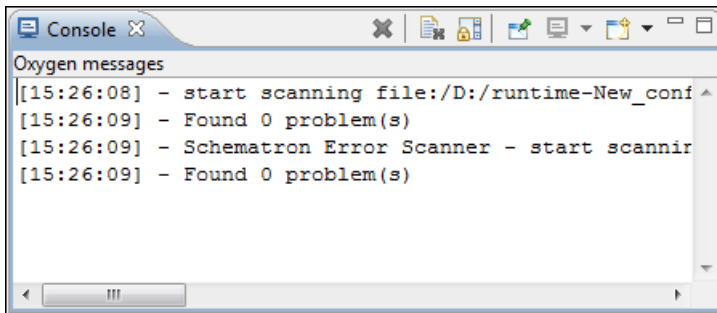


Figure 39: The Console view messages

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*.

XML Editor Specific Actions

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

Edit Actions

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

- **contextual menu of current editor** > **Toggle comment (Ctrl + /)** - 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 on 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 after the opening quote or before the closing quote of an attribute value - Select the whole attribute value.

Source Actions

The following actions can be applied on the text content of the XML editor:

- **contextual menu of current editor** > **Source** > **Escape Selection ...**  - Escapes a range of characters by replacing them with the corresponding character entities.
- **contextual menu of current editor** > **Source** > **Unescape Selection ...**  - Replaces the character entities with the corresponding characters.
- **contextual menu of current editor** > **Source** >  **Indent selection (Ctrl + I) ((Cmd + I on Mac OS))** - Corrects the indentation of the selected block of lines if it does not follow the current *indenting preferences of the user*.



- **contextual menu of current editor** > **Source** >  **Format and Indent Element (Ctrl + Shift + I)** - Pretty prints the element that surrounds the caret position.
- **contextual menu of current editor** > **Source** > **Insert XInclude**  - Shows a dialog that allows you to browse and select the content to be included and generates automatically the corresponding XInclude instruction.
- **contextual menu of current editor** > **Source** > **Import entities list**  - Shows a dialog that allows you to select a list of files as sources for external DTD entities. The internal subset of the DOCTYPE declaration of your document will be updated with the chosen entities. For instance, if choosing the file `chapter1.xml` and `chapter2.xml`, the following section is inserted in the DOCTYPE:

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

- **contextual menu of current editor** > **Join and normalize** - The action works on the selection. It joins the lines by replacing the *line separator* with a single space character. It also normalizes the whitespaces by replacing a sequence of such characters with a single space.

XML Document Actions

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



- **contextual menu of current editor** > **Show Definition** - Moves the cursor to the definition of the current element in the schema associated with the edited XML document (DTD, XML Schema, Relax NG schema).
- **contextual menu of current editor** > **Copy XPath (Ctrl+Shift+.)** - Copies the XPath expression of the current element or attribute from the current editor to the clipboard.
- **contextual menu of current editor** >  **Go to Matching Tag** - Moves the cursor to the end tag that matches the start tag, or vice versa.
- **contextual menu of current editor** > **Go after Next Tag (Ctrl+])** - Moves the cursor to the end of the next tag.
- **contextual menu of current editor** > **Go after Previous Tag (Ctrl+[)** - Moves the cursor to the end of the previous tag.
- **XML** > **Associate XSLT/CSS Stylesheet**  - Inserts an `xml-stylesheet` processing instruction at the beginning of the document referencing either an XSLT or a CSS file depending on the user selection. Either reference is useful for rendering the document in a Web browser when the action **Open in browser** is executed. Referencing the XSLT file is also useful for automatic detection of the XSLT stylesheet when there is no scenario associated with the current document.


When associating the CSS stylesheet, the user can also specify a title for it if it is an alternate one. Setting a *Title* for the CSS makes it the author's preferred stylesheet. Selecting the **Alternate** checkbox makes the CSS an alternate stylesheet.


Oxygen XML Developer plugin fully implements the W3C recommendation regarding [Associating Style Sheets with XML documents](#). See also [Specifying external style sheets](#) in HTML documents.

XML Refactoring Actions

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

- **context menu of current editor** > **XML Refactoring** >  **Surround with tag... (Alt+Shift+E) ((Cmd+Alt+E on Mac OS))** - Allows you to choose a tag that encloses a selected portion of content. If there is no selection, the start and end tags are inserted at the caret position. The caret is placed:
 - between the start and end tag, if the **Cursor position between tags** option is set;
 - at the end of the start tag, in an insert-attribute position, if the **Cursor position between tags** option is not set.
- **context menu of current editor** > **XML Refactoring** >  **Surround with <tag> (Alt+Shift+/) ((Cmd+Alt+/ on Mac OS))** - Similar in behavior with the **Surround with tag...** action, except that it inserts the last tag used by the **Surround with tag...** action.

- **context menu of current editor > XML Refactoring >  Rename element (Alt+Shift+R) ((Cmd+Alt+R on Mac OS))** - the element from the caret position and the elements that have the same name as the current element can be renamed according with the options from the **Rename** dialog.




context menu of current editor > XML Refactoring >  Rename prefix (Alt+Shift+P) ((Cmd+Alt+P on Mac OS)) - the prefix of the element from the caret position and the elements that have the same prefix as the current element can be renamed according with the options from the **Rename** dialog.

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

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

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

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

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

Smart Editing

The following helper actions are available in the XML editor:

- *Closing tag auto-expansion* - If you want to insert content into an auto closing tag like `<tag/>` deleting the `/` character saves some keystrokes by inserting a separate closing tag automatically and placing the cursor between the start and end tags: `<tag></tag>`
- *Auto-rename matching tag* - When you edit the name of the start tag, Oxygen XML 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 pretty-print operation](#).
- *Indent on Enter* - The [Indent on Enter option](#) indents the new line inserted when Enter is pressed.
- *Smart Enter* - The [Smart Enter option](#) inserts an empty line between the start and end tags. If Enter is pressed between a start and an end tag the action places the cursor in an indented position on the empty line between the lines that contain the start and end tag.
- *Triple click* - A triple click with the left mouse button selects a different region of text of the current document depending on the position of the click in the document:
 - if the click position is inside a start tag or an end tag then the entire element enclosed by that tag is selected
 - if the click position is immediately after a start tag or immediately before an end tag then the entire content of the element enclosed by that tag is selected, including all the child elements but excluding the start tag and the end tag of the element
 - otherwise the triple click selects the entire current line of text

Syntax Highlight Depending on Namespace Prefix

The *syntax highlight scheme of an XML file type* allows the configuration of a color per each type of token which can appear in an XML file. Distinguishing between the XML tag tokens based on the namespace prefix brings additional visual help in editing some XML file types. For example in XSLT stylesheets elements from different namespaces like XSLT, XHTML, XSL:FO or XForms are inserted in the same document and the editor panel can become cluttered.

Marking tags with different colors based on the namespace prefix allows easier identification of the tags.

```

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

```

Figure 40: Example of Coloring XML Tags by Prefix

Editing XHTML Documents

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

Editing XML Schemas

An XML Schema describes the structure of an XML document and is used to validate XML document instances against it, in order to check that the XML instances conform to the specified requirements. If an XML instance conforms to the schema then it is said to be valid, otherwise it is invalid.

Oxygen XML Developer plugin provides two editing modes for working with XML Schema: the usual *Text* editing mode and the visual *Design* editing mode.

XML Schema Text Editor

This page is used to edit the XML Schema in a text mode. It offers powerful content completion support, a synchronized Outline view and multiple *refactory actions*. The outline view has two display modes: the 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 you have to enable the *Show Full Model XML Schema diagram* checkbox from the *Diagram* preferences page.

Special Content Completion Features

The editor enhances *the content completion of the XML editor* inside the `xs:annotation/xs:appinfo` elements of an XML Schema with special support for the elements and attributes from a custom schema (by default ISO Schematron). This content completion enhancement can be configured from the *XSD Content Completion* preferences page.

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

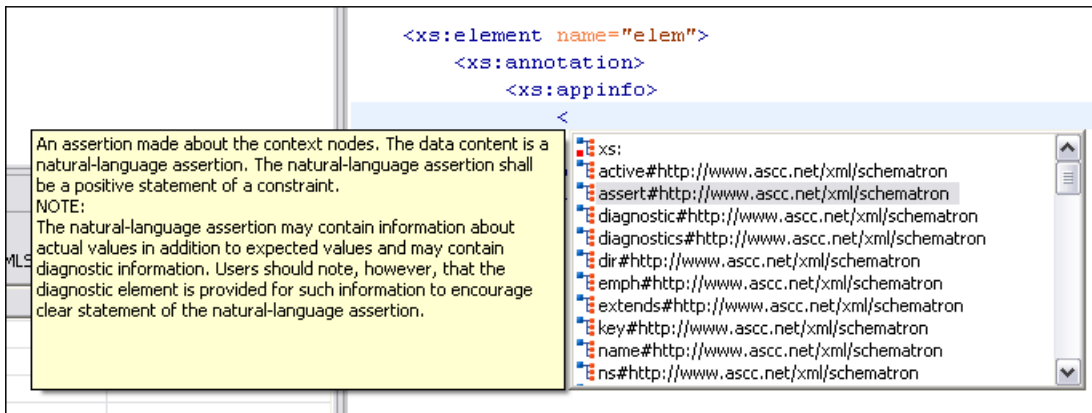


Figure 41: Schematron Support in XML Schema Content Completion

XML Schema Actions

- The **Show Definition** action accessed from the **contextual menu of current editor** > **Schema** > **Show Definition** (**Ctrl+Alt+ENTER**) moves the cursor to the definition of the referenced XML Schema item. The referenced item can be an element, group, simple type or complex type. The same action is executed on a double click on a component name in the *Schema Outline view*. You can define a scope for this action in the same manner you define it for *Search Declarations*.

Flatten an XML Schema

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

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

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

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

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

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

XML Schema Diagram Editor

This section explains how to use the graphical diagram of a W3C XML Schema.

Introduction

XML Schemas enable document designers to specify the allowed structure and content of an XML document and to check if an XML document is valid.

Oxygen 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.

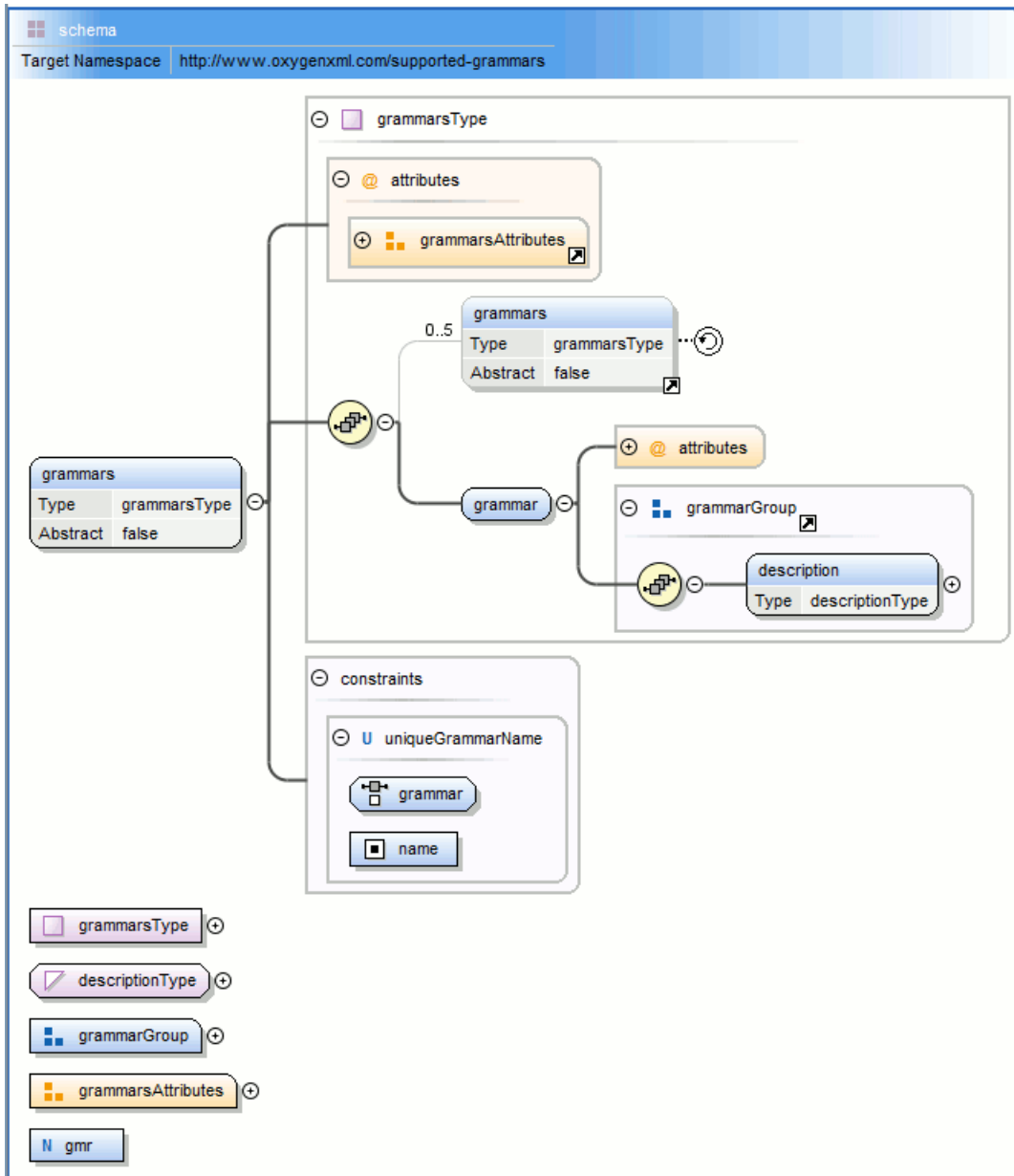
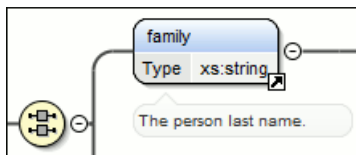


Figure 42: XML Schema Diagram

XML Schema Components

A schema diagram contains a series of interconnected components. To quickly identify the relation between two connected components, the connection is represented as:

- a thick line to identify a connection with a required component (in the following image, family is a required element);

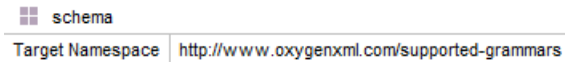


- a thin line to identify a connection with an optional component (in the following image, email is an optional element).



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

xs:schema



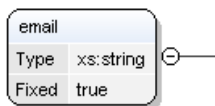
Defines the root element of a schema. A schema document contains representations for a collection of schema components, e.g. type definitions and element declarations, which have a common target namespace. See more info at <http://www.w3.org/TR/xmlschema-1/#element-schema>.

By default it displays the *targetNamespace* property when rendered.

xs:schema properties

Property Name	Description	Possible Values
Target Namespace	The schema target namespace.	Any URI
Element Form Default	Determining whether local element declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Attribute Form Default	Determining whether local attribute declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Block Default	Default value of the <code>block</code> attribute of <code>xs:element</code> and <code>xs:complexType</code> .	#all, extension, restriction, substitution, restriction extension, restriction substitution, extension substitution, restriction extension substitution, [Empty]
Final Default	Default value of the <code>final</code> attribute of <code>xs:element</code> and <code>xs:complexType</code> .	#all, restriction, extension, restriction extension, [Empty]
Version	Schema version	Any token
ID	The schema id	Any ID
Component	The edited component name.	Not editable property.
SystemID	The schema system id	Not editable property.

xs:element



Defines an element. An element declaration is an association of a name with a type definition, either simple or complex, an (optional) default value and a (possibly empty) set of identity-constraint definitions. See more info at <http://www.w3.org/TR/xmlschema-1/#element-element>.

An element by default displays the following properties when rendered in the diagram: *default*, *fixed*, *abstract* and *type*. When referenced or declared locally, the element graphical representation also contains the value for the *minOccurs*

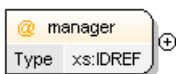
and *maxOccurs* properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the element are drawn using dotted lines if the element is optional.

xs:element properties

Property Name	Description	Possible Values	Mentions
Name	The element name. Always required.	Any NCName for global or local elements, any QName for element references.	If missing, will be displayed as '[element]' in diagram.
Is Reference	When set, the local element is a reference to a global element.	true/false	Appears only for local elements.
Type	The element type.	All declared or built-in types. In addition, the following anonymous types are available: [ST-restriction], [ST-union], [ST-list], [CT-anonymous], [CT-extension SC], [CT-restriction SC], [CT-restriction CC], [CT-extension CC].	For all elements. For references, the value is set in the referred element.
Base Type	The extended/restricted base type.	All declared or built-in types	For elements with complex type, with simple or complex content.
Mixed	Defines if the complex type content model will be mixed.	true/false	For elements with complex type.
Content	The content of the complex type.	simple/complex	For elements with complex type which extends/restricts a base type. It is automatically detected.
Content Mixed	Defines if the complex content model will be mixed.	true/false	For elements with complex type which has a complex content.
Default	Default value of the element. A default value is automatically assigned to the element when no other value is specified.	Any string	The fixed and default attributes are mutually exclusive.
Fixed	A simple content element may be fixed to a specific value using this attribute. A fixed value is also automatically assigned to the element and you cannot specify another value.	Any string	The fixed and default attributes are mutually exclusive.
Min Occurs	Minimum number of occurrences of the element.	A numeric positive value. Default value is 1	Only for references/local elements
Max Occurs	Maximum number of occurrences of the element.	A numeric positive value. Default value is 1	Only for references/local elements

Property Name	Description	Possible Values	Mentions
Substitution Group	Qualified name of the head of the substitution group to which this element belongs.	All declared elements	For global and reference elements
Abstract	Controls whether the element may be used directly in instance XML documents. When set to true, the element may still be used to define content models, but it must be substituted through a substitution group in the instance document.	true/false	For global elements and element references
Form	Defines if the element is "qualified" (i.e., belongs to the target namespace) or "unqualified" (i.e., doesn't belong to any namespace).	unqualified/qualified	Only for local elements
Nilable	When this attribute is set to true, the element can be declared as nil using an <code>xsi:nil</code> attribute in the instance documents.	true/false	For global elements and element references
Block	Controls whether the element can be subject to a type or substitution group substitution. '#all' blocks any substitution, 'substitution' blocks any substitution through substitution groups and 'extension'/'restriction' block any substitution (both through <code>xsi:type</code> and substitution groups) by elements or types, derived respectively by extension or restriction from the type of the element. Its default value is defined by the <code>blockDefault</code> attribute of the parent <code>xs:schema</code> .	#all, restriction, extension,substitution, extension restriction, extension substitution, restriction substitution, restriction extension substitution	For global elements and element references
Final	Controls whether the element can be used as the head of a substitution group for elements whose types are derived by extension or restriction from the type of the element. Its default value is defined by the <code>finalDefault</code> attribute of the parent <code>xs:schema</code> .	#all, restriction, extension, restriction extension, [Empty]	For global elements and element references
ID	The component id.	Any id	For all elements.

Property Name	Description	Possible Values	Mentions
Component	The edited component name.	Not editable property.	For all elements.
Namespace	The component namespace.	Not editable property.	For all elements.
System ID	The component system id.	Not editable property.	For all elements.

xs:attribute

The manager ID.

Defines an attribute. See more info at <http://www.w3.org/TR/xmlschema-1/#element-attribute>.

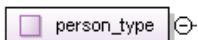
An attribute by default displays the following properties when rendered in the diagram: *default*, *fixed*, *use* and *type*. Connectors to the attribute are drawn using dotted lines if the attribute use is optional. The attribute name is stroked out if prohibited.

xs:attribute properties

Property Name	Description	Possible Value	Mentions
Name	Attribute name. Always required.	Any NCName for global/local attributes, all declared attributes' QName for references.	For all local or global attributes. If missing, will be displayed as '[attribute]' in the diagram.
Is Reference	When set, the local attribute is a reference.	true/false	For local attributes.
Type	Qualified name of a simple type.	All global simple types and built-in simple types. In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for creating anonymous simple types more easily.	For all attributes. For references, the type is set to the referred attribute.
Default	Default value. When specified, an attribute is added by the schema processor (if it is missing from the instance XML document) and it is given this value. The default and fixed attributes are mutually exclusive.	Any string	For all local or global attributes. For references the value is from the referred attribute.
Fixed	When specified, the value of the attribute is fixed and must be equal to this value. The default and fixed attributes are mutually exclusive.	Any string	For all local or global attributes. For references the value is from the referred attribute.
Use	Possible usage of the attribute. Marking an attribute "prohibited" is useful to exclude attributes	optional, required, prohibited	For local attributes


Property Name	Description	Possible Value	Mentions
Form	during derivations by restriction. Specifies if the attribute is qualified (i.e., must have a namespace prefix in the instance XML document) or not. The default value for this attribute is specified by the <code>attributeFormDefault</code> attribute of the <code>xs:schema</code> document element.	unqualified/qualified	For local attributes.
ID	The component id.	Any id	For all attributes.
Component	The edited component name.	Not editable property.	For all attributes.
Namespace	The component namespace.	Not editable property.	For all attributes.
System ID	The component system id.	Not editable property.	For all attributes.

xs:complexType



Defines a top level complex type. Complex Type Definitions provide for: See more data at <http://www.w3.org/TR/xmlschema-1/#element-complexType>.

- Constraining element information items by providing Attribute Declarations governing the appearance and content of attributes.
- Constraining element information item children to be empty, or to conform to a specified element-only or mixed content model, or else constraining the character information item children to conform to a specified simple type definition.
- Using the mechanisms of Type Definition Hierarchy to derive a complex type from another simple or complex type.
- Specifying post-schema-validation info set contributions for elements.
- Limiting the ability to derive additional types from a given complex type.
- Controlling the permission to substitute, in an instance, elements of a derived type for elements declared in a content model to be of a given complex type.

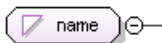
 **Tip:** A complex type which is a base type to another type will be rendered with yellow background.

xs:complexType properties

Property Name	Description	Possible Values	Mentions
Name	The name of the complex type. Always required.	Any NCName	Only for global complex types. If missing, will be displayed as '[complexType]' in diagram.
Base Type Definition	The name of the extended/restricted types.	Any from the declared simple or complex types.	For complex types with simple or complex content.
Derivation Method	The derivation method.	restriction/ extension	Only when base type is set. If the base type is a simple type, the derivation method is always extension.


Property Name	Description	Possible Values	Mentions
Content	The content of the complex type.	simple/ complex	For complex types which extend/restrict a base type. It is automatically detected.
Content Mixed	Specifies if the complex content model will be mixed.	true/false	For complex contents.
Mixed	Specifies if the complex type content model will be mixed.	true/false	For global and anonymous complex types.
Abstract	When set to <code>true</code> , this complex type cannot be used directly in the instance documents and needs to be substituted using an <code>xsi:type</code> attribute.	true/false	For global and anonymous complex types.
Block	Controls whether a substitution (either through a <code>xsi:type</code> or substitution groups) can be performed for a complex type, which is an extension or a restriction of the current complex type. This attribute can only block such substitutions (it cannot "unblock" them), which can also be blocked in the element definition. The default value is defined by the <code>blockDefault</code> attribute of <code>xs:schema</code> .	all, extension, restriction, extension restriction, [Empty]	For global complex types.
Final	Controls whether the complex type can be further derived by extension or restriction to create new complex types.	all, extension, restriction, extension restriction, [Empty]	For global complex types.
ID	The component id.	Any id	For all complex types.
Component	The edited component name.	Not editable property.	For all complex types.
Namespace	The component namespace.	Not editable property.	For all complex types.
System ID	The component system id.	Not editable property.	For all complex types.

`xs:simpleType`



The person name.

Defines a simple type. A simple type definition is a set of constraints on strings and information about the values they encode, applicable to the normalized value of an attribute information item or of an element information item with no element children. Informally, it applies to the values of attributes and the text-only content of elements. See more info at <http://www.w3.org/TR/xmlschema-1/#element-simpleType>.

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

`xs:simpleType` properties

Name	Description	Possible Values	Scope
Name	Simple type name. Always required.	Any NCName.	Only for global simple types. If missing, will be displayed as '[simpleType]' in diagram.
Derivation	The simple type category: restriction, list or union.	restriction,list or union	For all simple types.
Base Type	A simple type definition component. Required if derivation method is set to restriction.	All global simple types and built-in simple types. In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for easily create anonymous simple types.	For global and anonymous simple types with the derivation method set to restriction.
Item Type	A simple type definition component. Required if derivation method is set to list.	All global simple types and built-in simple types(from schema for schema). In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for easily create anonymous simple types.	For global and anonymous simple types with the derivation method set to list. Derivation by list is the process of transforming a simple datatype (named the item type) into a whitespace-separated list of values from this datatype. The item type can be defined inline by adding a simpleType definition as a child element of the list element, or by reference, using the itemType attribute (it is an error to use both).
Member Types	Category for grouping union members.	Not editable property.	For global and anonymous simple types with the derivation method set to union.
Member	A simple type definition component. Required if derivation method is set to union.	All global simple types and built-in simple types(from schema for schema). In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for easily create anonymous simple types.	For global and anonymous simple types with the derivation method set to union. Deriving a simple datatype by union merges the lexical spaces of several simple datatypes (called member types) to create a new simple datatype. The member types can be defined either by reference (through the memberTypes attribute) or embedded as simple datatype local definitions in the xs:union element. Both styles can be mixed.
Final	Blocks any further derivations of this datatype	#all, list, restriction, union, list restriction, list union,	Only for global simple types.

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

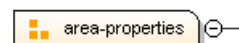
xs:group

Defines a group of elements to be used in complex type definitions. See more info at <http://www.w3.org/TR/xmlschema-1/#element-group>.

When referenced, the graphical representation also contains the value for the *minOccurs* and *maxOccurs* properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the group are drawn using dotted lines if the group is optional.

xs:group properties

Property Name	Description	Possible Values	Mentions
Name	The group name. Always required.	Any NCName for global groups, all declared groups for reference.	If missing, will be displayed as '[group]' in diagram.
Min Occurs	Minimum number of occurrences of the group.	A numeric positive value. Default value is 1.	Appears only for reference groups.
Max Occurs	Maximum number of occurrences of the group.	A numeric positive value. Default value is 1.	Appears only for reference groups.
ID	The component id.	Any id	For all groups.
Component	The edited component name.	Not editable property.	For all groups.
Namespace	The component namespace.	Not editable property	For all groups.
System ID	The component system id.	Not editable property.	For all groups.

xs:attributeGroup

The properties of an area.

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

xs:attributeGroup properties

Property Name	Description	Possible Values	Mentions
Name	Attribute group name. Always required.	Any NCName for global attribute groups, all declared attribute groups for reference.	For all global or referred attribute groups. If missing, will be displayed as '[attributeGroup]' in diagram.

Property Name	Description	Possible Values	Mentions
ID	The component id.	Any id	For all attribute groups.
Component	The edited component name.	Not editable property.	For all attribute groups.
Namespace	The component namespace.	Not editable property.	For all attribute groups.
System ID	The component system id.	Not editable property.	For all attribute groups.

xs:include

Adds multiple schemas with the same target namespace to a document. See more info at <http://www.w3.org/TR/xmlschema-1/#element-include>.

xs:include properties

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

xs:import

Adds multiple schemas with different target namespace to a document. See more info at <http://www.w3.org/TR/xmlschema-1/#element-import>.

xs:import properties

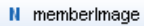
Property Name	Description	Possible Values
Schema Location	Imported schema location	Any URI
Namespace	Imported schema namespace	Any URI
ID	Import ID	Any ID
Component	The component name	Not editable property.

xs:redefine

Redefines simple and complex types, groups, and attribute groups from an external schema. See more info at <http://www.w3.org/TR/xmlschema-1/#element-redefine>.

xs:redefine properties

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

xs:notation


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

xs:notation properties

Property Name	Description	Possible values	Mentions
Name	The notation name. Always required.	Any NCName.	If missing, will be displayed as '[notation]' in diagram.
System Identifier	The notation system identifier.	Any URI	Required if public identifier is absent, otherwise optional.
Public Identifier	The notation public identifier.	A Public ID value	Required if system identifier is absent, otherwise optional.
ID	The component id.	Any ID	For all notations.
Component	The edited component name.	Not editable property.	For all notations.
Namespace	The component namespace.	Not editable property.	For all notations.
System ID	The component system id.	Not editable property.	For all notations.

xs:sequence, xs:choice, xs:all



Figure 43: An xs:sequence in diagram

xs:sequence specifies that the child elements must appear in a sequence. Each child element can occur from 0 to any number of times. See more info at <http://www.w3.org/TR/xmlschema-1/#element-sequence>.



Figure 44: An xs:choice in diagram

xs:choice allows only one of the elements contained in the declaration to be present within the containing element. See more info at <http://www.w3.org/TR/xmlschema-1/#element-choice>.



Figure 45: An xs:all in diagram

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

The compositor graphical representation also contains the value for the minOccurs and maxOccurs properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the compositor are drawn using dotted lines if the compositor is optional.

xs:sequence, xs:choice, xs:all properties

Property Name	Description	Possible Values	Mentions
Compositor	Compositor type.	sequence, choice, all.	'all' is only available as a child of a group or complex type.

Property Name	Description	Possible Values	Mentions
Min Occurs	Minimum occurrences of compositor.	A numeric positive value. Default is 1.	The property is not present if compositor is 'all' and is child of a group.
Max Occurs	Maximum occurrences of compositor.	A numeric positive value. Default is 1.	The property is not present if compositor is 'all' and is child of a group.
ID	The component id.	Any ID	For all compositors.
Component	The edited component name.	Not editable property.	For all compositors.
System ID	The component system id.	Not editable property.	For all compositors.

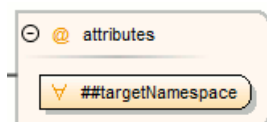
xs:any

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

The graphical representation also contains the value for the `minOccurs` and `maxOccurs` properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the wildcard are drawn using dotted lines if the wildcard is optional.

xs:any properties

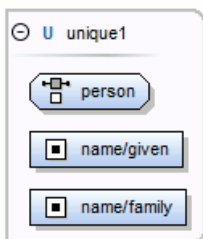
Property Name	Description	Possible Values
Namespace	The list of allowed namespaces. The namespace attribute expects a list of namespace URIs. In this list, two values have a specific meaning: '##targetNamespace' stands for the target namespace, and '##local' stands for local attributes (without namespaces).	##any, ##other, ##targetNamespace, ##local or anyURI
Process Contents	Type of validation required on the elements allowed for this wildcard.	skip, lax, strict
Min Occurs	Minimum occurrences of any	A numeric positive value. Default is 1.
Max Occurs	Maximum occurrences of any	A numeric positive value. Default is 1.
ID	The component id.	Any ID.
Component	The name of the edited component.	Not editable property.
System ID	The component system id.	Not editable property.

xs:anyAttribute

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

xs:anyAttribute properties

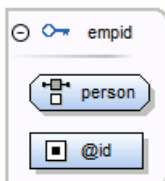
Property Name	Description	Possible Value
Namespace	The list of allowed namespaces. The namespace attribute expects a list of namespace URIs. In this list, two values have a specific meaning: '##targetNamespace' stands for the target namespace, and '##local' stands for local attributes (without namespaces).	##any, ##other, ##targetNamespace, ##local or anyURI
Process Contents	Type of validation required on the elements allowed for this wildcard.	skip, lax, strict
ID	The component id.	Any ID.
Component	The name of the edited component.	Not editable property.
System ID	The component system id.	Not editable property.

xs:unique

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

xs:unique properties

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

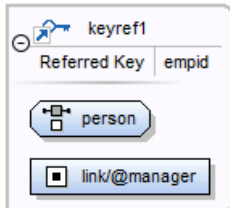
xs:key

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

xs:key properties

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

Property Name	Description	Possible Value
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
Namespace	The component namespace.	Not editable property.
System ID	The component system id.	Not editable property.

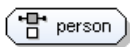
xs:keyRef

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

A keyref by default displays the *Referenced Key* property when rendered.

xs:keyRef properties

Property Name	Description	Possible Values
Name	The keyref name. Always required.	Any NCName.
Referred Key	The name of referred key.	any declared element constraints.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
Namespace	The component namespace.	Not editable property.
System ID	The component system id.	Not editable property.

xs:selector

Specifies an XPath expression that selects a set of elements for an identity constraint. See more info at <http://www.w3.org/TR/xmlschema-1/#element-selector>.

xs:selector properties

Property Name	Description	Possible Values
XPath	Relative XPath expression identifying the element on which the constraint applies.	An XPath expression.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not editable property.

xs:field

Specifies an XPath expression that specifies the value used to define an identity constraint. See more info at <http://www.w3.org/TR/xmlschema-1/#element-field>.

xs:field properties

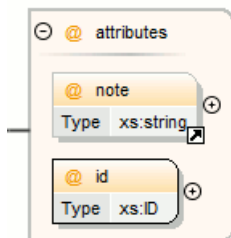
Property Name	Description	Possible Values
XPath	Relative XPath expression identifying the field(s) composing the key, key reference, or unique constraint.	An XPath expression.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not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

Constructs Used to Group Schema Components

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

- [Attributes](#)
- [Constraints](#)
- [Substitutions](#)

Attributes

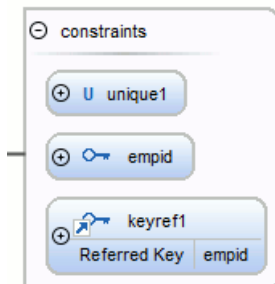


Groups all attributes and attribute groups belonging to a complex type.

Attributes properties

Property Name	Description	Possible Values
Component	The element for which the attributes are displayed.	Not editable property.
System ID	The component system id.	Not editable property.

Constraints

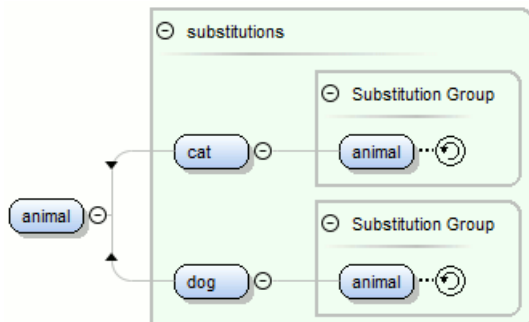


Groups all constraints (*xs:key*, *xs:keyRef* or *xs:unique*) belonging to an element.

Attributes properties

Property Name	Description	Possible Values
Component	The element for which the constraints are displayed.	Not editable property.
System ID	The component system id.	Not editable property.

Substitutions






Groups all elements which can substitute the current element.


Attributes properties


Property Name	Description	Possible Values
Component	The element for which the substitutions are displayed.	Not editable property.
System ID	The component system id.	Not editable property.

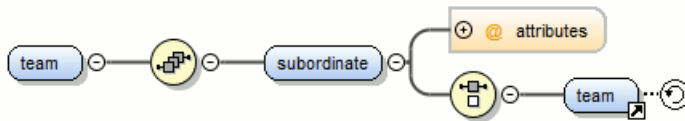
Navigation in the Schema Diagram

The following editing and navigation features work for all types of schema components:

- Move/refer components in the diagram using drag-and-drop actions.
- Select consecutive components on the diagram (components from the same level) using the *Shift* key to . You can also make discontinuous selections in the schema diagram using the *Ctrl* key.
- Use the arrow keys to navigate the diagram vertically and horizontally.
- Use *Home/End* keys to navigate to the first/last component from the same level. Use **(Ctrl - Home)** key combination to go to the diagram root and **(Ctrl - End)** to go to the last child of the selected component.
- You can easily go back to a previously visited component while moving from left to right. The path will be preserved only if you use the left arrow key or right arrow key. For example, if the current selection is on the second attribute from an attribute group and you press the left arrow key to navigate to the attribute group, when you press the right arrow key, then the selection will be moved to the second attribute.
- Go back and forward between components viewed or edited in the diagram by selecting them in the **Outline** view:
 -  **Back** (go to previous schema component)
 -  **Forward** (go to next schema component)
 -  **Go to Last Modification** (go to last modified schema component)
- Copy, refer or move global components, attributes, and identity constraints to a different position and from one schema to another using the **Cut/Copy** and **Paste/Paste as Reference** actions.
- Go to the definition of an element or attribute with the **Show Definition** action.
- 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.

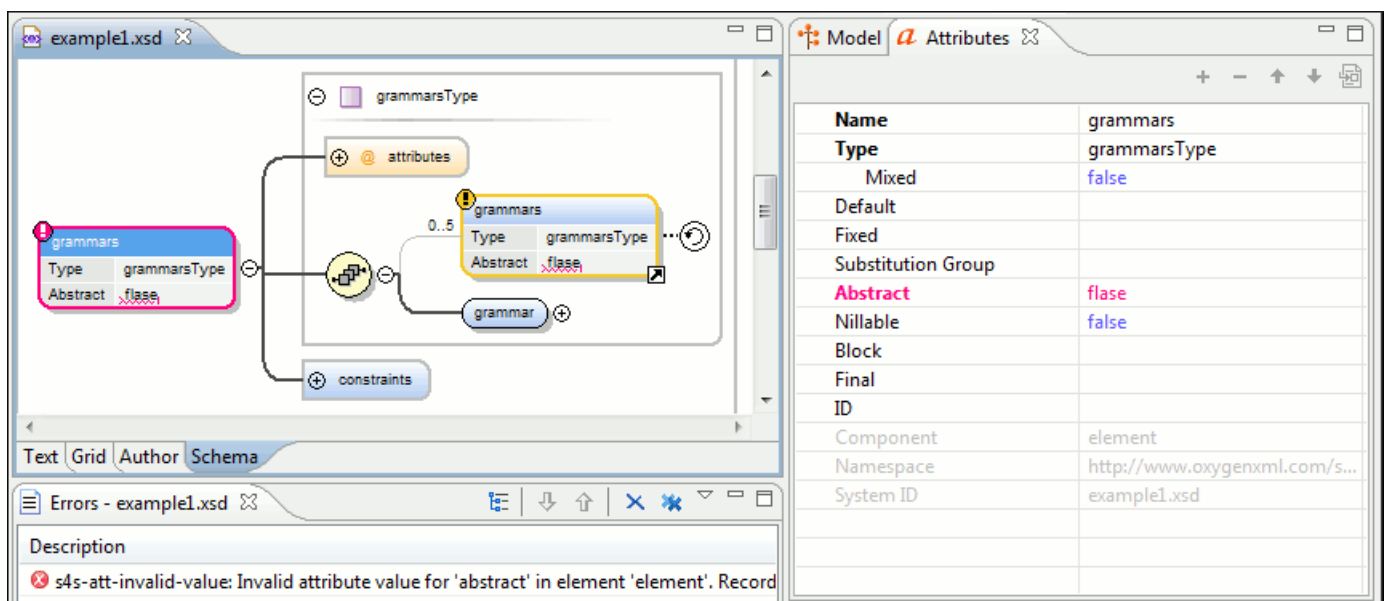


Figure 46: XML Schema Validation

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



- in the [Attributes View](#)
- in the diagram by surrounding the component that has the error with a red border

Invalid facets for a component are highlighted in the [Facets View](#).

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

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

If you validate the entire schema using **Document > Validate Document (Alt+Shift+V)** ((**Cmd+Alt+V on Mac OS**)) or the action available on the **Validate** toolbar, all validation errors will be presented in the **Errors** tab. To resolve an error, just click on it (or double click for errors located in other schemas) and the corresponding schema component will be displayed as the diagram root so that you can easily correct the error.

- 
Important: If the schema imports only the namespace of other schema without specifying the schema location and a *catalog is set-up* that maps the namespace to a certain location both the validation and the diagram will correctly identify the imported schema.
- 
Tip: If the validation action finds that the schema contains unresolved references, the application will suggest the use of validation scenarios, but only if the current edited schema is a XML Schema module.

Schema Editing Actions

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




Drag and drop action provides the easiest way to move the existing components to other locations in the schema. For example, an element reference can be quickly inserted in the diagram with a drag and drop from the **Outline** view to a compositor in the diagram. Also the components order in an `xs:sequence` can be easily changed using drag and drop.

If this property has not been set, you can easily set the attribute/element type by dragging over it a simple type or complex type from the diagram. If the type property for a simple type or complex type is not already set, you can set it by dragging over it a simple or complex type.

Depending on the drop area, different actions are available:

- **move** - Context dependent, the selected component is moved to the destination;
- **refer** - Context dependent, the selected component is referred from the parent;
- **copy** - If (**Ctrl**) key is pressed, a copy of the selected component is inserted to the destination.

Visual clues about the operation type are indicated by the mouse pointer shape:

-  - When moving a component;
-  - When referring a component;
-  - When copying a component.



You can edit some schema components directly in the diagram. For these components, you can edit the name and the additional properties presented in the diagram by double clicking the value you want to edit. If you want to edit the name of a selected component, you can also press (**Enter**). The list of properties which can be displayed for each component can be customized *in the Preferences*.

When editing references, you can choose from a list of available components. Components from an imported schema for which the target namespace does not have an associated prefix is displayed in the list as `componentName#targetNamespace`. If the reference is from a target namespace which was not yet mapped, you are prompted to add prefix mappings for the inserted component namespace in the current edited schema.


You can also change the compositor by double-clicking it and choose the compositor you want from the proposals list.






There are some components that cannot be edited directly in the diagram: imports, includes, redefines. The editing action can be performed if you double-click or press (**Enter**) on an import/include/define component. An edit dialog is displayed, allowing you to customize the directives.


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


-  **Show Definition** ((**Ctrl - Shift - Enter**)) - Shows the definition for the current selected component. For references, this action is available by clicking the arrow displayed in its bottom right corner.
-  **Open Schema** ((**Ctrl - Shift - Enter**)) - Opens the selected schema. This action is available for `xsd:import`, `xsd:include` and `xsd:redefine` elements. If the file you try to open does not exist, a warning message is displayed and you have the possibility to create the file.
- **Edit Attributes...** (**Alt - Shift - Enter**) - Allows you to edit the attributes of the selected component in a dialog that presents the same attributes as in the *Attributes View* and the *Facets View*. The actions that can be performed on attributes in this dialog are the same actions presented in the two views.

- **Append child** - Offers a list of valid components to append depending on the context. For example to a complex type you can append a compositor, a group, attributes or identity constraints (`unique`, `key`, `keyref`). You can set a name for a named component after it was added in the diagram.
- **Insert before** - Inserts before the selected component in the schema. The list of components that can be inserted depends on the context. For example, before an `xsd:import` you can insert an `xsd:import`, `xsd:include` or `xsd:redefine`. You can set a name for a named component after it was added in the diagram.
- **Insert after** - Inserts a component after the selected component on the schema. The list of components that can be inserted depends on the context. You can set a name for a named component after it was added in the diagram.
- **New global** - Inserts a global component in the schema diagram. This action does not depend on the current context. If you choose to insert an import you have to specify the URL of the imported file, the target namespace and the import ID. The same information, excluding the target namespace, is requested for an `xsd:include` or `xsd:redefine` element. See the **Edit Import** dialog for more details.

 **Note:** If the imported file has declared a target namespace, the field **Namespace** is completed automatically.

- **Edit Schema Namespaces...** - When performed on the schema root, it allows you to edit the schema target namespace and namespace mappings. You can also invoke the action by double-clicking the target namespace property from **Attributes** view for the schema or by double-clicking the schema component.
- **Edit Annotations...** - Allows you to edit the annotation for the selected schema component in the **Edit Annotations** dialog. You can perform the following operations in the dialog:
 - **Edit all appinfo/documentation items for a specific annotation** - All `appinfo/documentation` items for a specific annotation are presented in a table and can be easily edited. Information about an annotation item includes: type (`documentation/appinfo`), content, source (optional, specify the source of the `documentation/appinfo` element) and `xml:lang`. The content of a `documentation/appinfo` item can be edited in the **Content** area below the table.
 - **Insert/Insert before/Remove documentation/appinfo.**  - Allows you to insert a new annotation item (`documentation/appinfo`). You can add a new item before the item selected in table by pressing the  button. Also you can delete the selected item using the  button.
 - **Move items up/down** - To do this use the  and  buttons.
 - **Insert/Insert before/Remove annotation** - Available for components that allow multiple annotations like schemas or redefines.
 - **Specify an ID for the component annotation.** The ID is optional.

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

 **Note:** Annotations are rendered by default under the graphical representation of the component. When you have a reference to a component with annotations, these annotations are presented in the diagram also below the reference component. The **Edit Annotations** action invoked from the contextual menu edit the annotations for the reference. If the reference component does not have annotations, you can edit the annotations of the referred component by double-clicking the annotations area. Otherwise you can edit the referred component annotations only if you go to the definition of the component.

- **Extract Global Element** - Action available for local elements. A local element is made global and is replaced with a reference to the global element. The local element properties that are also valid for the global element declaration are kept.

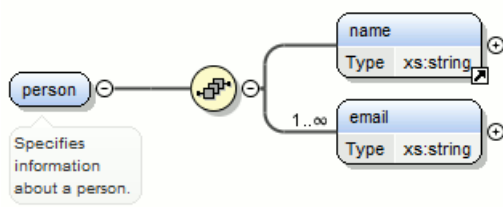
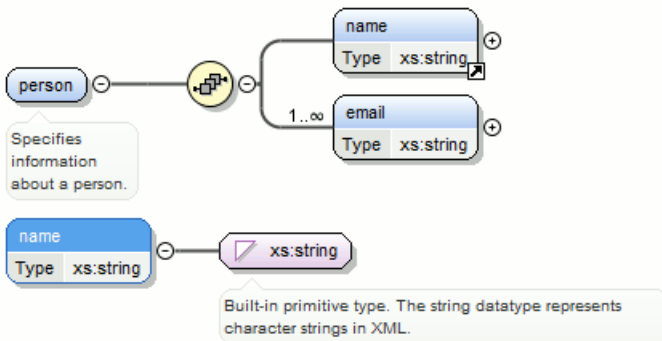


Figure 47: Extracting a Global Element

If you execute **Extract Global Element** on element name, the result is:



- **Extract Global Attribute** - Action available for local attributes. A local attribute is made global and replaced with a reference to the global attribute. The properties of local attribute that are also valid in the global attribute declaration are kept.

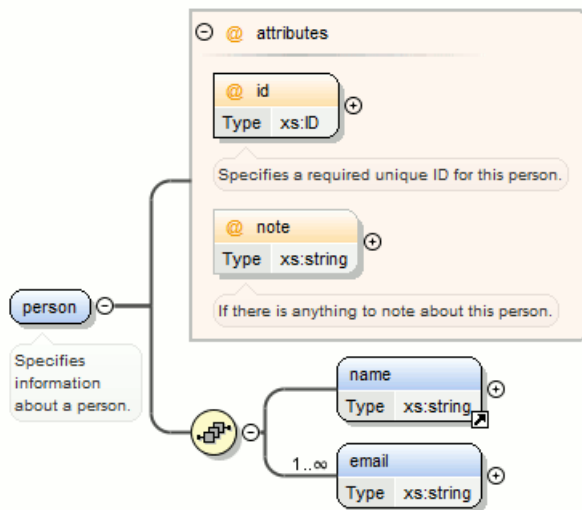
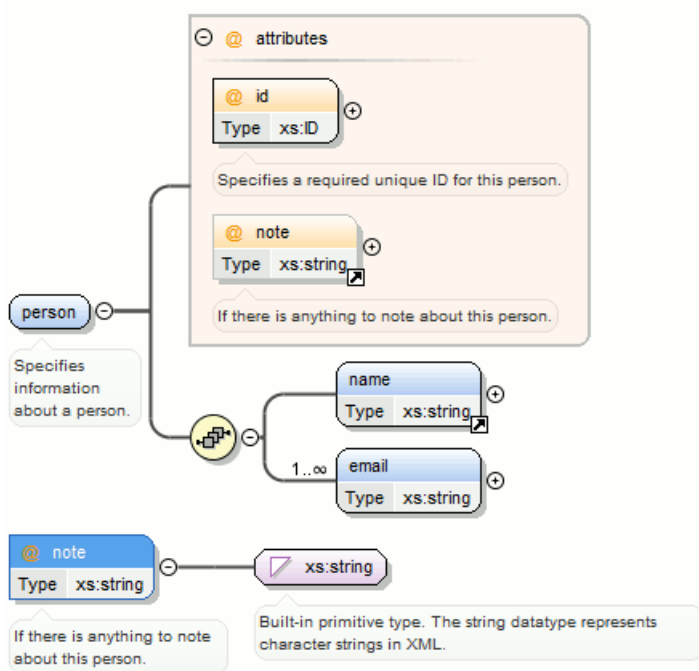
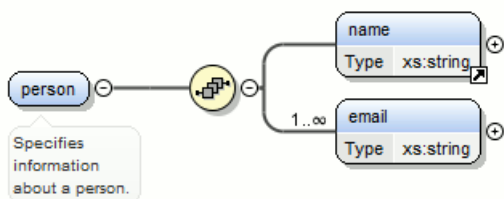


Figure 48: Extracting a Global Attribute

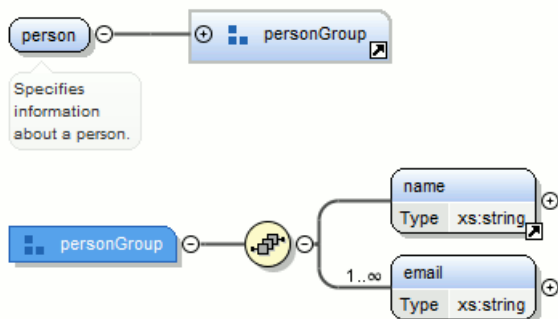
If you execute **Extract Global Attribute** on attribute note the result is:



- **Extract Global Group** - Action available for compositors (sequence, choice, all). This action extracts a global group and makes a reference to it. The action is enabled only if the parent of the compositor is not a group.



If you execute **Extract Global Group** on the sequence element, the **Extract Global Component** dialog is shown and you can choose a name for the group. If you type `personGroup`, the result



is:

Figure 49: Extracting a Global Group

- **Extract Global Type** - Action used to extract an anonymous simple type or an anonymous complex type as global. For anonymous complex types, the action is available on the parent element.

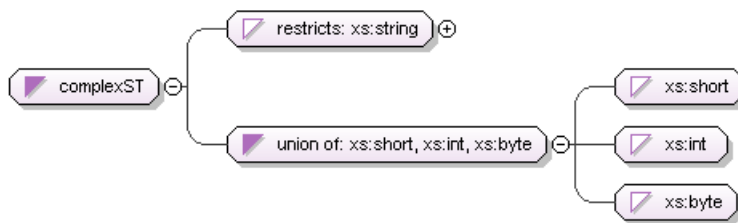


Figure 50: Extracting a Global Simple Type

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

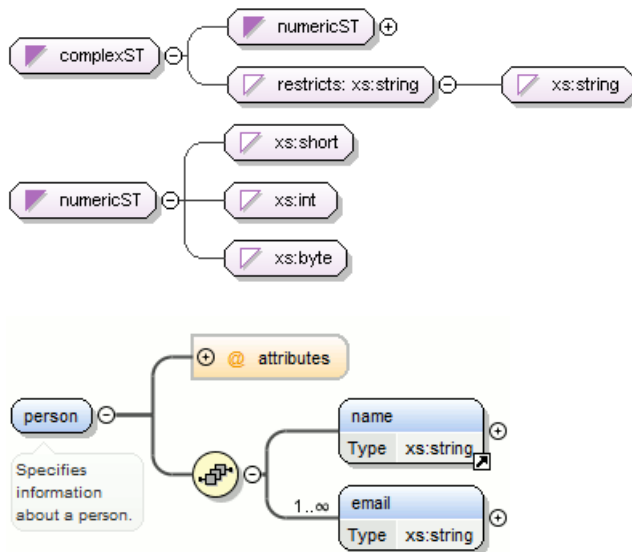
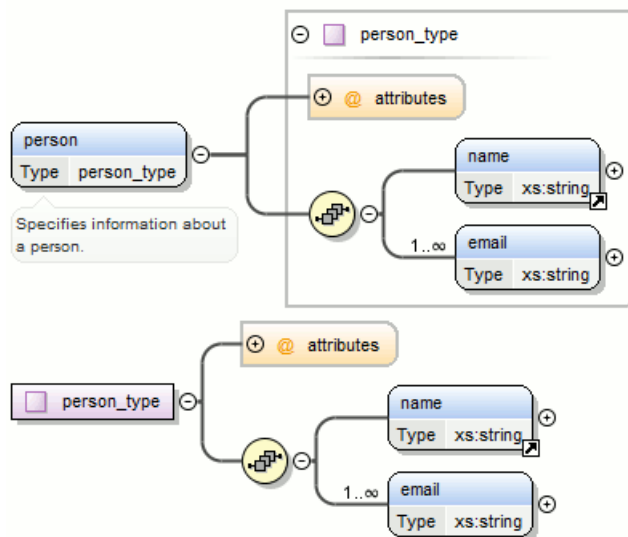











Figure 51: Extracting a Global Complex Type

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



- **Rename Component** - Rename the selected component.
-  **Cut (Ctrl - X)** - Cut the selected component(s).

-  **Copy (Ctrl - C)** - Copy the selected component(s).
-  **Paste (Ctrl - V)** - Paste the component(s) from the clipboard as children of the selected component.
- **Paste as Reference** - Create references to the copied component(s). If not possible a warning message is displayed.
- **Remove (Delete)** - Remove the selected component(s).
- **Optional** - Can be performed on element/attribute/group references, local attributes, elements, compositors, and element wildcards. The `minOccurs` property is set to 0 and the `use` property for attributes is set to `optional`.
- **Unbounded** - Can be performed on element/attribute/group references, local attributes, elements, compositors, and element wildcards. The `maxOccurs` property is set to `unbounded` and the `use` property for attributes is set to `required`.
- **Search** - Can be performed on local elements or attributes. This action makes a reference to a global element or attribute.
-  **Search References** - Searches all references of the item found at current cursor position in the defined scope if any.
- **Search References in...** - Searches all references of the item found at current cursor position in the specified scope.
- **Search Occurrences in File** - Searches all occurrences of the item found at current cursor position in the current file.
-  **Component Dependencies** - Allows you to see the dependencies for the current selected component.
- **Resource Hierarchy** - Allows you to see the hierarchy for the current selected resource.
- **Resource Dependencies** - Allows you to see the dependencies for the current selected resource.
-  **Expand all** - Expands recursively all sub-components of the selected component.
-  **Collapse all** - Collapses recursively all sub-components of the selected component.
- **Save as Image...** - Save the diagram as image, in JPEG, BMP, SVG or PNG format.
-  **Generate Sample XML Files** - Generate XML files using the current opened schema. The selected component is the XML document root. See more in the [Generate Sample XML Files](#) section.
-  **Options...** - Show the [Schema preferences panel](#).

Schema Outline View

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

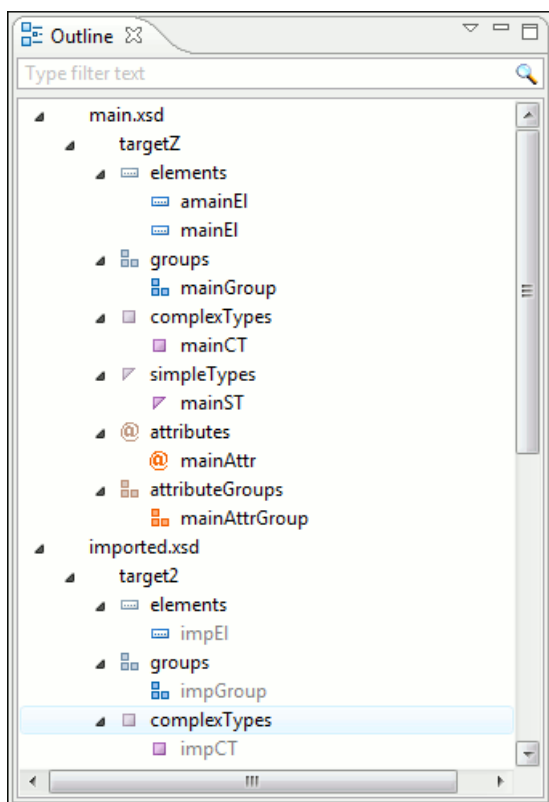


Figure 52: The Outline View for XML Schema


The **Outline** view provides the following options:

- **Selection update on caret move** - Allows a synchronization between **Outline** view and schema diagram. The selected view from the diagram is also selected in the **Outline** view.
- **Sort** - Allows you to sort alphabetically the schema components.
- **Show all components** - Displays all components that were collected starting from the main files. Components that are not referable from the current file are marked with an orange underline. To refer them, you need to add an import directive with the *componentNS* namespace.
- **Show referable components** - Displays all components (collected starting from the main files) that can be referred from the current file. This option is set by default.
- **Show local components** - Displays components defined in the current file only.
- **Group by location/namespace/type** - These three operations allow you to group the components by location, namespace, or type. When grouping by namespace, the main schema target namespace is the first presented in the **Outline** view.

The following contextual menu actions are available:

- **Remove (Delete)** - Removes the selected item from the diagram.
- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any.
- **Search References in...** - Searches all references of the item found at current cursor position in the specified scope.
- **Component Dependencies** - Allows you to see the dependencies for the current selected component.
- **Resource Hierarchy** - Allows you to see the hierarchy for the current selected resource.
- **Resource Dependencies** - Allows you to see the dependencies for the current selected resource.
- **Rename Component** - Renames the selected component.
- **Generate Sample XML Files...** - Generate XML files using the current opened schema. The selected component is the XML document root.

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

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

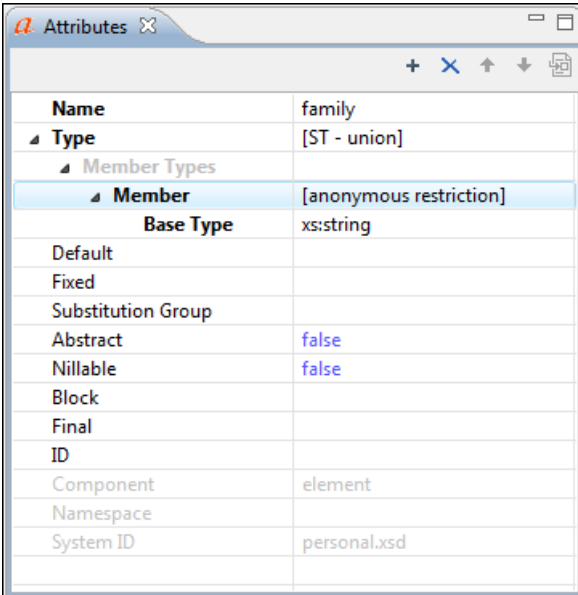
- * - any string
- ? - any character
- , - patterns separator

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

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

The Attributes View

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



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

Figure 53: The Attributes View

The default value of a property is presented in the **Attributes** view with blue foreground. The properties that can't be edited are rendered with gray foreground. A non-editable category which contains at least one child is rendered with bold. Bold properties are properties with values set explicitly to them.

Properties for components which do not belong to the current edited schema are read-only but if you double-click them you can choose to open the corresponding schema and edit them.

You can edit a property by double-clicking on by pressing Enter. For most properties you can choose valid values from a list or you can specify another value. If a property has an invalid value or a warning, it will be highlighted in the table with the corresponding foreground color. By default properties with errors are highlighted with red and the properties with warnings are highlighted with yellow. You can customize these colors from the [Document checking user preferences](#).

For imports, includes and redefines, the properties are not edited directly in the **Attributes** view. A dialog will be shown allowing you to specify properties for them.

The schema namespace mappings are not presented in **Attributes** view. You can view/edit these by choosing **Edit Schema Namespaces** from the contextual menu on the schema root. See more in the [Edit Schema Namespaces](#) section.

The **Attributes** view has five actions available on the toolbar and also on the contextual menu:

- **+ Add** - Allows you to add a new member type to an union's member types category.

- - **Remove** - Allows you to remove the value of a property.
- **Move Up** - Allows you to move up the current member to an union's member types category.
- **Move Down** - Allows you to move down the current member to an union's member types category.
- **Copy** - Copy the attribute value.
- **Show Definition** - Show the definition for the selected type.
- **Show Facets** - Allows you to edit the facets for a simple type.

The Facets View

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

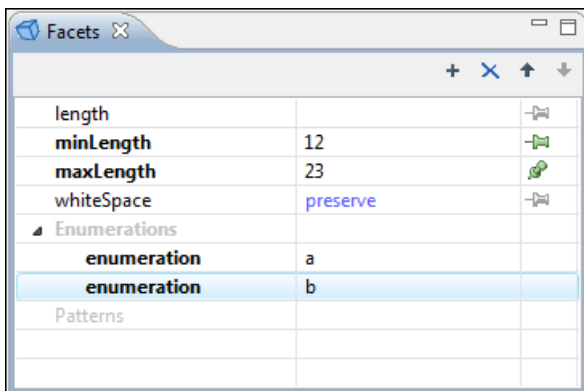


Figure 54: The Facets View

The default value of a facet is rendered in the **Facets** view with a blue color. The facets that can't be edited are rendered with a gray color. The grouping categories (eg: **Enumerations** and **Patterns**) are not editable. If these categories contain at least one child they are rendered with bold. Bold facets are facets with values set explicitly to them.

- **Important:** Usually inherited facets are presented as default in the **Facets** view but if patterns are inherited from a base type and also specified in the current simple type only the current specified patterns will be presented. You can see the effective pattern value obtained by combining the inherited and the specified patterns as a tooltip on the **Patterns** category.

Facets for components which do not belong to the current edited schema are read-only but if you double-click them you can choose to open the corresponding schema and edit them.

You can edit a facet by double-clicking on it or by pressing Enter, when that facet is selected. For some facets you can choose valid values from a list or you can specify another value. If a facet has an invalid value or a warning, it will be highlighted in the table with the corresponding foreground color. By default facets with errors are presented with red and the facets with warnings with yellow. You can customize the error colors from the [Document Checking user preferences](#).

The **Facets** view has four toolbar actions available also on the contextual menu:

- **Add** - Allows you to add a new enumeration or a new pattern.
- **Remove** - Allows you to remove the value of a facet.
- **Move Up** - Allows you to move up the current enumeration/pattern in **Enumerations/Patterns** category.
- **Move Down** - Allows you to move down the current enumeration/pattern in **Enumerations/Patterns** category.
- **Copy** - Copy the attribute value.
- **Open in Regular Expressions Builder** - Allows you to open the pattern in the [XML Schema Regular Expressions Builder](#)

Facets can be fixed to prevent a derivation from modifying its value. To fix a facet value just press the pin button.

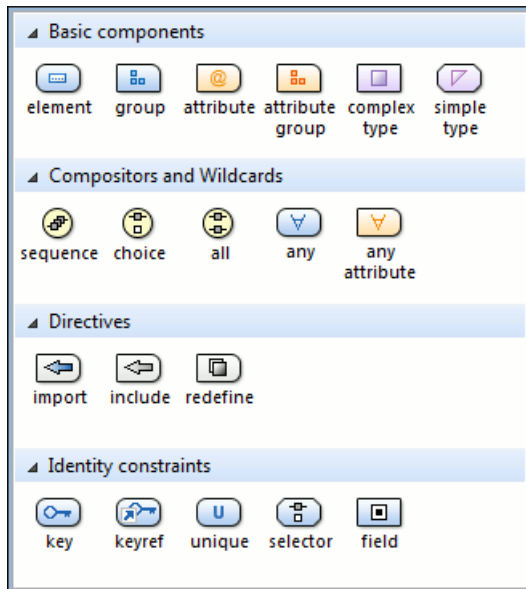
Editing Patterns

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

The Palette View

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

Figure 55: Palette View





Components are organized functionally into 4 collapsible categories:

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

To add a component to the edited schema:

- click and hold a graphic symbol from the **Palette** view, then drag the component into the **Design** view;
- a line dynamically connects the component with the XML schema structure;
- release the component into a valid position.

 **Note:** You cannot drop a component into an invalid position. When you hover the component into an invalid position, the mouse cursor changes its shape into . Also, the connector line changes its color from the usual dark grey to the color defined in the *Validation error highlight color* option (default color is red).

Edit Schema Namespaces

You can use the dialog **XML Schema Namespaces** to easily set a target namespace and define namespace mappings for a newly created XML Schema. In the **Design** 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.

Contextual Editing

Smaller interrelated modules that define a complex XML Schema cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, a function defined in a main schema document is not visible when you edit an included or imported module. Oxygen 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.

To set a main schema files, you need to define a validation scenario and 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;

Create an XML Schema From a Relational Database Table

To create an XML Schema from the structure of a relational database table use *the special wizard available in the Tools menu*.

Generate Sample XML Files

To generate sample XML files from an XML Schema, use the **Generate Sample XML Files...** action. It is also available on the contextual menu from the schema *Design mode*.

The Schema Tab

W3C XML Schema

URL: file:/D:/runtime-New_configuration/OxygenEclipseSamples/s

Namespace: http://schemas.openxmlformats.org/package/2006/content-types

Root Element: Default

Output folder: D:\runtime-New_configuration\OxygenEclipseSamples\samples\framework

Filename prefix: instance Extension: xml

Number of instances: 1

Open first instance in editor

Namespaces

Default Namespace: <NO_NAMESPACE>

Namespace prefix	Namespace
p	http://schemas.openxmlformats.org/presentationml/2006/main
a	http://schemas.openxmlformats.org/drawingml/2006/main
r	http://schemas.openxmlformats.org/officeDocument/2006/rela...
m	http://schemas.openxmlformats.org/officeDocument/2006/math
sl	http://schemas.openxmlformats.org/schemaLibrary/2006/main
wp	http://schemas.openxmlformats.org/drawingml/2006/wordpro...
dc	http://purl.org/dc/elements/1.1/
dcmitype	http://purl.org/dc/dcmitype/
cdr	http://schemas.openxmlformats.org/drawingml/2006/chartDra...
vt	http://schemas.openxmlformats.org/officeDocument/2006/doc...
dcterms	http://purl.org/dc/terms/
w	http://schemas.openxmlformats.org/wordprocessingml/2006/...

Load settings Export settings

Figure 56: The Generate Sample XML Files Dialog

Complete the dialog as follows:

- **URL** - Schema location as an URL. A history of the last used URLs is available in the drop-down box.
- **Namespace** - Displays the namespace of the selected schema.
- **Document root** - After the schema is selected, this drop-down box is populated with all root candidates gathered from the schema. Choose the root of the output XML documents.
- **Output folder** - Path to the folder where the generated XML instances will be saved.
- **Filename prefix and Extension** - Generated file names have the following format: `prefixN.extension`, where N represents an incremental number from 0 up to *Number of instances - 1*.
- **Number of instances** - The number of XML files to be generated.
- **Open first instance in editor** - When checked, the first generated XML file is opened in editor.
- **Namespaces** - Here you can specify the default namespace as well as the proxies (prefixes) for namespaces.
- **Load settings / Export settings** - The current settings can be saved for further usage with the **Export settings** button, and reloaded when necessary with the **Load settings** button.

The Options Tab

The **Options** tab allows you to set specific options for different namespaces and elements.

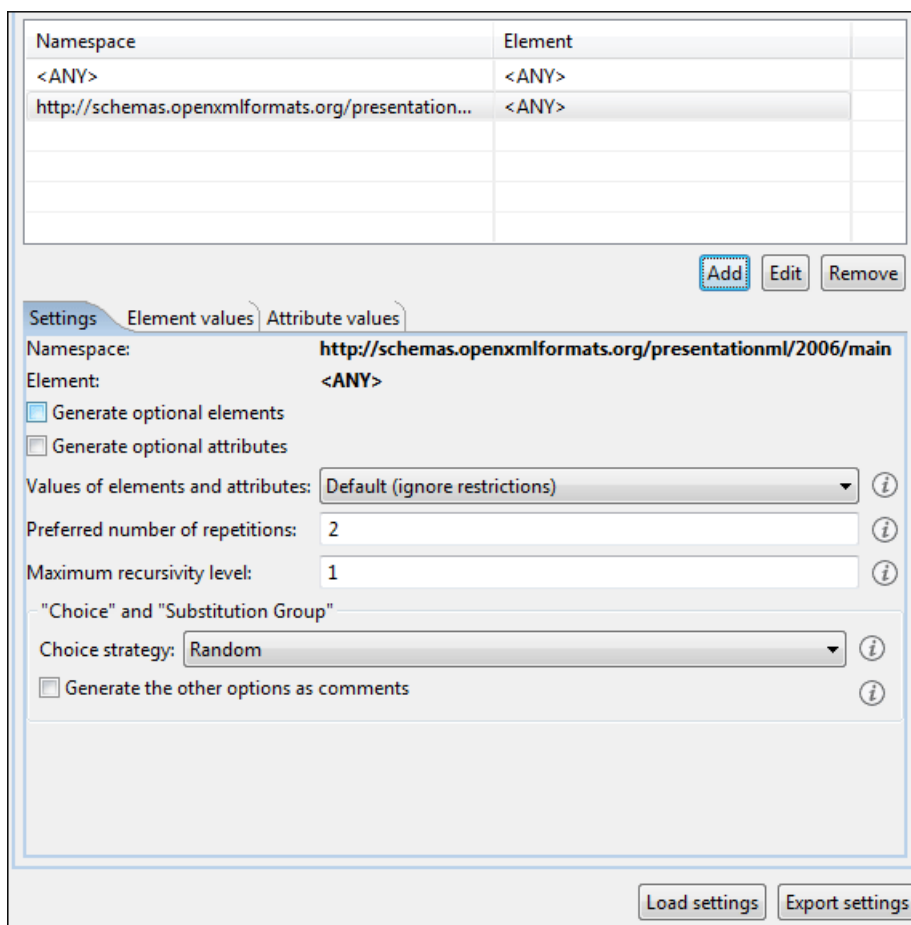


Figure 57: The Generate Sample XML Files Dialog

- **Namespace / Element table** - Allows you to set a namespace for each element name that appears in an XML document instance. The following prefix-to-namespace associations are available:
 - All elements from all namespaces (<ANY> - <ANY>). This is the default setting and can be customized from the *XML Instances Generator* preferences page.
 - All elements from a specific namespace.
 - A specific element from a specific namespace.
- **Settings**
 - **Generate optional elements** - When checked, all elements are generated, including the optional ones (having the `minOccurs` attribute set to 0 in the schema).
 - **Generate optional attributes** - When checked, all attributes are generated, including the optional ones (having the `use` attribute set to `optional` in the schema.)
 - **Values of elements and attributes** - Controls the content of generated attribute and element values. Several choices are available:
 - **None** - No content is inserted;
 - **Default** - Inserts a default value depending of data type descriptor of the particular element or attribute. The default value can be either the data type name or an incremental name of the attribute or element (according to the global option from the **XML Instances Generator** preferences page). Note that type restrictions are ignored when this option is enabled. For example, if an element is of a type that restricts an **xs:string** with the **xs:maxLength** facet in order to allow strings with a maximum length of 3, the XML instance generator tool may generate string element values longer than 3 characters.
 - **Random** - Inserts a random value depending of data type descriptor of the particular element or attribute.

**Important:**

If all of the following are true, the **XML Instances Generator** outputs invalid values:

- at least one of the restrictions is a regexp;
- the value generated after applying the regexp does not match the restrictions imposed by one of the facets.

This limitation leads to attributes or elements with values set to *Invalid*.

- **Preferred number of repetitions** - Allows the user to set the preferred number of repeating elements related with `minOccurs` and `maxOccurs` facets defined in XML Schema.
 - If the value set here is between `minOccurs` and `maxOccurs`, then that value is used;
 - If the value set here is less than `minOccurs`, then the `minOccurs` value is used;
 - If the value set here is greater than `maxOccurs`, then that value is used.
- **Maximum recursion level** - If a recursion is found, this option controls the maximum allowed depth of the same element.
- **Choice strategy** - Option used in case of `xs:choice` or `substitutionGroup` elements. The possible strategies are:
 - **First** - the first branch of `xs:choice` or the head element of `substitutionGroup` is always used;
 - **Random** - a random branch of `xs:choice` or a substitute element or the head element of a `substitutionGroup` is used.
- **Generate the other options as comments** - Option to generate the other possible choices or substitutions (for `xs:choice` and `substitutionGroup`). These alternatives are generated inside comments groups so you can uncomment and use them later. Use this option with care (for example on a restricted namespace and element) as it may generate large result files.
- **Load settings / Export settings** - The current settings can be saved for further usage with the **Export settings** button, and reloaded when necessary with the **Load settings** button.
- **Element values** - The **Element values** tab allows you to add values that are used to generate the elements content. If there are more than one value, then the values are used in a random order.
- **Attribute values** - The **Attribute values** tab allows you to add values that are used to generate the attributes content. If there are more than one value, then the values are used in a random order.

The Advanced Tab

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

This tab allows you to set advanced options that controls the output values of elements and attributes.

- **Use incremental attribute / element names as default** - If checked, the value of an element or attribute starts with the name of that element or attribute. For example, for an `a` element the generated values are: `a1`, `a2`, `a3`, etc. If not checked, the value is the name of the type of that element / attribute, for example: `string`, `decimal`, etc.
- **Maximum length** - The maximum length of string values generated for elements and attributes.
- **Discard optional elements after nested level** - The optional elements that exceed the specified nested level are discarded. This option is useful for limiting deeply nested element definitions that can quickly result in very large XML documents.

XML Schema Regular Expressions Builder

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

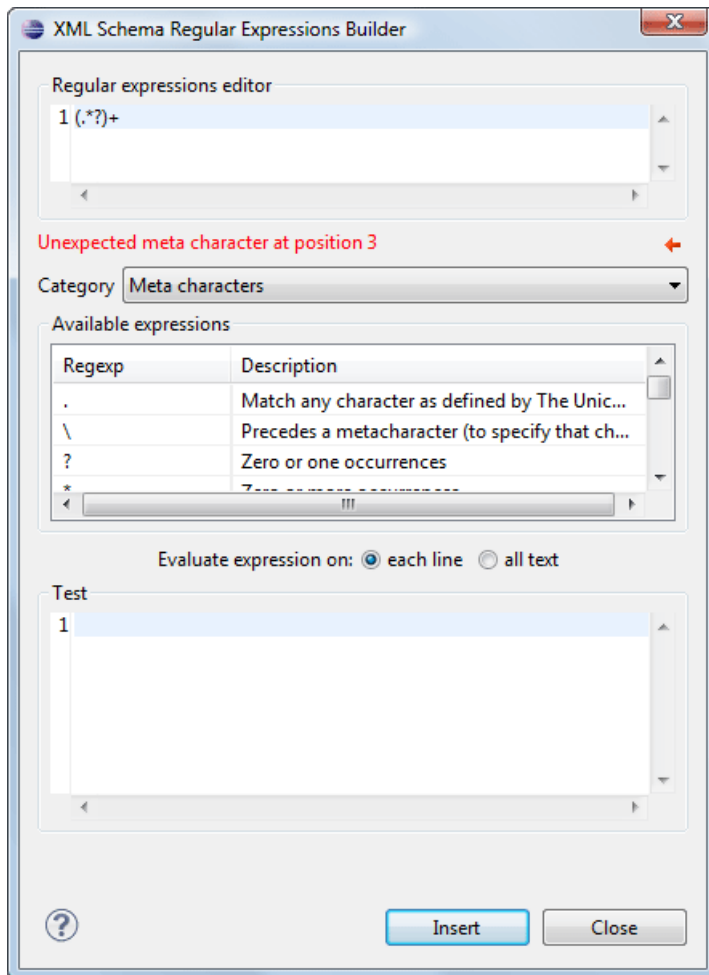


Figure 58: XML Schema Regular Expressions Builder Dialog

The dialog contains the following sections:

- **Regular expressions editor** - allows you to edit the regular expression to be tested and used. Content completion is available and presents a list with all the predefined expressions. It is triggered by pressing **(Ctrl - Space)**.
- **Error display area** - if the edited regular expression is incorrect, an error message will be displayed here. The message contains the description and the exact location of the error. Also, a click on the quick navigation button (←) highlights the error inside the regular expression.
- **Category** combo box - here you can choose from several categories of predefined expressions. The selected category influences the displayed expressions in the **Available expressions** table.
- **Available expressions** table - holds the available regular expressions and a short description for each of them. The set of expressions depends on the category selected in the previous combo box. You can add an expression in the **Regular expressions editor** by double-clicking on the expression row in the table. You will notice that in the case of **Character categories** and **Block names** the expressions are also listed in complementary format. For example: `\p{Lu}` - Uppercase letters; `\P{Lu}` - Complement of: Uppercase letters.
- **Evaluate expression on** radio buttons - there are available two options:
 - **Evaluate expression on each line** - the edited expression will be applied on each line in the **Test** area;
 - **Evaluate expression on all text** - the edited expression will be applied on the whole text.

- **Test** area - a text editor which allows you to enter a text sample on which the regular expression will be applied. All matches of the edited regular expression will be highlighted.

After editing and testing your regular expression you can insert it in the current editor. The **Insert** button will become active when an editor is opened in the background and there is an expression in the **Regular expressions editor**.

The regular expression builder cannot be used to insert regular expressions in *the grid version* or *the schema version* of a document editor. Accordingly, the **Insert** button of the dialog will be disabled if the current document is edited in grid mode.

- ☞ **Note:** Some regular expressions may block indefinitely the Java Regular Expressions engine. If the execution of the regular expression does not end in about five seconds, the application displays a dialog that allows you to interrupt the operation.

Generating Documentation for an XML Schema

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

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

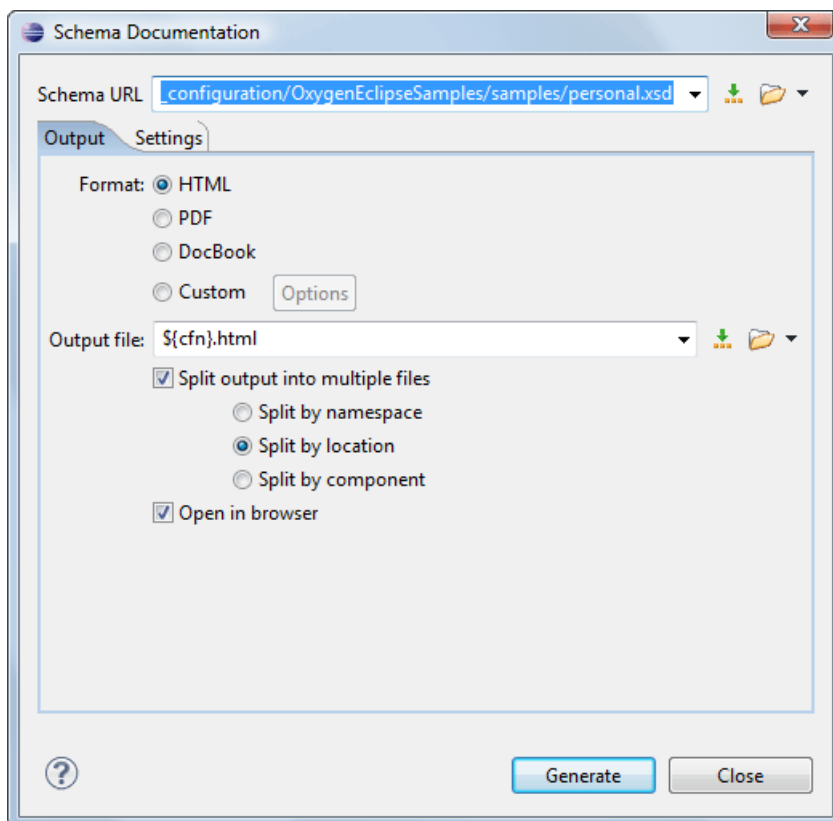


Figure 59: The Output panel of the Schema Documentation Dialog

The **Schema URL** field of the dialog panel must contain the full path to the XML Schema (XSD) file you want to generate documentation for. The schema may be a local or a remote one. You can specify the path to the schema using the editor variables.

The following options are available in the **Settings** tab:

- **Format** - Allows you to choose between three predefined formats (**HTML**, **PDF**, **DocBook**) and a custom one (**Custom**). This allows you to control the output format by providing a custom stylesheet.
- **Output file** - Name of the output file.
- **Split output into multiple files** - Instructs the application to split the output into multiple files. You can choose to split them by namespace, location or component name.
- **Open in Browser/System Application** - Opens the result in the system application associated with the output file type.



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

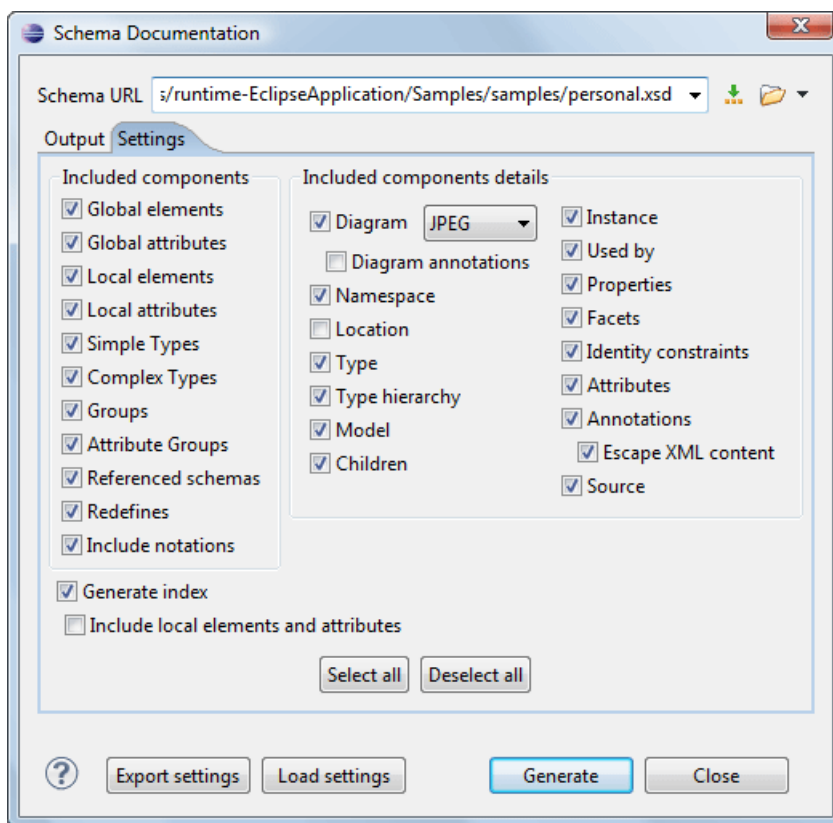


Figure 60: The Settings Panel of the Schema Documentation Dialog

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

- **Diagram** - Displays the diagram for each component. You can choose the image format (JPEG, PNG, SVG) to use for the diagram section.
- **Diagram annotations** - This option controls whether or not the annotations of the components presented in the diagram sections should be included.
- **Namespace** - Displays the namespace for each component.
- **Location** - Displays the schema location for each component.
- **Type** - Displays the component type if it is not an anonymous one.
- **Type hierarchy** - Displays the types hierarchy.

- **Model** - Displays the model (sequence, choice, all) presented in BNF form. Different separator characters are used depending on the information item used:
 - `xs:all` - its children will be separated by space characters;
 - `xs:sequence` - its children will be separated by comma characters;
 - `xs:choice` - its children will be separated by / characters.
- **Children** - Displays the list of component's children.
- **Instance** - Displays an XML instance generated based on each schema element.
- **Used by** - Displays the list of all the components that refer the current one. The list is sorted by component type and name.
- **Properties** - Displays some of the component's properties.
- **Facets** - Displays the facets for each simple type
- **Identity constraints** - Displays the identity constraints for each element. For each constraint there are presented the name, type (unique, key, keyref), refer attribute, selector and field(s).
- **Attributes** - Displays the attributes for the component. For each attribute there are presented the name, type, fixed or default value, usage and annotation.
- **Annotations** - Displays the annotations for the component. If you choose **Escape XML Content**, the XML tags are present in the annotations.
- **Source** - Displays the text schema source for each component.
- **Generate index** - Displays an index with the components included in the documentation.
- **Include local elements and attributes** - If checked, local elements and attributes are included in the documentation index.
- **Export settings / Load settings** - The current settings can be saved for further usage (for example for generating documentation from command-line interface) with the **Export settings** button, and reloaded when necessary with the **Load settings** button.

These options are persistent between sessions.

Generate Documentation in HTML Format

The HTML documentation contains images corresponding to the schema definitions as the ones displayed by *the schema diagram editor*. These images are divided in clickable areas which are linked to the definitions of the clicked names of types or elements. The documentation of a definition includes a **Used By** section with links to the other definitions which refer to it. If the **Escape XML Content** option is unchecked, the HTML or XHTML tags used inside the `xs:documentation` elements of the input XML Schema for formatting the documentation text (for example ``, `<i>`, `<u>`, ``, ``, etc.) are rendered in the generated HTML documentation.

The generated images format is **PNG**. The image of an XML Schema component contains the graphical representation of that component as it is rendered in *the Schema Diagram panel of the Oxygen XML Developer plugin's XSD editor panel*.

The screenshot shows a web-based interface for schema documentation. On the left, there is a 'Table of Contents' panel with a 'Group by:' dropdown set to 'Location'. Below it, a tree view shows the schema structure under 'personal.xsd', including elements like 'email', 'family', 'given', 'link', 'name', 'person', 'personnel', and 'uri'. The main area is titled 'Element name' and displays the following information:

- Namespace:** No namespace
- Annotations:** Specifies the person family and given name.
- Diagram:** A UML diagram showing the 'name' element containing 'family' and 'given' elements, both of type 'xs:string'.
- Properties:** Content: complex
- Used by:** Element: person
- Model:** ALL(family given)
- Children:** family, given
- Instance:**

```
<name>
  <family>{1,1}</family>
  <given>{1,1}</given>
</name>
```
- Source:**

```
<xs:element name="name">
  <xs:annotation>
    <xs:documentation>Specifies the person family and given name.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:all>
      <xs:element ref="family"/>
      <xs:element ref="given"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

On the right, a 'Showing:' panel lists various features that are checked, including Annotations, Attributes, Diagrams, Facets, Identity Constraints, Instances, Properties, Source, and Used by.

Figure 61: Schema Documentation Example

The generated documentation includes a table of contents. You can group the contents by namespace, location, or component type. After the table of contents there is presented some information about the main schema, the imported, included, and redefined schemas. This information contains the schema target namespace, schema properties (attribute form default, element form default, version) and schema location.

Namespace	No namespace
Properties	Attribute Form Default: unqualified
	Element Form Default: unqualified
Schema location	file:D:/personal.xsd

Figure 62: Information About a Schema

If you choose to split the output into multiple files, the table of contents is displayed in the left frame. The contents are grouped in the same mode. If you split the output by location, each file contains a schema description and the components that you have chosen to include. If you split the output by namespace, each file contains information about schemas from that namespace and the list with all included components. If you choose to split the output by component, each file contains information about a schema component.

After the documentation is generated you can collapse details for some schema components. This can be done using the **Showing** view:

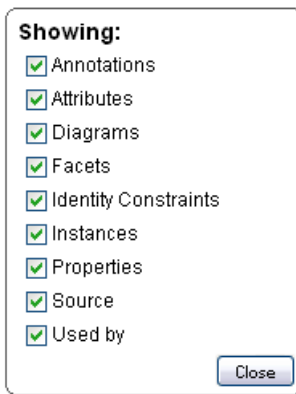


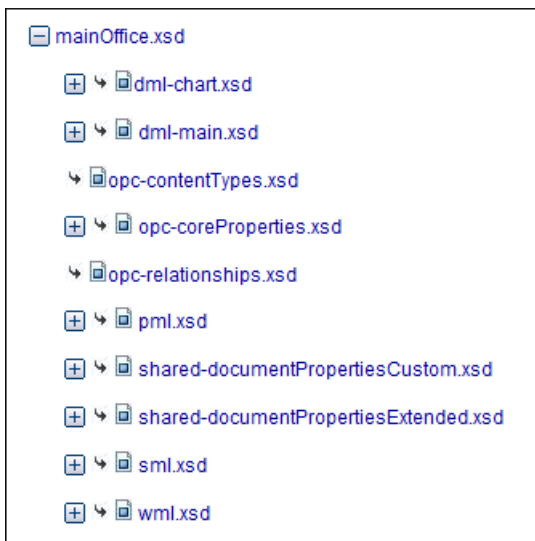
Figure 63: The Showing View

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

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

Figure 64: Documentation for a Schema Component

If the schema contains imported or included modules, their dependencies tree is generated in the documentation.



Generate Documentation in PDF, DocBook or a Custom Format

Schema documentation can be also generated in PDF, DocBook, or a custom format. You can choose the format from the *Schema Documentation* Dialog. For the PDF and DocBook formats, the option to split the output in multiple files is disabled.

When choosing PDF, the documentation is generated in DocBook format and after that a transformation using the FOP processor is applied to obtain the PDF file. To configure the FOP processor, see the *FO Processors* preferences page.

If you generate the documentation in DocBook format you can apply a transformation scenario on the output file, for example one of the scenarios proposed by Oxygen 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 `{INSTALLATION_DIRECTORY}/frameworks/schema_documentation`. You can create a custom format starting from one of the stylesheets used in the predefined HTML, PDF, and DocBook formats. These stylesheets are available in `{INSTALLATION_DIRECTORY}/frameworks/schema_documentation/xsl`.

When using a custom format you can also copy additional resources into the output folder or choose to keep the intermediate XML files created during the documentation process.

Generating Documentation From the Command-Line Interface

You can export the settings of the **Schema Documentation** dialog to an XML file by pressing the **Export settings** button. With the exported settings file, you can generate the same documentation from the command-line interface by running the following scripts:

- `schemaDocumentation.bat` on Windows;
- `schemaDocumentation.sh` (on Mac OS X / Unix / Linux).

The scripts are located in the Oxygen 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>
            <field name="includeLocalAttributes">
                <Boolean xml:space="preserve">>true</Boolean>
            </field>
            <field name="includeSimpleTypes">
                <Boolean xml:space="preserve">>true</Boolean>
            </field>
            <field name="includeComplexTypes">
                <Boolean xml:space="preserve">>true</Boolean>
            </field>
            <field name="includeGroups">
                <Boolean xml:space="preserve">>true</Boolean>
            </field>
            <field name="includeAttributesGroups">
                <Boolean xml:space="preserve">>true</Boolean>
            </field>
            <field name="includeRedefines">
                <Boolean xml:space="preserve">>true</Boolean>
            </field>
            <field name="includeReferencedSchemas">
                <Boolean xml:space="preserve">>true</Boolean>
            </field>
            <field name="detailsDiagram">
                <Boolean xml:space="preserve">>true</Boolean>
            </field>
            <field name="detailsNamespace">
                <Boolean xml:space="preserve">>true</Boolean>
            </field>

```


```


<field name="detailsLocation">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsType">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsTypeHierarchy">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsModel">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsChildren">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsInstance">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsUsedby">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsProperties">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsFacets">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsAttributes">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsIdentityConstr">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsEscapeAnn">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsSource">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
<field name="detailsAnnotations">
  <Boolean xml:space="preserve">>true</Boolean>
</field>
</xsdDocumentationOptions>
</entry>
</map>
</serialized>

```

Searching and Refactoring Actions

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

- **XSD** >  > **References (Alt+Shift+S R) ((Cmd+Alt+S R on Mac OS))** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog is shown and you have the possibility to define another search scope.
- **contextual menu of current editor** > **Search** > **References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog.

- **XSD** >  > **Declarations** (**Alt+Shift+S D**) ((**Cmd+Alt+S D on Mac OS**)) - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog will be shown and you have the possibility to define another search scope.

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

- **contextual menu of current editor** > **Search** > **Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when define a scope in the dialog above. Action is not available in **Design** mode.
- **XSD** > **Occurrences in File** (**Alt+Shift+S O**) ((**Cmd+Alt+S O on Mac OS**)) - Searches all occurrences of the item at the caret position in the currently edited file.
- **contextual menu of current editor** > **Rename Component...** - Renames the selected component. Specify the new component name and the file or files affected by the modification in the following dialog:

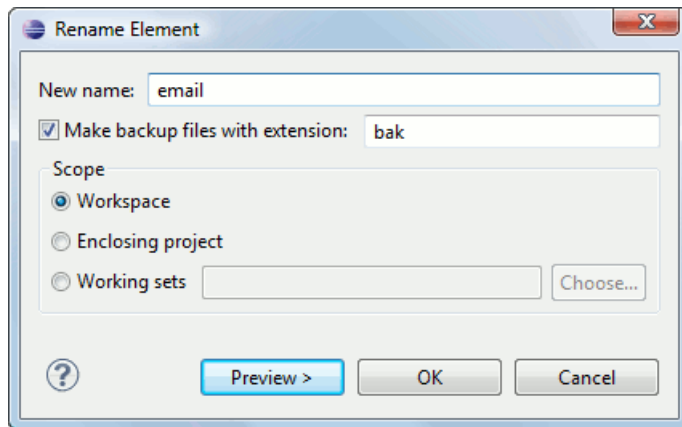


Figure 65: Rename Component Dialog

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

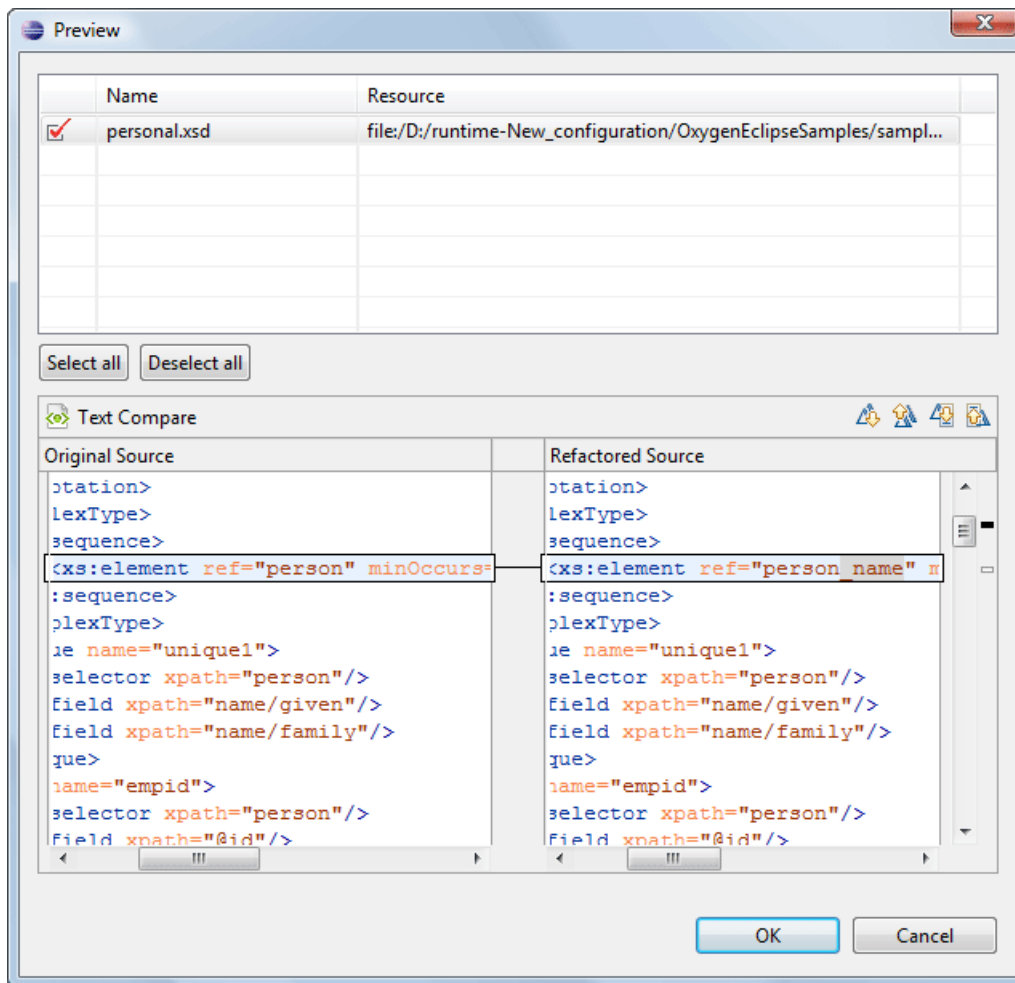


Figure 66: Preview Dialog

Search and Refactor Operations Scope


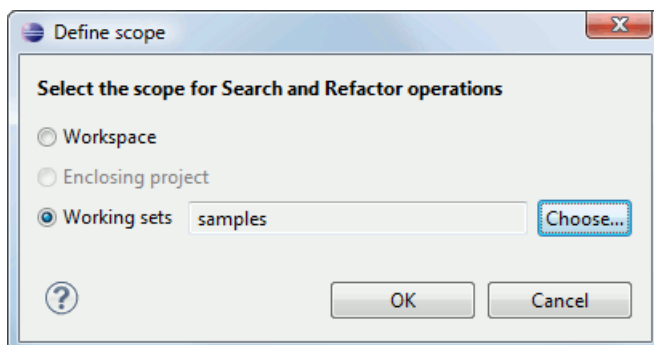
By *scope* you should understand the collection of documents that define the context of the search and refactor operations. To control it you can use the  **Change scope** operation, available in the Quick Assist action set or on the **Resource Hierarchy/Dependency View** toolbar. Here you can restrict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

Figure 67: Change Scope Dialog



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

Resource Hierarchy / Dependencies View

The **Resource Hierarchy / Dependencies** view allows you to easily see the hierarchy / dependencies for an XML Schema, a *Relax NG schema* or an *XSLT stylesheet*. You can open the view from **Window > Show View > Other > oXygen > Resource Hierarchy/Dependencies**.

This view is useful for example when you want to start from an XML Schema (XSD) file and build and review the hierarchy of all the other XSD files that are imported, included or redefined in the given XSD file. Also the same view is able to build the inversed-tree structure, that is the structure of all other XSD files that import, include or redefine the given XSD file. The scope of the search is configurable: the current Oxygen project, a set of local folders, etc.

The build process for the hierarchy view is started with the **Resource Hierarchy** action available on the contextual menu of the editor panel.

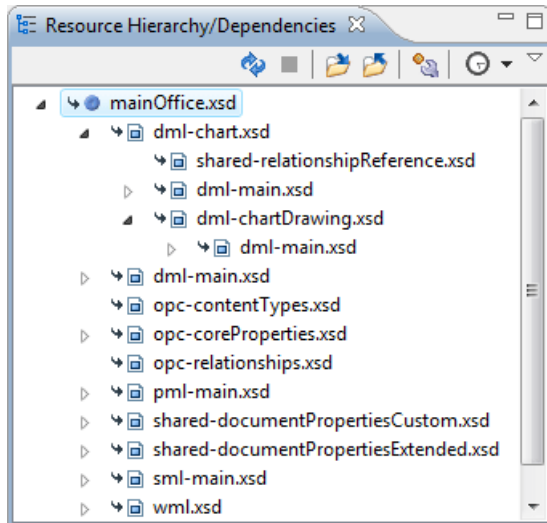


Figure 68: Resource Hierarchy/Dependencies View - Hierarchy for mainOffice.xsd

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

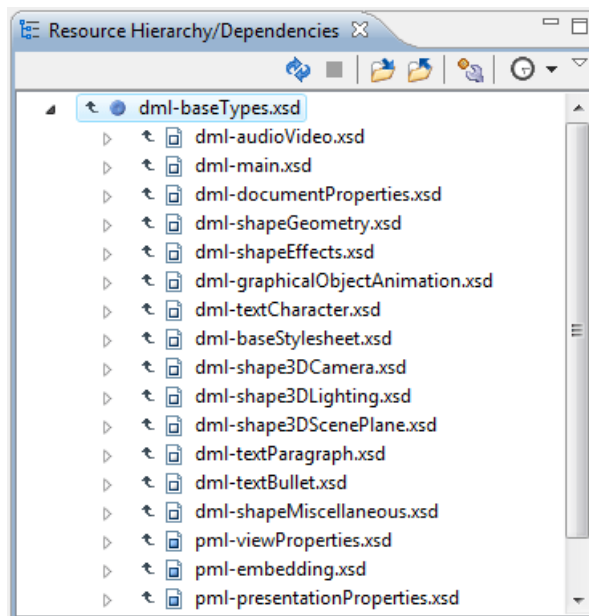










Figure 69: Resource Hierarchy/Dependencies View - Dependencies for dml-baseTypes.xsd

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

-  - Refreshes the Hierarchy/Dependencies structure.
-  - Stop the hierarchy/dependencies computing.
-  - Allows you to choose a schema to compute the hierarchy structure.
-  - Allows you to choose a schema to compute the dependencies structure.
-  - Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.
-  - Repeats a previous dependencies computation.

The contextual menu contains the following actions:

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

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

Component Dependencies View

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

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

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

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

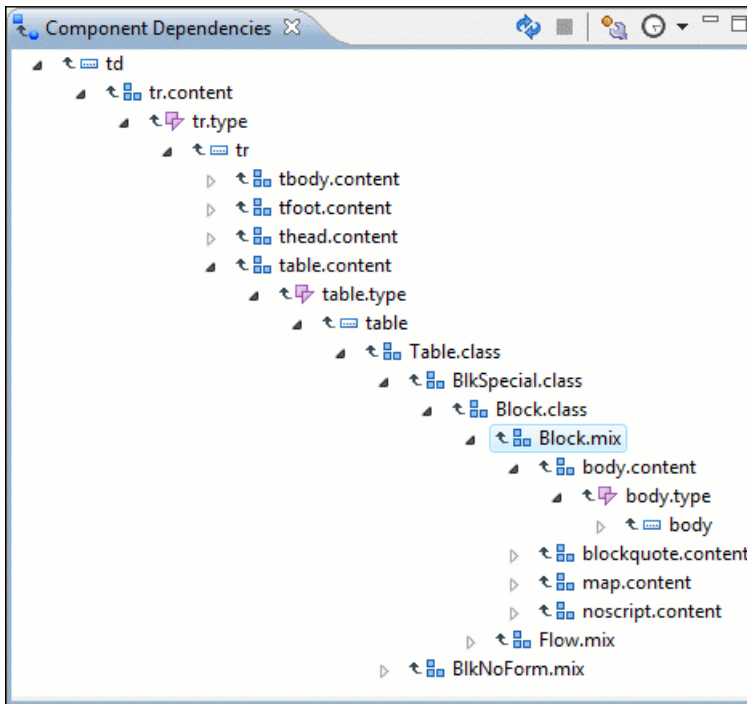


Figure 70: Component Dependencies View - Hierarchy for xhtml111.xsd

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

- - Refreshes the dependencies structure.
- - Stop the dependencies computing.
- - Allows you to configure a search scope to compute the dependencies structure.
- - Contains a list of previously executed dependencies computations.

The contextual menu contains the following actions:

- **Go to First Reference** - Selects the first reference of the referred component from the current selected component in the dependencies tree.
- **Go to Component** - Shows the definition of the currently selected component in the dependencies tree.

Tip: If a component contains multiple references to another components, a small table is shown containing all these references. When a recursive reference is encountered, it is marked with a special icon .

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 set on automatic search by default and can be configured in the **Options > Preferences > Editor > Mark Occurrences** page. A search can also be triggered with the **Search > Search Occurrences in File (Ctrl+Shift+U)** contextual menu action. All matches are displayed in a separate tab of the **Results** view.

XML Schema Quick Assist Support

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

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

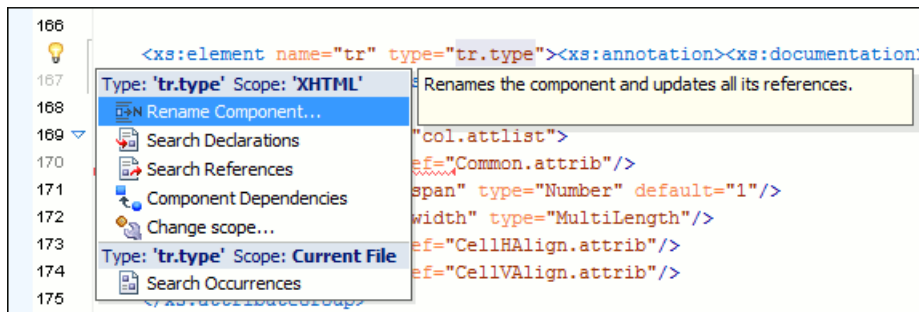


Figure 71: Quick Assist Support

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

- **Rename Component** - Renames the component and all its dependencies;
- **Search Declaration** - Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference;
- **Search References** - Searches all references of the component in a predefined scope;
- **Component Dependencies** - Searches the component dependencies in a predefined scope;
- **Change Scope** - Configures the scope that will be used for future search or refactor operations;
- **Search Occurrences** - Searches all occurrences of the file within the current scope.

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.

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**.

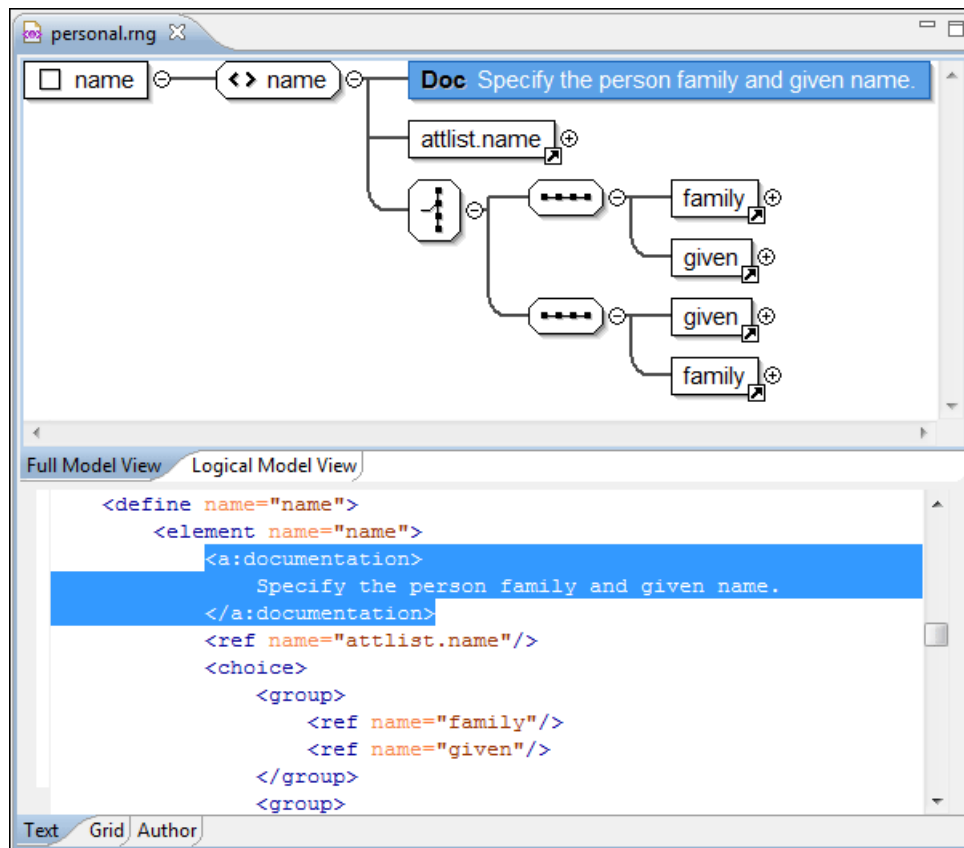


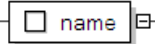
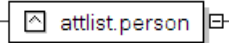

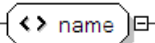
Figure 72: Relax NG Schema Editor - Full Model View

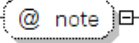
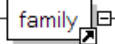




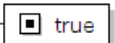

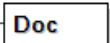


The following references can be expanded in place: patterns, includes, and external references. This expansion mechanism, coupled with the synchronization support, makes the schema navigation easy.

All the element and attribute names are editable: double-click any name to start editing it.

Symbols Used in the Schema Diagram

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

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

-  - attribute pattern with the name attribute set to the value shown inside the rectangle (in this case note).
-  - ref pattern with the name attribute set to the value shown inside the rectangle (in this case family).
-  - oneOrMore pattern.
-  - zeroOrMore pattern.
-  - optional pattern.
-  - choice pattern.
-  - value pattern, used for example inside a choice pattern.
-  - group pattern.
-  - pattern from the Relax NG Annotations namespace (<http://relaxng.org/ns/compatibility/annotations/1.0>) which is treated as a documentation element in a Relax NG schema.
-  - text pattern.
-  - empty pattern.

Logical Model View

The **Logical Model View** presents the compiled schema which is a single pattern. The patterns that form the element content are defined as top level patterns with generated names. These names are generated depending of the elements name class.

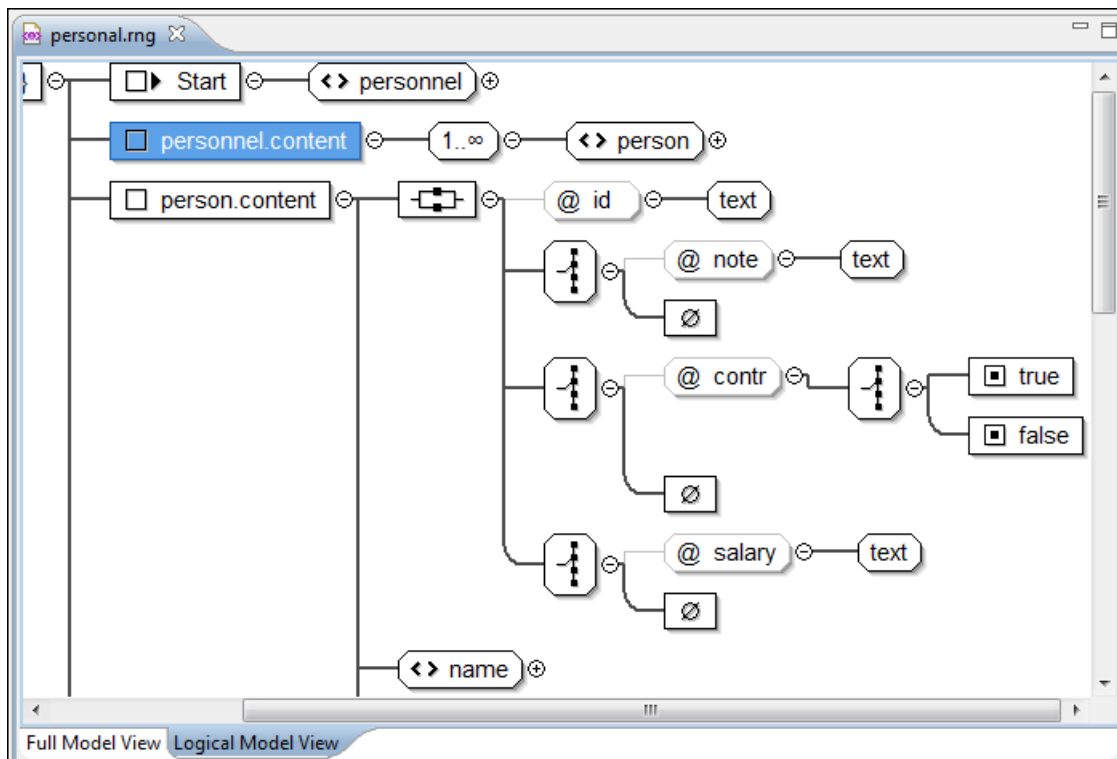


Figure 73: Logical Model View for a Relax NG Schema

Actions Available in the Diagram View


The contextual menu offers the following actions:

- **Append child** - Appends a child to the selected component.
- **Insert Before** - Inserts a component before the selected component.
- **Insert After** - Inserts a component after the selected component.
- **Edit attributes** - Edits the attributes of the selected component.
- **Remove** - Removes the selected component.
- **Show only the selected component** - Depending on its state (selected/not selected), either the selected component or all the diagram components are shown.
- **Show Annotations** - Depending on its state (selected/not selected), the documentation nodes are shown or hidden.
- **Auto expand to references** - This option controls how the schema diagram is automatically expanded. If you select it and then edit a top-level element or you make a refresh, the diagram is expanded until it reaches referred components. If this option is left unchecked, only the first level of the diagram is expanded, showing the top-level elements. For large schemas, the editor disables this option automatically.
- **Collapse Children** - Collapses the children of the selected view.
- **Expand Children** - Expands the children of the selected view.
- **Print Selection...** - Prints the selected view.
- **Save as Image...** - Saves the current selection as JPEG, BMP, SVG or PNG image.
- **Refresh** - Refreshes the schema diagram according to the changes in your code. They represent changes in your imported documents or changes that are not reflected automatically in the compiled schema).

If the schema is not valid, you see only an error message in the **Logical Model View** instead of the diagram.

Relax NG Outline View

The Relax NG **Outline** view presents a list with the patterns that appear in the diagram in both the **Full Model View** and **Logical Model View** cases. It allows a quick access to a component by name. By default it is displayed on screen. If you closed the **Outline** view you can reopen it from menu **Window > Show View > Other > oXygen > Outline**.

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

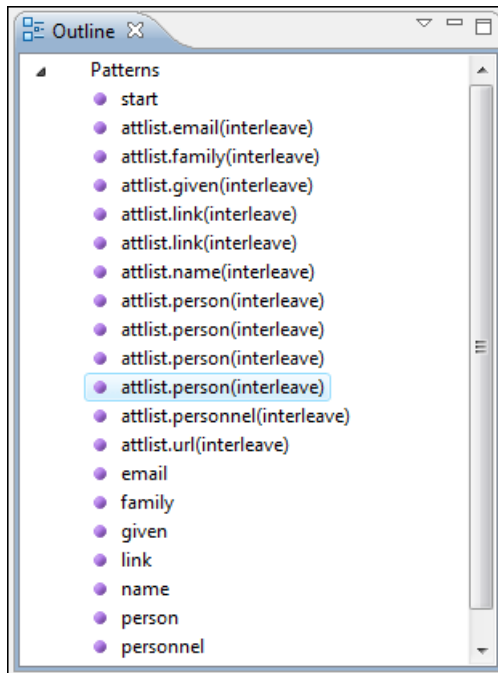









Figure 74: Relax NG Outline View

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

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

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

-  **Selection update on caret move** - Allows a synchronization between **Outline** view and schema diagram. The selected view from the diagram will be also selected in the **Outline** view.
-  **Show components** - Shows the define patterns collected from the current document.
-  **Flat presentation mode of the filtered results** - when active, the application flattens the filtered result elements to a single level.
-  **Show comments and processing instructions** - show/hide comments and processing instructions in the **Outline** view.
-  **Show text** - show/hide additional text content for the displayed elements.
-  **Show attributes** - show/hide attribute values for the displayed elements. The displayed attribute values can be changed from [the Outline preferences panel](#).
-  **Configure displayed attributes** - displays the [XML Structured Outline preferences page](#).

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

Relax NG Editor Specific Actions

The list of actions specific for the Relax NG (full syntax) editor is:

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

Searching and Refactoring Actions

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

- **RNG > Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this scope, a warning dialog is shown and you can define another search scope. A search scope includes the project or a collection of files and folders.

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

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

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

- **RNG > Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this scope a warning dialog is shown and you can define another search scope. A search scope includes the project or a collection of files and folders.
- **contextual menu of current editor > Search > Search Declarations in...** - Searches all declarations of the item found at current cursor position in the files specified in the search scope.
- **RNG > Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.
- **contextual menu of current editor > Refactoring > Rename Component...** - Renames the selected component.

Resource Hierarchy/Dependencies View

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

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

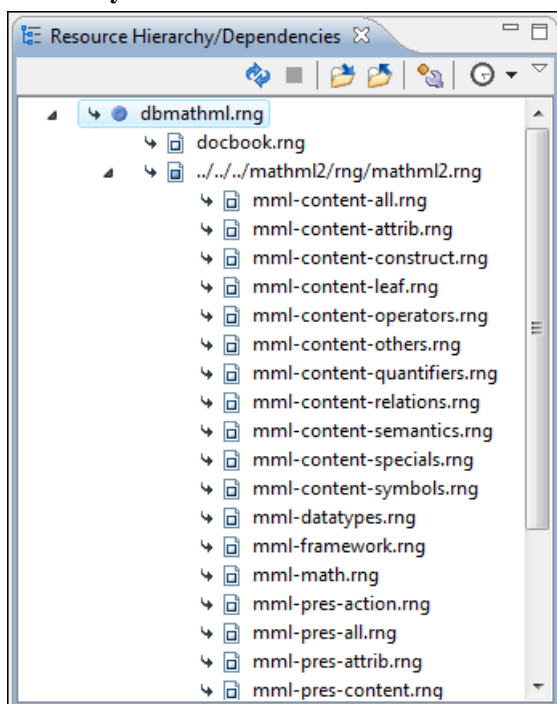


Figure 75: Resource Hierarchy/Dependencies View - hierarchy for `dbmathml.rng`

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

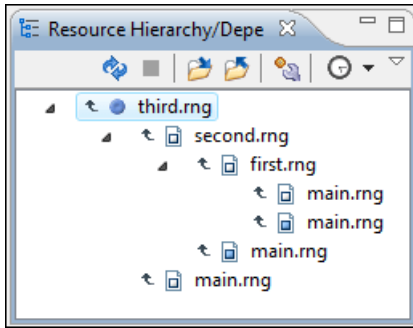


Figure 76: Resource Hierarchy/Dependencies View - Dependencies for third.rng

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

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

The following actions are available in the contextual menu:

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

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

Component Dependencies View

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

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

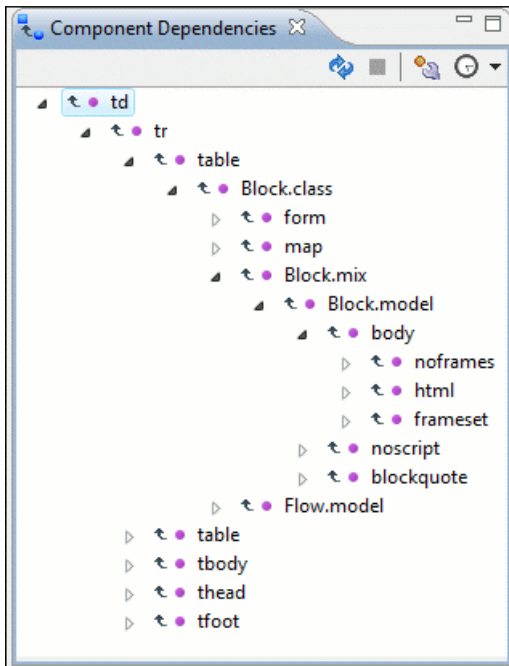








Figure 77: Component Dependencies View - Hierarchy for xhtml.rng

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

-  - Refreshes the dependencies structure.
-  - Allows you to stop the dependencies computing.
-  - Allows you to configure a search scope to compute the dependencies structure.
-  - Allows you to repeat a previous dependencies computation.

The following actions are available on the contextual menu:

- **Go to First Reference** - Selects the first reference of the referred component from the current selected component in the dependencies tree.
- **Go to Component** - Shows the definition of the current selected component in the dependencies tree.

 **Tip:** If a component contains multiple references to another components, a small table is shown containing all references. When a recursive reference is encountered, it is marked with a special icon .

RNG Quick Assist Support

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

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

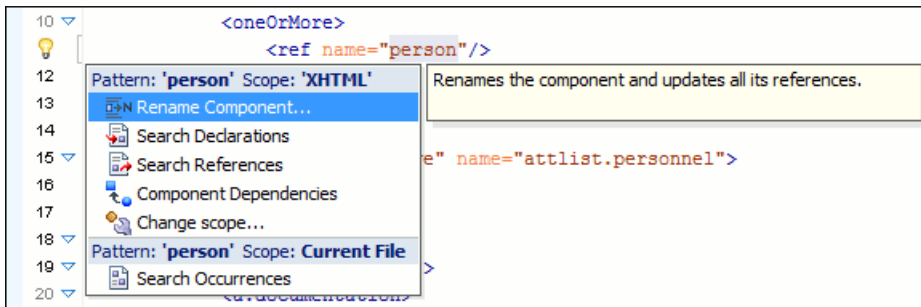


Figure 78: RNG Quick Assist Support

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

- **Rename Component** - Renames the component and all its dependencies;
- **Search Declaration** - Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference;
- **Search References** - Searches all references of the component in a predefined scope;
- **Component Dependencies** - Searches the component dependencies in a predefined scope;
- **Change Scope** - Configures the scope that will be used for future search or refactor operations;
- **Search Occurrences** - Searches all occurrences of the file within the current scope.

Configuring a Custom Datatype Library for a RELAX NG Schema

A RELAX NG schema can declare a custom datatype library for the values of elements found in XML document instances. The datatype library must be developed in Java and it must implement the interface [specified on the www.thaiopensource.com website](http://www.thaiopensource.com).

The jar file containing the custom library and any other dependent jar file must be added to the classpath of the application, that is the jar files must be added to the folder [Oxygen-plugin-folder]/lib and a line `<library name="lib/custom-library.jar"/>` must be added for each jar file to the file [Oxygen-plugin-folder]/plugin.xml in the `<runtime>` element.

To load the custom library, restart the Eclipse platform.

Editing NVDL Schemas

Some complex XML documents are composed by combining elements and attributes from different namespaces. More, the schemas which define these namespaces are not even developed in the same schema language. In such cases, it is difficult to specify in the document all the schemas which must be taken into account for validation of the XML document or for content completion. An NVDL (Namespace Validation Definition Language) schema can be used. This schema allows the application to combine and interleave multiple schemas of different types (W3C XML Schema, RELAX NG schema, Schematron schema) in the same XML document.

Oxygen 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**.

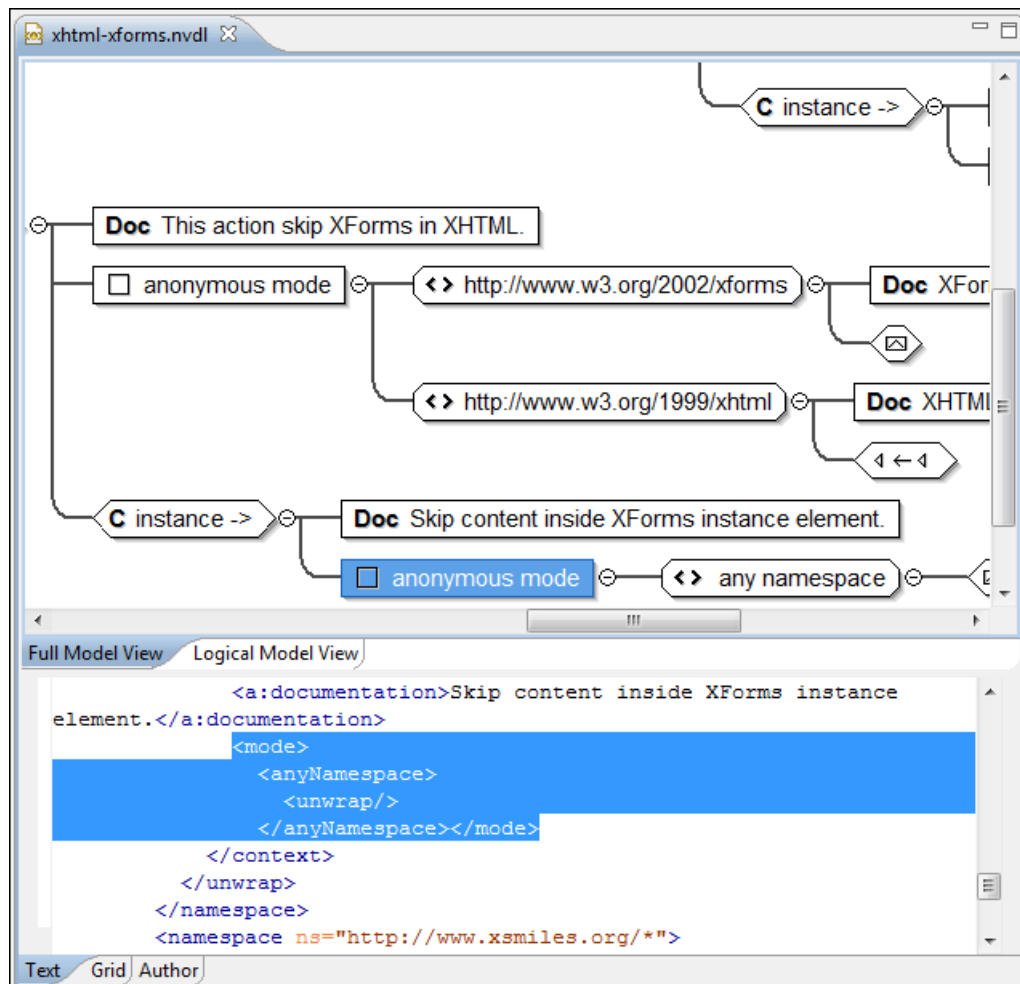


Figure 79: NVDL Schema Editor - Full Model View

The **Full Model View** renders all the NVDL elements with intuitive icons. This representation coupled with the synchronization support makes the schema navigation easy.

Double click on any diagram component in order to edit its properties.

Actions Available in the Diagram View

The contextual menu offers the following actions:

- **Show only the selected component** - Depending on its state (selected/not selected), either the selected component or all the diagram components are shown.
- **Show Annotations** - Depending on its state (selected/not selected), the documentation nodes are shown or hidden.
- **Auto expand to references** - This option controls how the schema diagram is automatically expanded. For instance, if you select it and then edit a top-level element or you trigger a diagram refresh, the diagram will be expanded until it reaches the referred components. If this option is left unchecked, only the first level of the diagram is expanded, showing the top-level elements. For large schemas, the editor disables this option automatically.
- **Collapse Children** - Collapses the children of the selected view.
- **Expand Children** - Expands the children of the selected view.
- **Print Selection...** - Prints the selected view.
- **Save as Image...** - Saves the current selection as image, in JPEG, BMP, SVG or PNG format.
- **Refresh** - Refreshes the schema diagram according to the changes in your code (changes in your imported documents or those that are not reflected automatically in the compiled schema).

If the schema is not valid, you see only an error message in the **Logical Model View** instead of the diagram.

NVDL Outline View

The NVDL **Outline** view presents a list with the named or anonymous rules that appear in the diagram. It allows a quick access to a rule by name. It can be opened from the **Window > Show View > Other > oXygen > 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.

Searching and Refactoring Actions

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

- **NVDL >  Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this scope, a warning dialog is shown. A search scope includes the project or a collection of files and directories.
You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.
- **contextual menu of current editor > Search > Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the dialog above.

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

- **NVDL >  Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this scope, a warning dialog is shown. You have the possibility to define another search scope. A search scope includes the project or a collection of files and folders.
- **contextual menu of current editor > Search > Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files specified in the search scope.
- **NVDL > Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.
- **contextual menu of current editor > Rename Component...** - Allows you to rename the current component.

Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected NVDL named mode. You can open the view from **Window > Show View > Other > oXygen > 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.

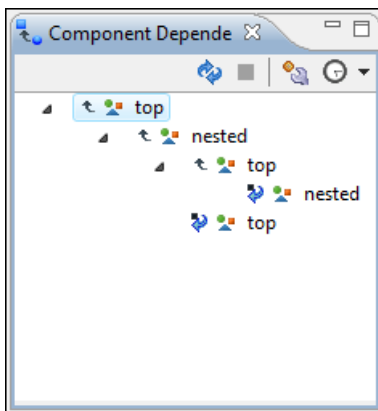








Figure 80: Component Dependencies View - Hierarchy for test.nvd1

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

-  - Refreshes the dependencies structure.
-  - Allows you to stop the dependencies computing.
-  - Allows you to configure a search scope to compute the dependencies structure. If you decide to set the application to use automatically the defined scope for future operations, select the corresponding checkbox.
-  - Repeats a previous dependencies computation.

The following actions are available in the contextual menu:

- **Go to First Reference** - Selects the first reference of the referred component from the current selected component in the dependencies tree.
- **Go to Component** - Shows the definition of the current selected component in the dependencies tree.

 **Tip:** If a component contains multiple references to another component, a small table containing all references is shown. When a recursive reference is encountered it is marked with a special icon .

Editing XSLT Stylesheets

This section explains the features of the XSLT editor.

Validating XSLT Stylesheets

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

Custom Validation of XSLT Stylesheets

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

There are two validators configured by default:

- **MSXML 4.0** - included in Oxygen XML 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

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

You can validate an XSLT document using the engine from transformation scenario or a custom validation scenario. If you choose to validate using the engine from transformation scenario, and a transformation scenario is not associated with the current document or the engine has no validation support, the default engine set in **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XSLT** is used. The list of reusable scenarios for documents of the same type as the current document is displayed in case you choose to use a custom validation scenario. For more details see [Validation Scenario](#).

Contextual Editing

Smaller interrelated modules that define a complex stylesheet cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, a function defined in a main stylesheet is not visible when you edit an included or imported module. Oxygen XML Developer plugin provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included files in the context of the larger stylesheet structure.

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

Syntax Highlight

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

Content Completion in XSLT Stylesheets

Inside XSLT templates of an XSLT stylesheet, the content completion assistant presents all the elements allowed in any context by the schema associated to the result of applying the stylesheet. That schema is *defined by the user in the [Content Completion / XSL preferences](#)* page and can be of type: XML Schema, DTD, RELAX NG schema, or NVDL schema. There are presented all the elements because in a template there is no context defined for the result document. The user is allowed to insert any element defined by the schema of the result document.

The content completion window lists the following item types defined in the current stylesheet and in the imported and included XSLT stylesheets:

- template modes
- template name
- variable names
- parameter names

The extension functions built in the Saxon transformation engine are presented in the content completion list only if the Saxon namespace (<http://saxon.sf.net> for XSLT version 2.0 or <http://icl.com/saxon> for XSLT version 1.0) is mapped to a prefix and one of the following conditions is true:

- the edited file has a transformation scenario that uses as transformation engine Saxon 6.5.5 (for XSLT version 1.0), Saxon 9.3.0.5 PE or Saxon 9.3.0.5 EE (for XSLT version 2.0);
- the edited file has a validation scenario that uses as validation engine Saxon 6.5.5 (for version 1.0), Saxon 9.3.0.5 PE or Saxon 9.3.0.5 EE (for version 2.0);

- the validation engine specified in [Options](#) page is Saxon 6.5.5 (for version 1.0), Saxon 9.3.0.5 PE or Saxon 9.3.0.5 EE (for version 2.0).

Namespace prefixes in the scope of the current context are presented at the top of the content completion window to speed up the insertion into the document of prefixed elements.

For the common namespaces like XSL namespace (<http://www.w3.org/1999/XSL/Transform>), XML Schema namespace (<http://www.w3.org/2001/XMLSchema>) or Saxon namespace (<http://icl.com/saxon> for version 1.0, <http://saxon.sf.net/> for version 2.0), Oxygen XML Developer plugin provides an easy mode to map them by proposing a prefix for these namespaces.

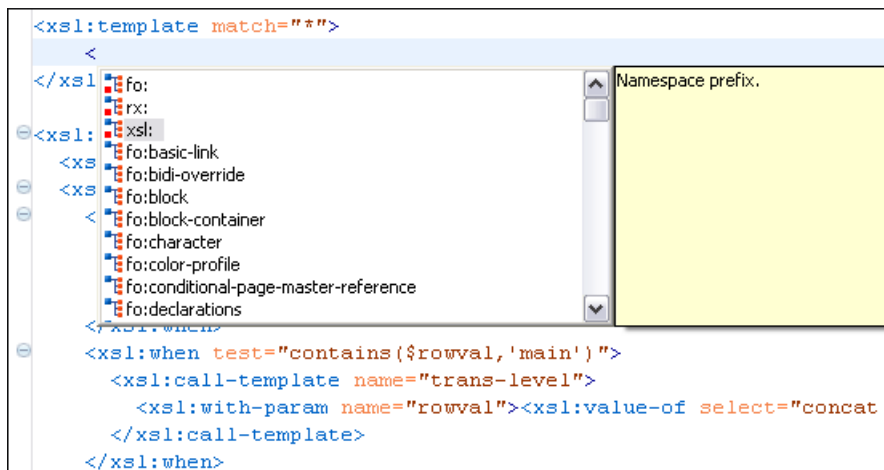


Figure 81: Namespace Prefixes in the Content Completion Window

Content Completion in XPath Expressions

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

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

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

Content completion for XPath expressions is started:

- on XPath operators detected in one of the `match`, `select` and `test` attributes of XSLT elements: `"`, `'`, `/`, `//`, `(`, `[`, `|`, `:`, `::`, `$`
- for attribute value templates of non-XSLT elements, that is the `{` character when detected as the first character of the attribute value
- on request, if the combination CTRL + Space is pressed inside an edited XPath expression.

The items presented in the content completion window are dependent on:

- the context of the current XSLT element;
- the XML document associated with the edited stylesheet in the stylesheet transformation scenario;
- the XSLT version of the stylesheet (1.0 or 2.0).

For example, if the document associated with the edited stylesheet is:

```
<personnel>
  <person id="Big.Boss">
```

```

    <name>
      <family>Boss</family>
      <given>Big</given>
    </name>
    <email>chief@oxygenxml.com</email>
    <link subordinates="one.worker" />
  </person>
  <person id="one.worker">
    <name>
      <family>Worker</family>
      <given>One</given>
    </name>
    <email>one@oxygenxml.com</email>
    <link manager="Big.Boss" />
  </person>
</personnel>

```

and you enter an `xsl:template` element using the content completion assistant, the following actions are triggered:

- the `match` attribute is inserted automatically;
- the cursor is placed between the quotes;
- the XPath content completion assistant automatically displays a popup window with all the XSLT axes, XPath functions and elements and attributes from the XML input document that can be inserted in the current context.

The set of XPath functions depends on the XSLT version declared in the root element `xsl:stylesheet`: 1.0 or 2.0.

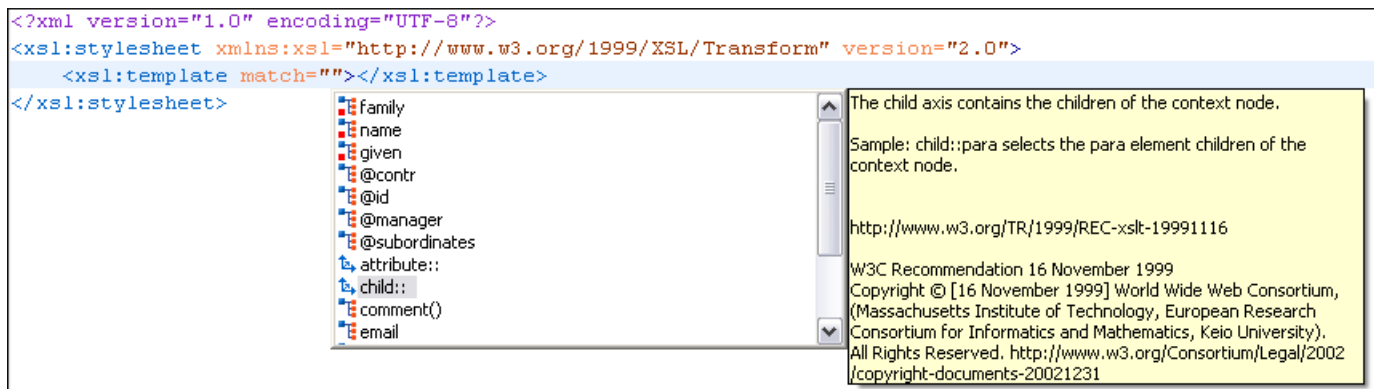


Figure 82: Content Completion in the `match` Attribute

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:

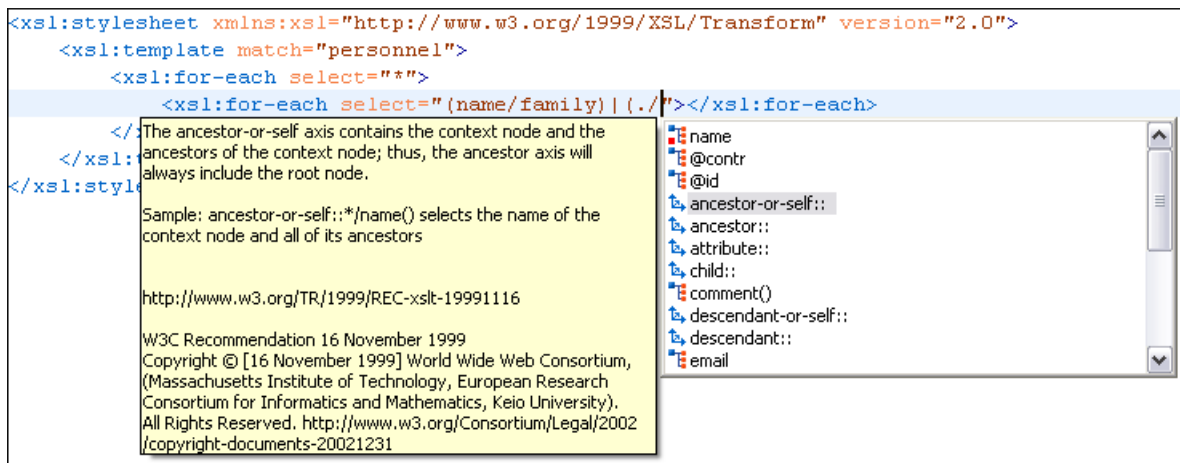


Figure 83: Content Completion in the `select` Attribute

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

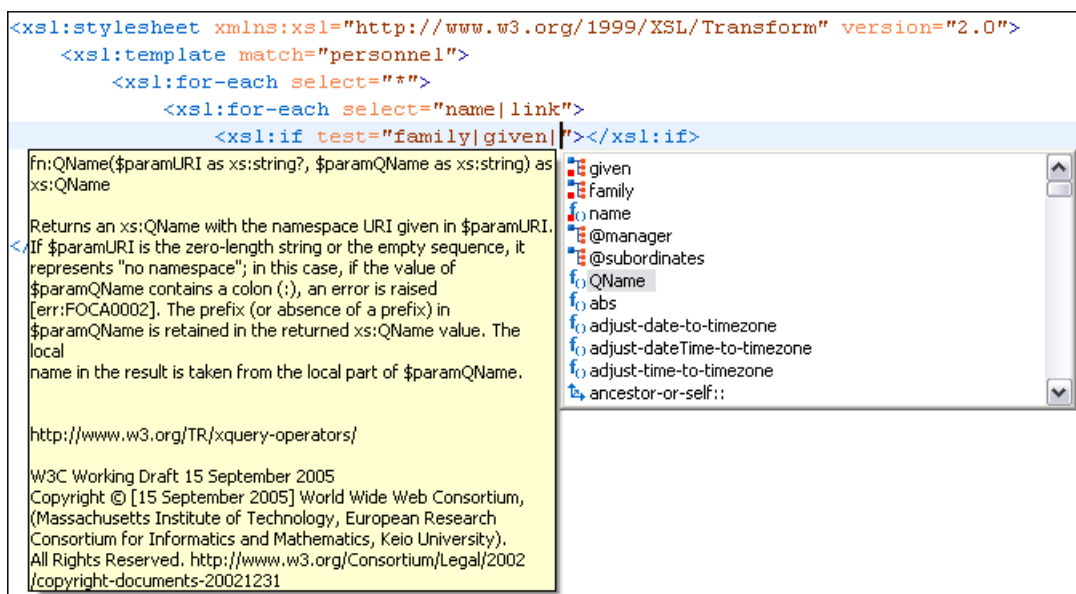


Figure 84: Content Completion in the `test` Attribute

XSLT variable references are easier to insert in XPath expressions with the help of the content completion popup triggered by the `$` character which signals the start of such a reference in an XPath expression.



Figure 85: Content Completion in the `test` Attribute

If the { character is the first one in the value of the attribute, the same content completion assistant is available also in attribute value templates of non-XSLT elements.

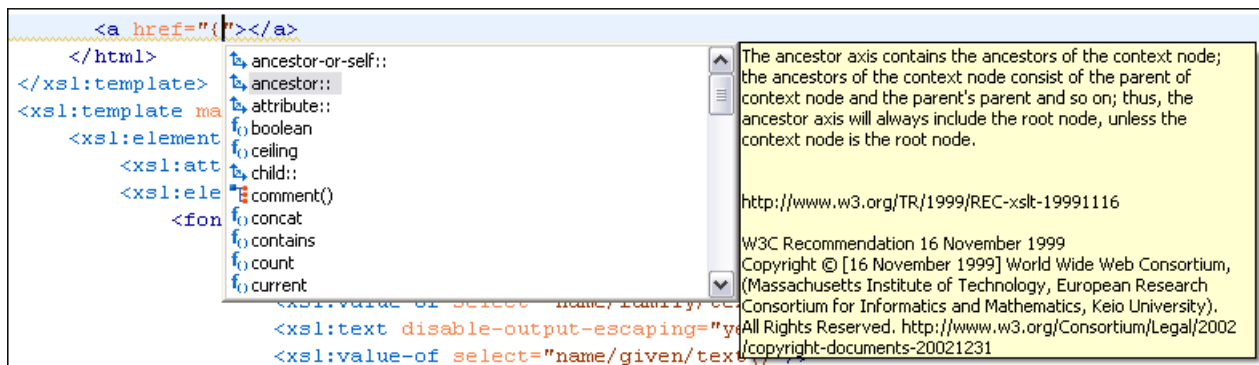


Figure 86: Content Completion in Attribute Value Templates

The time delay *configured in Preferences* page for all content completion windows is applied also for the XPath expressions content completion window.

Tooltip Helper for the XPath Functions Arguments

When editing the arguments of an XPath/XSLT function, Oxygen XML 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 *v1* and *v2*.

```
<xsl:template match="/">
  <xsl:value-of select="concat(abs($v1),
abs($v2))" ></xsl:value-of>
</xsl:template>
```

When moving the caret before the first `abs` function, Oxygen XML Developer plugin identifies it as the first argument of the `concat` function. The tooltip shows in bold font the following information about the first argument:

- its name is `$arg1`;
- its type is `xdt:anyAtomicType`;
- it is optional (note the ? sign after the argument type).

The function takes also other arguments, having the same type, and returns a `xs:string`.

```
name: concat($arg1 as xdt:anyAtomicType?, $arg2 as xdt:anyAtomicType?, ...) as xs:string
match:
select="concat(abs($v1), abs($v2))" ></xsl:value-of>
</xsl:template>
stylesheet
```

Figure 87: XPath Tooltip Helper - Identify the `concat` Function's First Argument

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

The screenshot shows an XSLT code editor with a tooltip. The code is:


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

 The tooltip points to the expression `abs($arg as numeric?)` and displays its argument `$arg as numeric?` in bold font.

Figure 88: XPath Tooltip Helper - Identify the `abs` Function's Argument

Further, clicking the second `abs` function name, the editor detects that it represents the second argument of the `concat` function. The tooltip is repainted to display the second argument in bold font.

The screenshot shows the same XSLT code editor. The tooltip now points to the second `abs` function call `abs($v2)` and displays its argument `$v2` in bold font.

Figure 89: XPath Tooltip Helper - Identify the `concat` Function's Second Argument

The tooltip helper is available also in the XPath toolbar and the **XPath Builder** view.

Code Templates

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

The XSL code template called **Template-Match-Mode**

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

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

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

The XSLT/XQuery Input View

The structure of the XML document associated to the edited XSLT stylesheet, or the structure of the source documents of the edited XQuery is displayed in a tree form in a view called **XSLT/XQuery Input**. The tree nodes represent the elements of the documents.

The XSLT Input View

If you click a node, the corresponding template from the stylesheet is highlighted. A node can be dragged from this view and dropped in the editor area for quickly inserting `xsl:template`, `xsl:for-each`, or other XSLT elements that have the `match/select/test` attribute already completed. The value of the attribute is the correct XPath expression referring to the dragged tree node. This value is based on the current editing context of the drop spot.

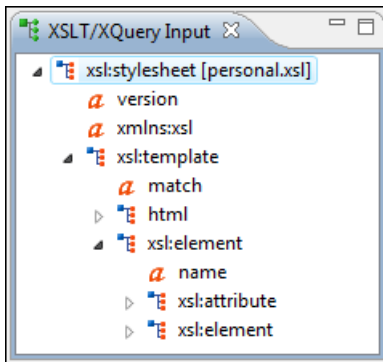


Figure 90: XSLT Input View

For example, for the following XML document:

```
<personnel>
  <person id="Big.Boss">
    <name>
      <family>Boss</family>
      <given>Big</given>
    </name>
    <email>chief@oxygenxml.com</email>
    <link subordinatates="one.worker"/>
  </person>
  <person id="one.worker">
    <name>
      <family>Worker</family>
      <given>One</given>
    </name>
    <email>one@oxygenxml.com</email>
    <link manager="Big.Boss"/>
  </person>
</personnel>
```

and the following XSLT stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">
  <xsl:template match="personnel">
    <xsl:for-each select="*">
      </xsl:for-each>
    </xsl:template>
  </xsl:stylesheet>
```

if you drag the given element and drop it inside the xsl:for-each element, the following popup menu is displayed:



Figure 91: XSLT Input Drag and Drop Popup Menu

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

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

Figure 92: XSLT Input Drag and Drop Result

The XSLT Outline View

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

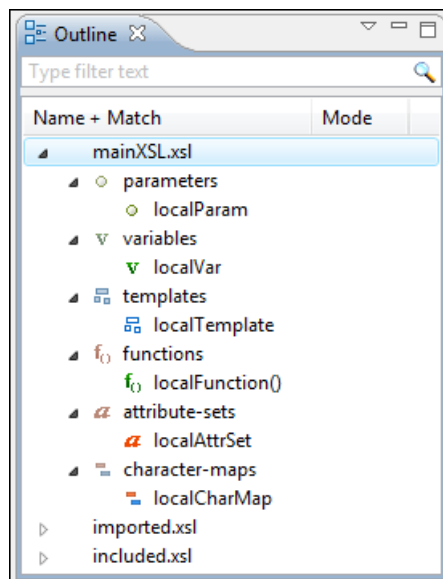












Figure 93: The XSLT Outline View

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's moves or the changes in the XSLT editor. Selecting one of the components from the outline view also selects the corresponding item in the source document.
- **Show XML structure** - Displays the document's XML structure in a tree-like structure.

-  **Sort** - Alphabetically sorts the stylesheet components.
- **Show all components** - Displays all components that were collected starting from the main file. This option is set by default.
- **Show local components** - Displays components defined in the current file only.
- **Group by location/type/mode** - The stylesheet components can be grouped by location, type, and mode.
-  **Show components** - Shows the define patterns collected from the current document.
-  **Flat presentation mode of the filtered results** - when active, the application flattens the filtered result elements to a single level.
-  **Show comments and processing instructions** - show/hide comments and processing instructions in the **Outline** view.
- **T Show text** - show/hide additional text content for the displayed elements.
-  **Show attributes** - show/hide attribute values for the displayed elements. The displayed attribute values can be changed from [the Outline preferences panel](#).
-  **Configure displayed attributes** - displays the [XML Structured Outline preferences page](#).


The following contextual menu actions are available:

- **Append Child** - Displays a list of elements that can be inserted as children of the current element.
- **Insert Before** - Displays a list of elements that can be inserted as siblings of the current element, before the current element.
- **Insert After** - Displays a list of elements that can be inserted as siblings of the current element, after the current element.
-  **Toggle Comment** - Comments/uncomments the currently selected element.
- **Remove (Delete)** - Removes the selected item from the stylesheet.
-  **Search References (Ctrl+Shift+R)** - Searches all references of the item found at current cursor position in the defined scope, if any. See [Finding XSLT References and Declarations](#) for more details.
- **Search References in...** - Searches all references of the item found at current cursor position in the specified scope. See [Finding XSLT References and Declarations](#) for more details.
-  **Component Dependencies** - Allows you to see the dependencies for the current selected component. See [Component Dependencies View](#) for more details.
-  **Rename Component** - Renames the selected component. See [XSLT Refactoring Actions](#) for more details.

The stylesheet components information is presented on two columns: the first column presents the name and match attributes, the second column the mode attribute. If you know the component name, match or mode, you can search it in the **Outline** view by typing one of these pieces of information in the filter text field from the 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 is synchronized with **Text** view; when you click a component in the **Outline** view, its definition is highlighted in the **Text** view.

XSLT Stylesheet Documentation Support

Oxygen 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 referred to as user-defined data elements. Such elements can contain the documentation for the stylesheet and its elements (top-level elements whose names are in the XSLT namespace). Oxygen XML Developer plugin offers its own XML schema that defines such documentation elements. The schema is named `stylesheet_documentation.xsd` and can be found in `[oxygen-install-folder]/frameworks/stylesheet_documentation`. The user can also specify a custom schema in *XSL Content Completion options*.

When content completion is invoked inside an XSLT editor by pressing **(CTRL+Space)**, it offers elements from the XSLT documentation schema (either the built-in one or one specified by user).

In **Text** mode, to add documentation blocks while editing use the **Add component documentation** action available in the contextual menu, **Source** submenu.

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>
    <xd:p>Search inside parameter <xd:i>string</xd:i> for the last
    occurrence of parameter
    <xd:i>searched</xd:i>. The substring starting from the 0
    position to the identified last
    occurrence will be returned. <xd:ref
    name="f:substring-after-last" type="function"
    xmlns:f="http://www.oxygenxml.com/doc/xsl/functions">See
    also</xd:ref></xd:p>
  </xd:desc>
  <xd:param name="string">
    <xd:p>String to be analyzed</xd:p>
  </xd:param>
  <xd:param name="searched">
    <xd:p>Marker string. Its last occurrence will be
    identified</xd:p>
  </xd:param>
  <xd:return>
    <xd:p>A substring starting from the beginning of
    <xd:i>string</xd:i> to the last
    occurrence of <xd:i>searched</xd:i>. If no occurrence is found
    an empty string will be
    returned.</xd:p>
  </xd:return>
</xd:doc>
```

Generating Documentation for an XSLT Stylesheet

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

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

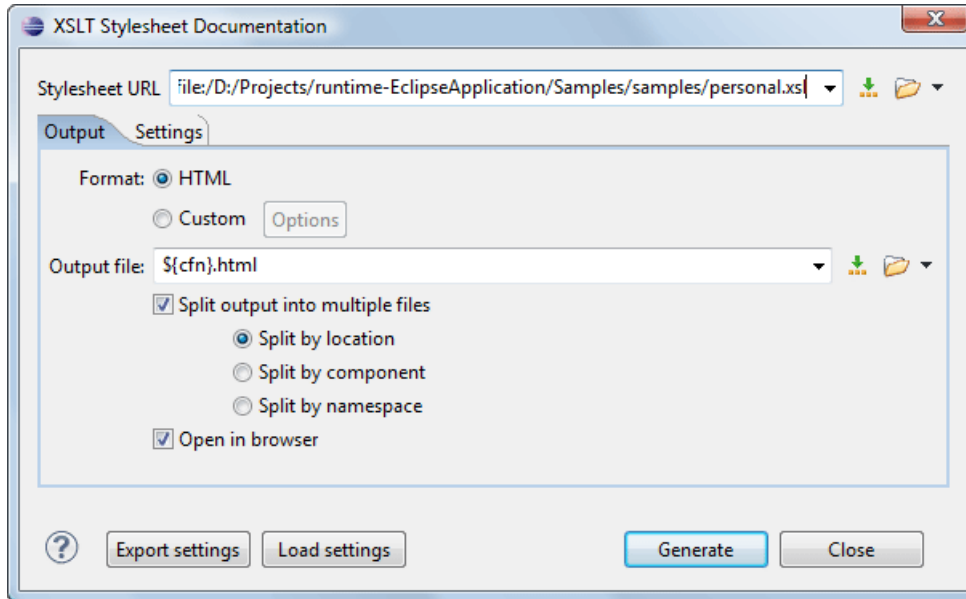


Figure 94: The Output Panel of the XSLT Stylesheet Documentation Dialog

The **XSL URL** field of the dialog panel must contain the full path to the XSL Stylesheet file you want to generate documentation for. The stylesheet can be either a local or a remote one. You can also specify the path of the stylesheet using editor variables.

You can choose to split the output into multiple files using different split criteria. For large XSLT stylesheets being documented, choosing a different split criterion may generate smaller output files providing a faster documentation browsing.

The available split criteria are:

- by location - each output file contains the XSLT elements from the same stylesheet;
- by namespace - each output file contains information about elements with the same namespace;
- by component - each output file contains information about one stylesheet XSLT element.

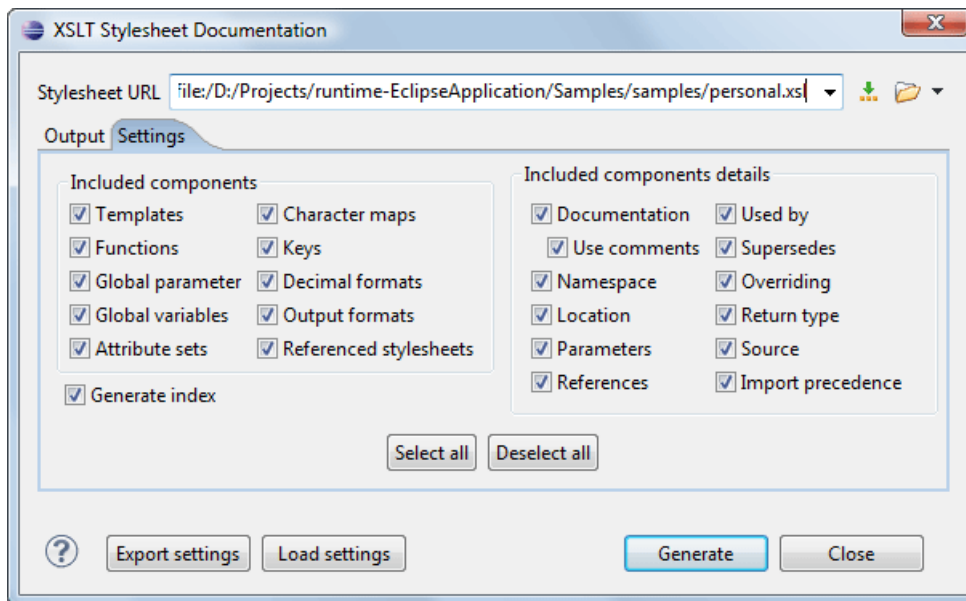


Figure 95: The Settings Panel of the XSLT Stylesheet Documentation Dialog

When you generate documentation for an XSLT stylesheet you can choose what XSLT elements to include in the output (templates, functions, global parameters, global variables, attribute sets, character maps, keys, decimal formats, output formats, XSLT elements from referenced stylesheets) and the details to include in the documentation:

- **Documentation** - Shows the documentation for each XSLT element. For HTML format, the user-defined data elements that are recognized and transformed in documentation blocks of the XSLT elements they precede, are the ones from the following schemas:
 - Oxygen XML 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 referred from within an element.
- **Used by** - Shows the list of all the XSLT elements that refer the current named element.
- **Supersedes** - Shows the list of all the XSLT elements that are superseded the current element.

- **Overriding** - Shows the list of all the XSLT elements that override the current element.
- **Return type** - Shows the return type of the function.
- **Source** - Shows the text stylesheet source for each XSLT element.
- **Import precedence** - Shows the computed import precedence as declared in XSL transformation specifications.
- **Generate index** - Creates an index with all the XSLT elements included in the documentation.
- **Load settings / Export settings** - The current settings can be saved for further usage (for example for generating documentation from command-line interface) with the **Export settings** button, and reloaded when necessary with the **Load settings** button.

Generate Documentation in HTML Format

The generated documentation looks like:

The screenshot displays a web-based interface for XSLT Stylesheet Documentation. It features a 'Table of Contents' on the left with a 'Group by' dropdown set to 'Location'. Below it are sections for 'Templates' and 'Imported modules'. The main content area is divided into two panels. The top panel, titled 'Main stylesheet xslDocHtml.xsl', shows a 'Description' section explaining the stylesheet's purpose and a list of 'Imported modules'. The bottom panel, titled 'Template createJsIdsArray', shows a 'Description' section explaining the template's function, a 'Parameters' section listing 'arrayName' and 'nodes', and a 'References' section listing 'getDivId(\$node as item())'. A 'Showing:' control panel on the right allows filtering the displayed content, with options for Documentation, Parameters, Used by, References, Imported modules, and Source, all of which are checked.

Figure 96: XSLT Stylesheet Documentation Example

The generated documentation includes the following:

- **Table of Contents** - You can group the contents by namespace, location, or component type. The XSLT elements from each group are sorted alphabetically (named templates are presented first and the match ones second).
- **Information about main, imported, and included stylesheets** - This information consists of:
 - XSLT modules included or imported by the current stylesheet;
 - the XSLT stylesheets where the current stylesheet is imported or included
 - and the stylesheet location.



Figure 97: 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.

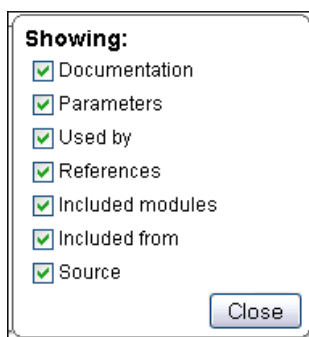


Figure 98: The Showing View

For each element included in the documentation, the section presents the element type followed by the element name (value of the name or match attribute for match templates).

Function func:substring-before-last

Documentation	<p>Description</p> <p>Get the substring before the last occurrence of the given substring</p> <p>Parameters</p> <p>string The string in which to search</p> <p>searched The string to search</p> <p>Return</p> <p>The substring starting from the start of the string to the index of the last occurrence of searched</p>							
Namespace	http://www.oxygenxml.com/doc/xsl/functions							
Type	xs:string							
Used by	<table style="width: 100%; border-collapse: collapse;"> <tr><td>Template</td><td>Nindex</td></tr> <tr><td>Function</td><td>func:substring-before-last(\$string as item(), \$searched as item())</td></tr> <tr><td>Variable</td><td>indexFile</td></tr> </table>	Template	Nindex	Function	func:substring-before-last(\$string as item(), \$searched as item())	Variable	indexFile	
Template	Nindex							
Function	func:substring-before-last(\$string as item(), \$searched as item())							
Variable	indexFile							
References	<table style="width: 100%; border-collapse: collapse;"> <tr><td>Function</td><td>substring-before-last(\$string as item(), \$searched as item())</td></tr> </table>		Function	substring-before-last(\$string as item(), \$searched as item())				
Function	substring-before-last(\$string as item(), \$searched as item())							
Parameters	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #e0f0e0;"><th>QName</th><th>Namespace</th></tr> </thead> <tbody> <tr><td>searched</td><td>No namespace</td></tr> <tr><td>string</td><td>No namespace</td></tr> </tbody> </table>		QName	Namespace	searched	No namespace	string	No namespace
QName	Namespace							
searched	No namespace							
string	No namespace							
Import precedence	7							
Source	<pre style="font-family: monospace; font-size: 0.9em;"> <xsl:function as="xs:string" name="func:substring-before-last"> <xsl:param name="string"/> <xsl:param name="searched"/> <xsl:variable name="toReturn"> <xsl:choose> <xsl:when test="contains(\$string, \$searched)"> <xsl:variable name="before" select="substring-before(\$string, \$searched)"/> <xsl:variable name="rec" select="func:substring-before-last(substring-after <xsl:choose> </pre>							

Figure 99: Documentation for an XSLT Element

Generate Documentation in a Custom Format

XSLT stylesheet documentation can be also generated in a custom format. You can choose the format from the [XSLT Stylesheet Documentation](#) dialog. Specify your own stylesheet to transform the intermediary XML generated in the documentation process. You must write your stylesheet based on the schema `xslDocSchema.xsd` from `[Oxygen-install-folder]/frameworks/stylesheet_documentation`. You can create a custom format starting from one of the stylesheets used in the predefined HTML, PDF, and DocBook formats. These stylesheets are available in `[Oxygen-install-folder]/frameworks/stylesheet_documentation/xsl`.

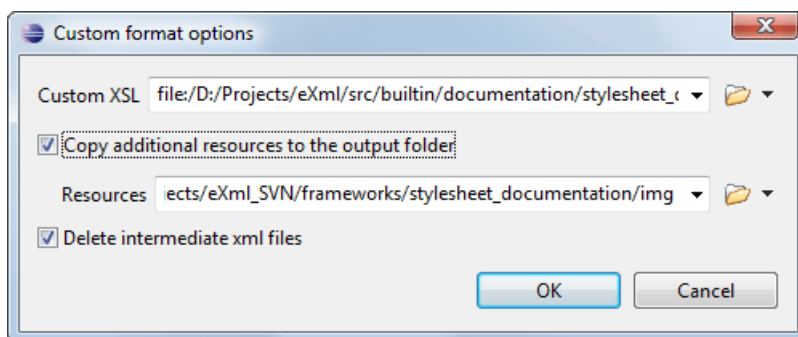


Figure 100: The Custom Format Options Dialog

When using a custom format, you can also copy additional resources into the output folder or choose to keep the intermediate XML files created during the documentation process.

Generating Documentation From the Command Line Interface

You can export the settings of the **XSLT Stylesheet Documentation** dialog to an XML file by pressing the **Export settings** button. With the exported settings file, you can generate the same documentation from the command line by running the script `stylesheetDocumentation.bat` (on Windows) / `stylesheetDocumentation.sh` (on Mac OS X / Unix / Linux) located in the Oxygen XML 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">
          <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>
      </xsdDocumentationOptions>
    </entry>
  </map>
</serialized>
```



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



```

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

```

Finding XSLT References and Declarations

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

- **XSL** >  > **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined but the current edited resource is not part of the range of determined resources, a warning dialog is shown. This dialog allows you to define another search scope.
- **contextual menu of current editor** > **Search** > **Search References in...** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the dialog above.
- **XSL** >  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope, if any. If a scope is defined but the current edited resource is not part of the range of resources determined by this scope, a warning dialog is shown.
- **contextual menu of current editor** > **Search** > **Search Declarations in...** - Searches all declarations of the item found at current cursor position in the file or files that you specify when define a new scope.
- **XSL** > **Search Occurrences in File** - Searches all occurrences of the item at the caret position in the currently edited file.
- **XSL** > **Show Definition** - Moves the cursor to the location of the definition of the current item.


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. Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.


This feature is set on automatic search by default and can be configured in the **Options > Preferences > Editor > Mark Occurrences** page. A search can also be triggered with the **Search > Search Occurrences in File (Ctrl+Shift+U)** contextual menu action. Matches are displayed in separate tabs of the **Results** view.


XSLT Refactoring Actions

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

- **XSL** >  > **Create Template from Selection...** - Opens a dialog that allows the user to specify the name of the new template to be created. The possible changes to perform on the document can be previewed before altering the document. After pressing OK the template is created and the selection is replaced with a `<xsl:call-template>` instruction referring the newly created template.

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

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

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

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

```
<person id="Big{test}Boss" />
```

you obtain:

```
<person>
  <xsl:attribute name="id">
    <xsl:text>Big</xsl:text>
    <xsl:value-of select="test" />
    <xsl:text>Boss</xsl:text>
  </xsl:attribute>
</person>
```

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

Resource Hierarchy/Dependencies View

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

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

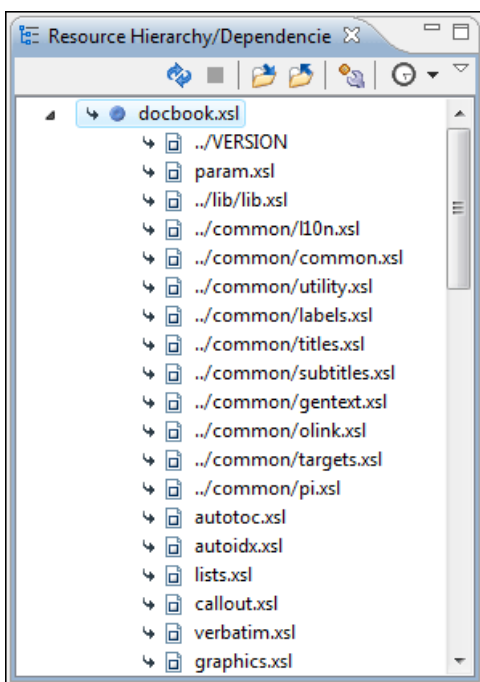


Figure 101: Resource Hierarchy/Dependencies View - Hierarchy for docbook.xsl

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

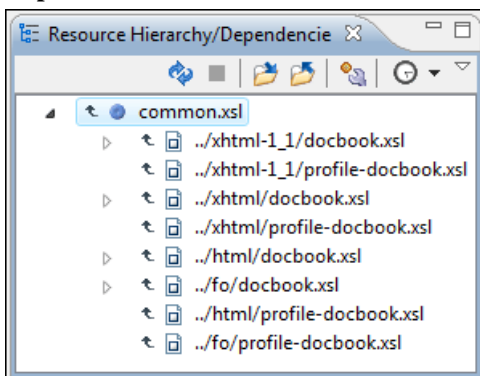








Figure 102: Resource Hierarchy/Dependencies View - Dependencies for common.xsl

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

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

The following actions are available in the contextual menu:

- **Open** - Opens the stylesheet. Alternatively, you can open the stylesheet by a double-click on the Hierarchy/Dependencies structure.
- **Copy location** - Copies the location of the stylesheet.
- **Show Resource Hierarchy** - Shows the hierarchy for the selected stylesheet.

- **Show Resource Dependencies** - Shows the dependencies for the selected stylesheet.
- **Expand All** - Expands all the children of the selected stylesheet from the Hierarchy/Dependencies structure.
- **Collapse All** - Collapses all the children of the selected stylesheet from the Hierarchy/Dependencies structure.

Component Dependencies View

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

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

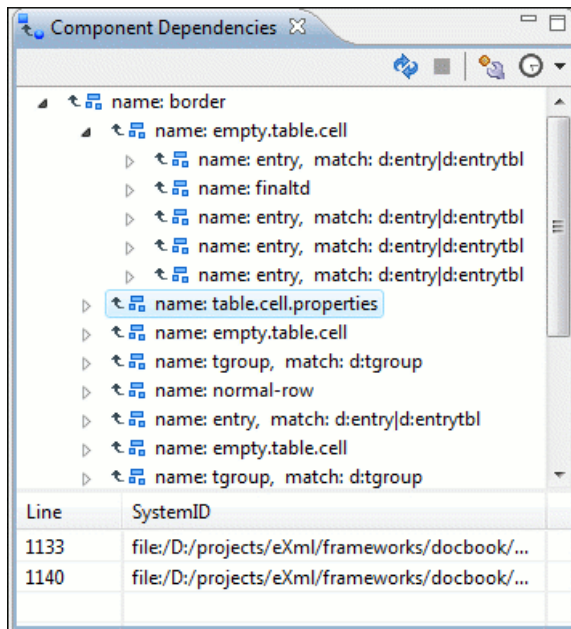






Figure 103: Component Dependencies View - Hierarchy for table.xsl

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

-  - Refreshes the dependencies structure.
-  - Stops the dependencies computing.
-  - Allows you to configure a search scope to compute the dependencies structure. You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.
-  - Allows you to repeat a previous dependencies computation.

The following actions are available on the contextual menu:

- **Go to First Reference** - Selects the first reference of the referred component from the current selected component in the dependencies tree.
- **Go to Component** - Shows the definition of the current selected component in the dependencies tree.

Tip:

If a component contains multiple references to another, a small table is shown containing all references.

When a recursive reference is encountered, it is marked with a special icon .

XSLT Quick Assist Support

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

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

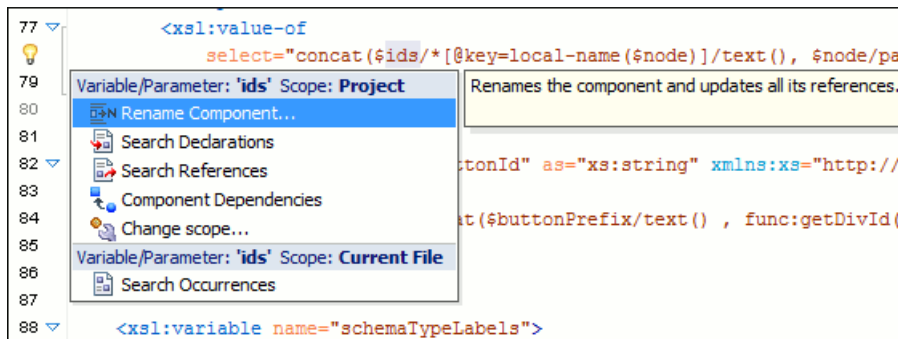


Figure 104: XSLT Quick Assist Support

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

- **Rename Component** - Renames the component and all its dependencies;
- **Search Declaration** - Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference;
- **Search References** - Searches all references of the component in a predefined scope;
- **Component Dependencies** - Searches the component dependencies in a predefined scope;
- **Change Scope** - Configures the scope that will be used for future search or refactor operations;
- **Search Occurrences** - Searches all occurrences of the file within the current scope.

Editing XQuery Documents

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

XQuery Outline View

The XQuery document structure is presented in the **XQuery Outline** view. The outline tree presents the list of all the components (namespaces, imports, variables, and functions) from both the edited XQuery file and its imports. It allows quick access to a component by knowing its name. It can be opened from the **Window > Show View > Other > oXygen > Outline** menu action.

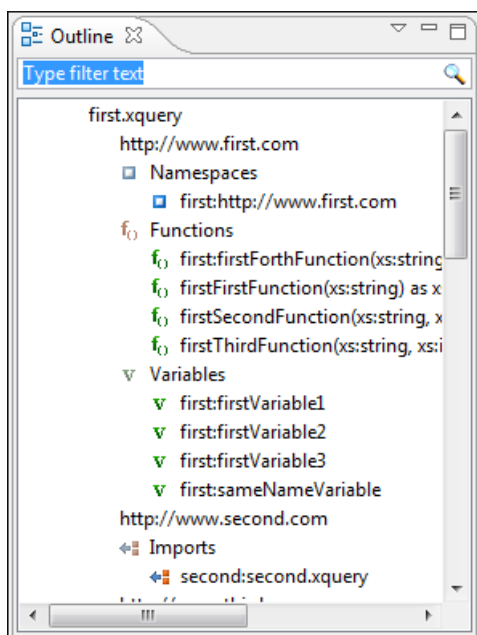


Figure 105: XQuery Outline View

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 local components** - Displays components defined in the current file only.
- **Group by location/namespace/type** - Allows you to group the components by location, namespace, and type. When grouping by namespace, the main XQuery module namespace is presented first in the **Outline** view.

If you know the component name, you can search it in the **Outline** view by typing its name in the filter text field from the top of the view or directly on the tree structure. When you type the component name in the filter text field you can switch to the tree structure using the arrow keys of the keyboard, **(Enter)**, **(Tab)**, **(Shift-Tab)**. To switch from tree structure to the filter text field, you can use **(Tab)**, **(Shift-Tab)**.

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

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

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

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

Folding in XQuery Documents

In a large XQuery document, the instructions enclosed in the '{' and '}' characters can be collapsed so that only the needed instructions remain in focus. The same *folding features available for XML documents* are also available in XQuery documents.

```

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

```

Figure 106: Folding in XQuery Documents

There is available the action **Go to Matching Bracket** **Ctrl+Shift+G** on contextual menu of XQuery editor for going to matching character when cursor is located at '{' character or '}' character. It helps for finding quickly matching character of current folding element.

Generating HTML Documentation for an XQuery Document

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

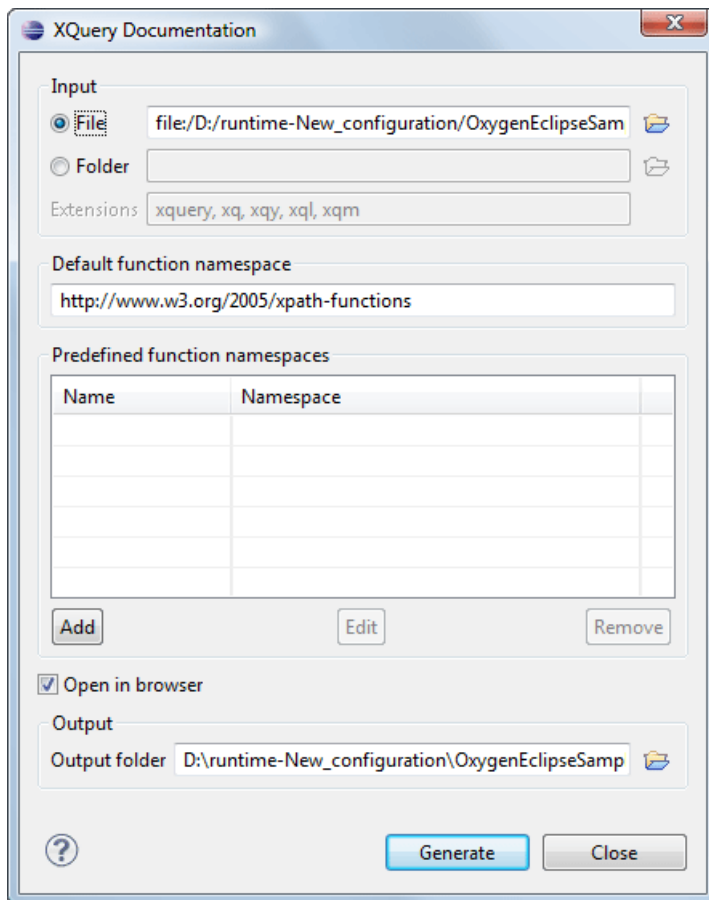


Figure 107: The XQuery Documentation Dialog

- **Input** - The **Input** panel allows the user to specify either the **File** or the **Folder** which contains the files for which to generate the documentation. One of the two text fields of the **Input** panel must contain the full path to the XQuery file. Extensions for the XQuery files contained in the specified directory can be added as comma-separated values. Default there are offered xquery, xq, xqy.
- **Default function namespace** - Optional URI for the default namespace for the submitted XQuery, only if it exists.
- **Predefined function namespaces** - Optional engine dependent, predefined namespaces that the submitted XQuery refers to. They allow the conversion to generate annotation information to support the presentation component hypertext linking, only if the predefined modules have been loaded into the local xqDoc XML repository.
- **Open in Browser/System Application** - Select this option if you want the result to be opened in the system application associated with that file type.
 - 👉 **Note:** If you already set the **Default Internet browser** option in the **Global** preferences page, it will take precedence over the default system application settings.
- **Output** - Allows the user to specify where the generated documentation is saved on disk.

Editing CSS Stylesheets

This section explains the features of the editor for CSS stylesheets and how these features should be used.

Validating CSS Stylesheets

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

The CSS properties accepted by the validator are the ones included in the current CSS profile that is selected in *the CSS validation preferences*. The profile **CSS 3 with Oxygen extensions** includes all the CSS 3 standard properties and the CSS extensions specific for Oxygen that can be used in Author mode. That means all Oxygen specific extensions are accepted in a CSS stylesheet by *the built-in CSS validator* when this profile is selected.

You can add custom CSS properties with a file called `customProperties.xml` located in the folder `[Oxygen-install-folder]/endorsed/builtin/css-validator`. The custom properties and their values are accepted by the CSS validator and are listed in the content completion window when editing a CSS stylesheet. For example the custom property called **custom** with the possible values **customValue1** and **customValue2** is specified with the following configuration file `customProperties.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<css_keywords>
  <property name="custom">
    <summary>Description for custom property.</summary>
    <value name="customValue1"/>
    <value name="customValue2"/>
  </property>
</css_keywords>
```

Content Completion in CSS Stylesheets

A content completion assistant like *the one available for XML documents* offers the CSS properties and the values available for each property. It is activated on the **(CTRL - Space)** shortcut and it is context-sensitive when invoked for the value of a property.

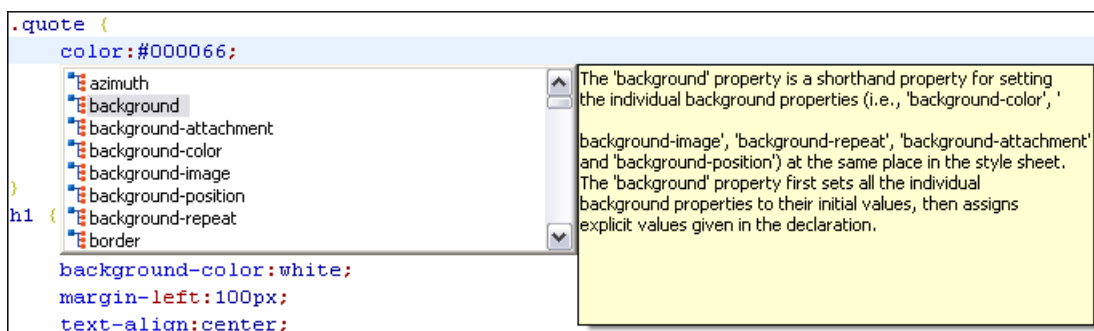


Figure 108: Content Completion in CSS Stylesheets

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

You can add custom CSS properties with a file called `customProperties.xml` located in the folder `[Oxygen-install-folder]/endorsed/builtin/css-validator`. The custom properties and their values are accepted by the CSS validator and are listed in the content completion window when editing a CSS stylesheet. For example the custom property called **custom** with the possible values **customValue1** and **customValue2** is specified with the following configuration file `customProperties.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<css_keywords>
  <property name="custom">
    <summary>Description for custom property.</summary>
    <value name="customValue1"/>
    <value name="customValue2"/>
  </property>
</css_keywords>
```

CSS Outline View

The CSS **Outline** view presents the import declarations for other CSS stylesheet files and all the selectors defined in the current CSS document. The selector entries can be presented as follows:

- in the order they appear in the document;
- sorted by element name used in the selector;
- sorted by the entire selector string representation.

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

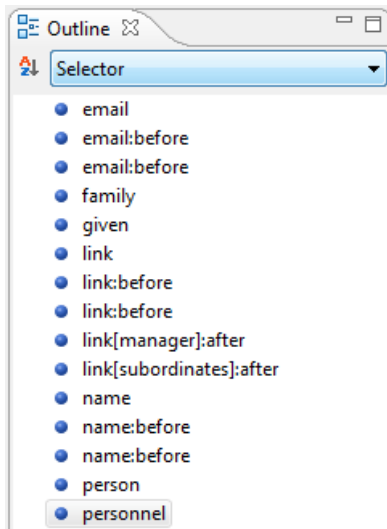


Figure 109: CSS Outline View

The selectors presented in this view can be quickly found using the key search field. When you press a sequence of character keys while the focus is in the view, the first selector that starts with that sequence is selected automatically.

Folding in CSS Stylesheets

In a large CSS stylesheet document, some styles can be collapsed so that only the needed styles remain in focus. The same *folding features available for XML documents* are also available in CSS stylesheets.

Formatting and Indenting CSS Stylesheets (Pretty Print)

If the edited CSS stylesheet becomes unreadable because of the bad alignment of the text lines, *the pretty-print operation available for XML documents* is also available for CSS stylesheets. It works in the same way as for XML documents and is available as the same menu and toolbar action.

Other CSS Editing Actions

The CSS editor type offers a reduced version of *the popup menu available in the XML editor*. Only *the 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 JSON Documents

This section explains the features of the Oxygen XML JSON Editor and how they should be used.

JSON Editor Text Mode

The **Text Mode** of the JSON editor provides the usual actions specific for a plain text editor: undo / redo, copy / paste, find / replace, drag and drop, and other editor actions like *validation* and *formatting and indenting (pretty print) document*.

You can use the two **Text** and **Grid** buttons available at the bottom of the editor panel if you want to switch between the editor **Text Mode** and **Grid Mode**.

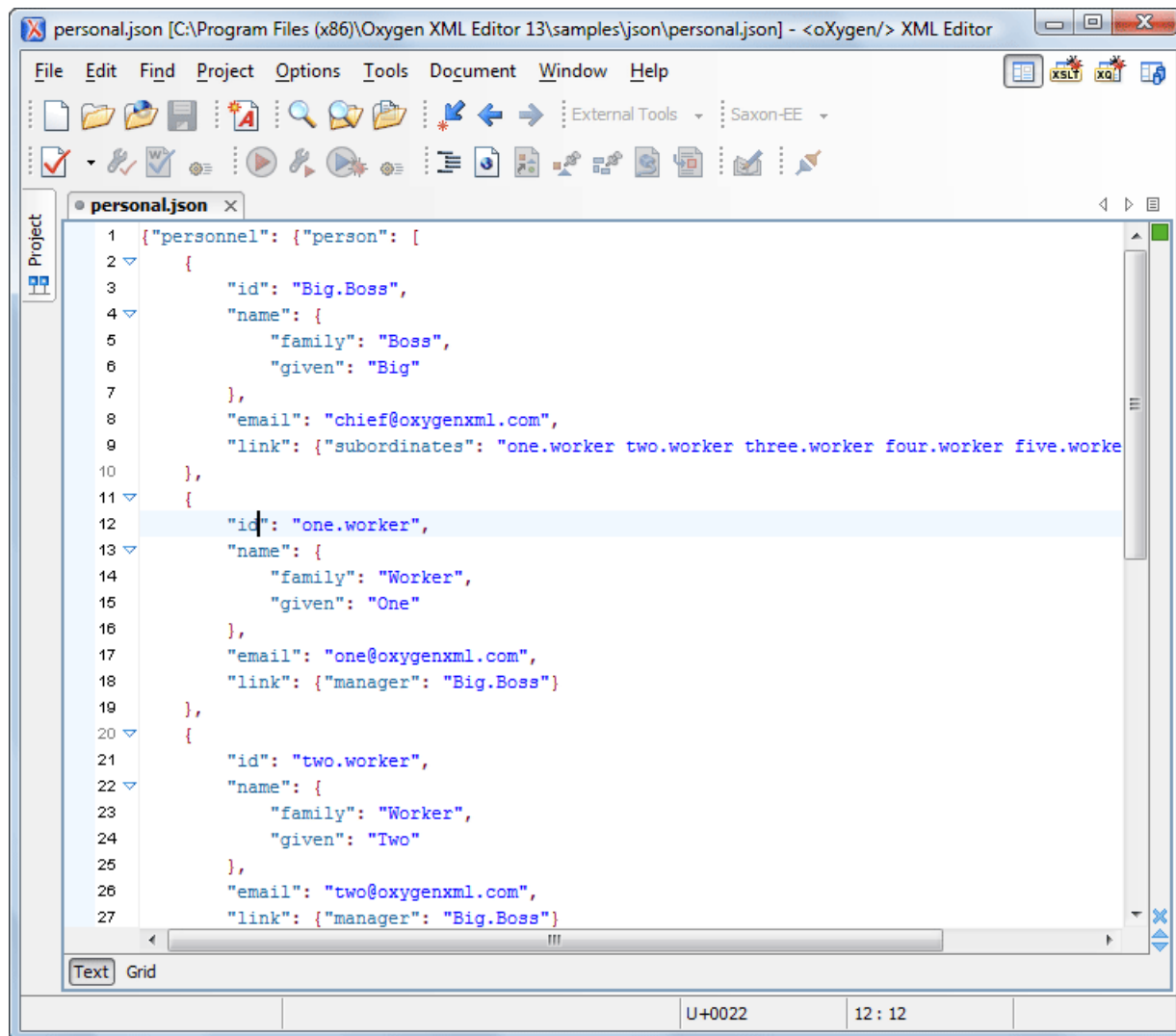


Figure 110: JSON Editor Text Mode

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.

JSON Editor Grid Mode

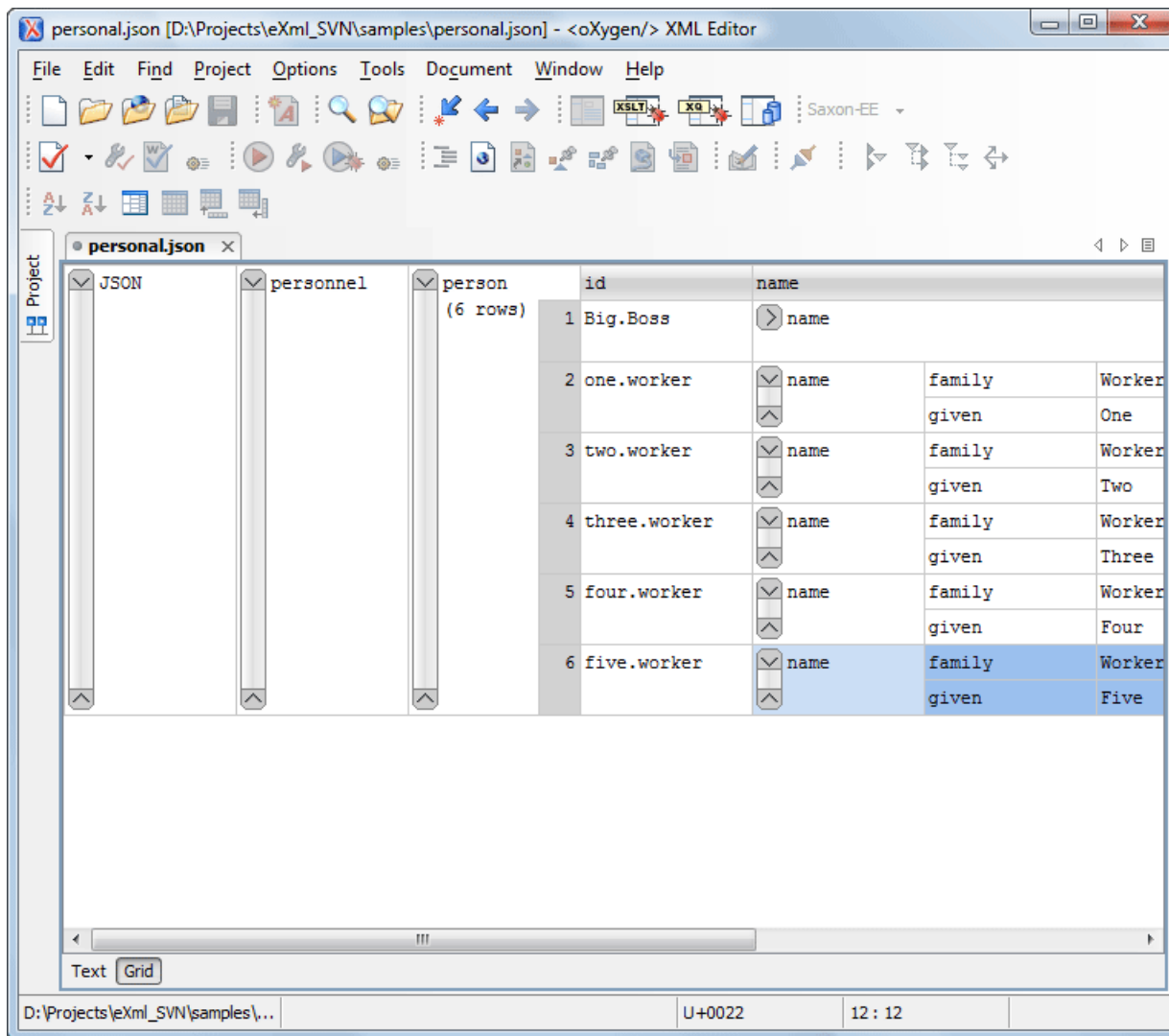


Figure 111: JSON Editor Grid Mode

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.

Validating JSON Documents

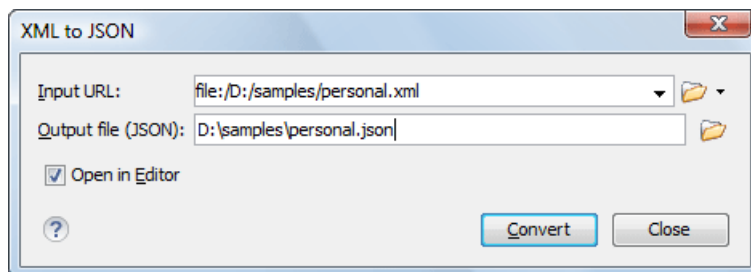
Oxygen XML Developer plugin includes a built-in JSON validator (based on the free JAVA source code available on www.json.org), integrated with the general validation support.

Convert XML to JSON

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

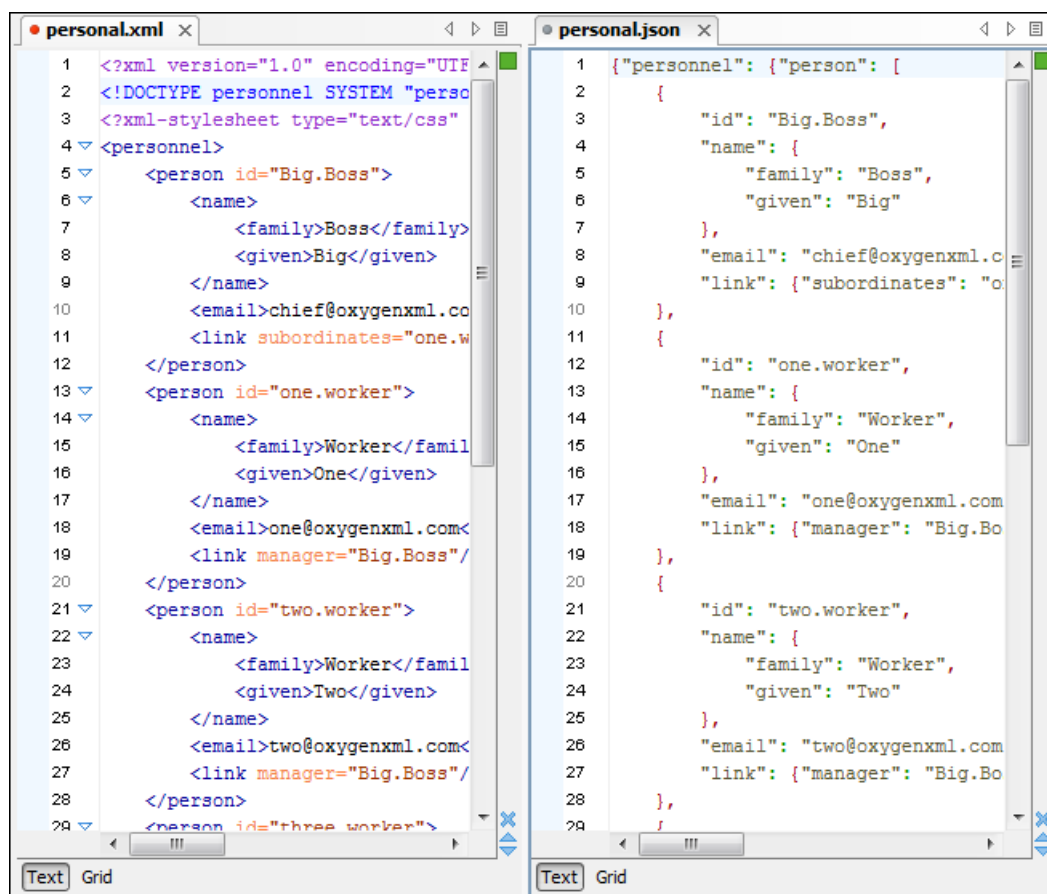
1. Go to menu **XML Tools > XML to JSON...**

The **XML to JSON** dialog 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 JSON Editor
5. Click the **OK** button.

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



Editing XProc Scripts

An XProc script is edited as an XML document that is validated against a RELAX NG schema. If the script has an associated transformation scenario, then the XProc engine from the scenario is invoked as validating engine. The default

engine for XProc scenarios is the Calabash engine which comes with Oxygen XML Developer plugin version 13.2.0

The content completion inside the element `input/inline` from the XProc namespace <http://www.w3.org/ns/xproc> offers elements from the following schemas depending on the `port` attribute of `input` and the parent of `input`. When invoking the content completion inside the XProc element `inline`, depending on the attribute `port` of its parent `input` element and the parent of element `input`, elements from different schemas are offered inside the proposals list.

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

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

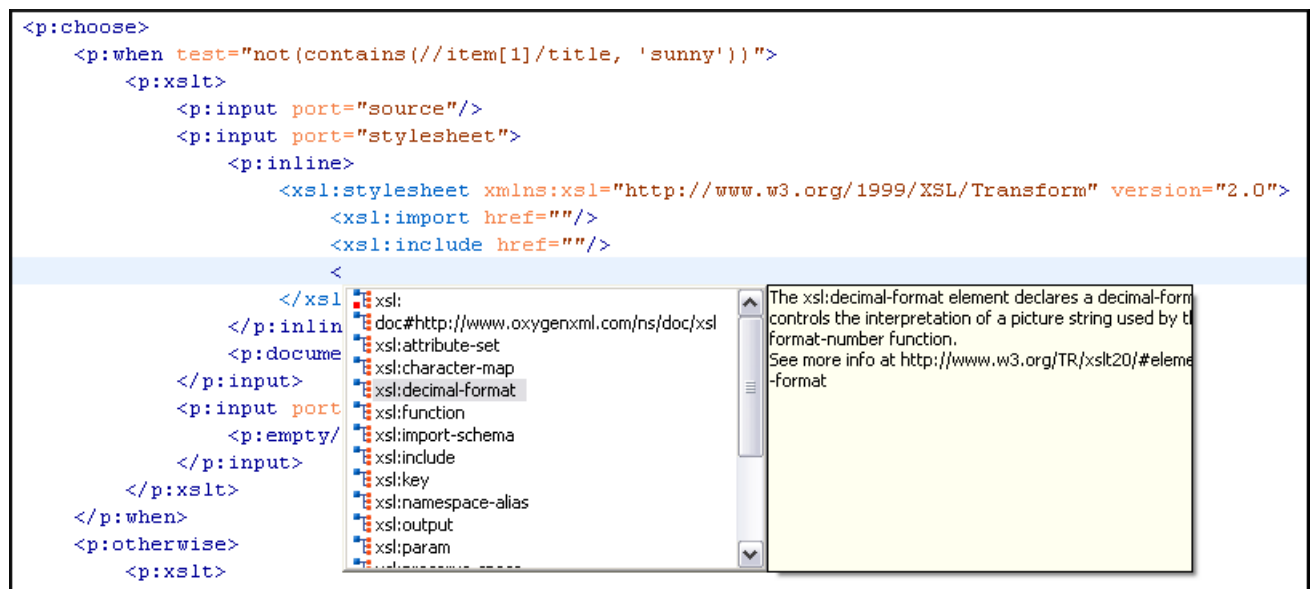


Figure 112: XProc Content Completion

Editing Schematron Schemas

Schematron is a simple and powerful Structural Schema Language for making assertions about patterns found in XML documents. It relies almost entirely on XPath query patterns for defining rules and checks. Schematron validation rules allow you to specify a meaningful error message (as opposed to a cryptic error code) which will be provided to the user if an error is encountered during validation stage.

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

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

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

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

Combined RELAX NG / W3C XML Schemas and Schematron Schema

Schematron rules can be embedded into W3C Schema through annotation (using the `appinfo` element) or in any element on any level of a RELAX NG Schema (taking into account that the RELAX NG validator ignores all elements that are not in the RELAX NG namespace).

Oxygen XML Developer plugin accepts such documents as Schematron validation schemas and it is able to extract and use the embedded rules. To validate a document with both RELAX NG schema and its embedded Schematron rules, you need two persistence associations like in the following example:

```
<?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 as Schematron Schema a W3C XML Schema having the Schematron rules embedded:

```
<?xml-model href="percent.xsd" type="application/xml"
schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

Validate an XML Document

To validate an XML document against a Schematron schema, invoke the **Validate** action either from the application's toolbar or from the **Project** view's contextual menu. If you would like to add a persistence association between your Schematron rules and the current edited XML document, use the Associate Schema action. A custom processing instruction is added into the document and the validation process will take into account the Schematron rules:

```
<?xml-model href="percent.sch" type="application/xml"
schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

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

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

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.

Chapter

5

Predefined Document Types

Topics:


- [Document Type](#)
- [The DocBook 4 Document Type](#)
- [The DocBook 5 Document Type](#)
- [The DocBook Targetset Document Type](#)
- [The DITA Topics Document Type](#)
- [The DITA Map Document Type](#)
- [The XHTML Document Type](#)
- [The TEI ODD Document Type](#)
- [The TEI P4 Document Type](#)
- [The TEI P5 Document Type](#)
- [The EPUB Document Type](#)

The following are the short presentations of some document types that come bundled with Oxygen XML Developer plugin . For each document type there are presented built-in transformation scenarios, document templates.

Document Type

A *document type* or *framework* is associated to an XML file according to a set of rules. It includes also many settings that improve editing in the Tagless editor for the category of XML files it applies for. These settings include:

- a default grammar used for validation and content completion in both Author mode and Text mode
- CSS stylesheet(s) for rendering XML documents in Author mode
- user actions invoked from toolbar or menu in Author mode
- predefined scenarios used for transformation of the class of XML documents defined by the document type
- XML catalogs
- directories with file templates
- user defined extensions for customizing the interaction with the content author in Author mode

 **Note:** The Author mode allows WYSIWYG-like visual editing of XML documents and is available only in the products Oxygen XML Editor and Oxygen XML Author.

The tagless editor comes with some predefined document types already configured when the application is installed on the computer. These document types describe well-known XML frameworks largely used today for authoring XML documents. Editing a document which conforms to one of these types is as easy as opening it or creating it from one of the predefined document templates which also come with the application.

The DocBook 4 Document Type

DocBook is a very popular set of tags for describing books, articles, and other prose documents, particularly technical documentation.

A file is considered to be a *DocBook 4* document when one of the following conditions are true:

- root element name is `book` or `article`
- the PUBLIC ID of the document contains the string `-//OASIS//DTD DocBook XML`

The schema of *DocBook 4* documents is `${frameworks}/docbook/dtd/docbookx.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Developer plugin install directory.

The XML catalog is stored in `${frameworks}/docbook/catalog.xml`.

Transformation Scenarios

Default transformation scenarios allow you to convert DocBook 4 to DocBook 5 documents and transform DocBook documents to HTML, HTML Chunk, PDF, XHTML, XHTML Chunk, WebHelp (experimental) and EPUB.

Templates

Default templates are available in the *New File wizard* and can be used for easily creating a skeletal form of a DocBook 4 book or article. These templates are stored in the `${frameworks}/docbook/templates/DocBook 4` folder.

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**

Inserting olink Links in Docbook Documents

An `olink` is a type of link between two Docbook XML documents.

The `olink` element is the equivalent for linking outside the current Docbook document. It has the attribute `targetdoc` for the document ID that contains the target element and the attribute `targetptr` for the ID (the value of an `id` or `xml:id` attribute) of the target element. The combination of those two attributes provides a unique identifier to locate cross references.

For example, the *Administrator Guide* is a book with the document ID `MailAdminGuide` and it contains a chapter about user accounts like the following:

```
<chapter id="user_accounts">
<title>Administering User Accounts</title>
<para>blah blah</para>
...
```

You can form a cross reference to that chapter by adding an `olink` in the *User Guide* like the following:

```
You may need to update your
<olink targetdoc="MailAdminGuide" targetptr="user_accounts">user accounts
</olink>
when you get a new machine.
```

1. Decide what documents are included in the domain for cross referencing.

An ID should be assigned to each document that will be referenced with an `olink`. Usually it is added as an `id` or `xml:id` attribute to the root element of the document. A document ID is a string that is unique for each document in your collection. For example the documentation may include a user's guide, an administrator's guide, and a reference document. These could have simple IDs like `ug`, `ag`, and `ref` or more specific IDs like `MailUserGuide`, `MailAdminGuide`, and `MailReference`.

2. Decide the output hierarchy.

For creating links between documents, the relative locations of the output documents must be known. Generally the HTML files for multiple documents are output to different directories if chunking is used. Before going further you must decide the names and locations of the HTML output directories for all the documents from the domain. Each directory will be represented by an element `<dir name="directory_name">` in the target database document. In the example from the next step the hierarchy is `documentation/guides/mailuser`, `documentation/guides/mailadmin`, `documentation/guides/reference`.

3. Create the target database document.

Each collection of documents has a master target database document that is used to resolve all `olinks` from that collection. The target database document is an XML file that is created once. It provides a framework that pulls in the target data for each document. The database document is static and all the document data is pulled in dynamically. An example is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE targetset
  SYSTEM "file:///tools/docbook-xsl/common/targetdatabase.dtd" [
<!ENTITY ugtargets SYSTEM "file:///doc/userguide/target.db">
<!ENTITY agtargetes SYSTEM "file:///doc/adminguide/target.db">
<!ENTITY reftargets SYSTEM "file:///doc/man/target.db">
]>
<targetset>
  <targetsetinfo>
    Description of this target database document,
    which is for the examples in olink doc.
  </targetsetinfo>

  <!-- Site map for generating relative paths between documents -->
  <sitemap>
    <dir name="documentation">
      <dir name="guides">
        <dir name="mailuser">
          <document targetdoc="MailUserGuide"
```

```

        baseuri="userguide.html">
        &ugtargets;
    </document>
</dir>
<dir name="mailadmin">
    <document targetdoc="MailAdminGuide">
        &agtargets;
    </document>
</dir>
</dir>
<dir name="reference">
    <dir name="mailref">
        <document targetdoc="MailReference">
            &reftargets;
        </document>
    </dir>
</dir>
</dir>
</sitemap>
</targetset>

```

An example of a target .db file:

```

<!DOCTYPE div
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">

  <ttml>Administering User Accounts</ttml>
  <xref>How to administer user accounts</xref>
  <div element="part" href="#d5e4" number="I">
    <ttml>First Part</ttml>
    <xref>Part I, "First Part"</xref>
    <div element="chapter" href="#d5e6" number="1">
      <ttml>Chapter Title</ttml>
      <xref>Chapter 1, Chapter Title</xref>
      <div element="sect1" href="#src_chapter" number="1"
targetptr="src_chapter">
        <ttml>Section1 Title</ttml>
        <xref>xreflabel_here</xref>
      </div>
    </div>
  </div>
</div>

```

4. Generate the target data files.

These files are the target .db files from the above example of target database document. They are created with the same DocBook transformation scenario as the HTML or XHTML output. The XSLT parameter called `collect.xref.targets` must be set to the value `yes`. The default name of a target data file is `target.db` but it can be changed by setting an absolute file path in the XSLT parameter `targets.filename`.

5. Insert olink elements in the DocBook XML documents.

When a DocBook XML document is edited in Author mode Oxygen provides the **Insert OLink** action on the toolbar. This action allows selecting the target of an olink from the list of all possible targets from a specified target database document. In the following image the target database document is called `target.xml`.

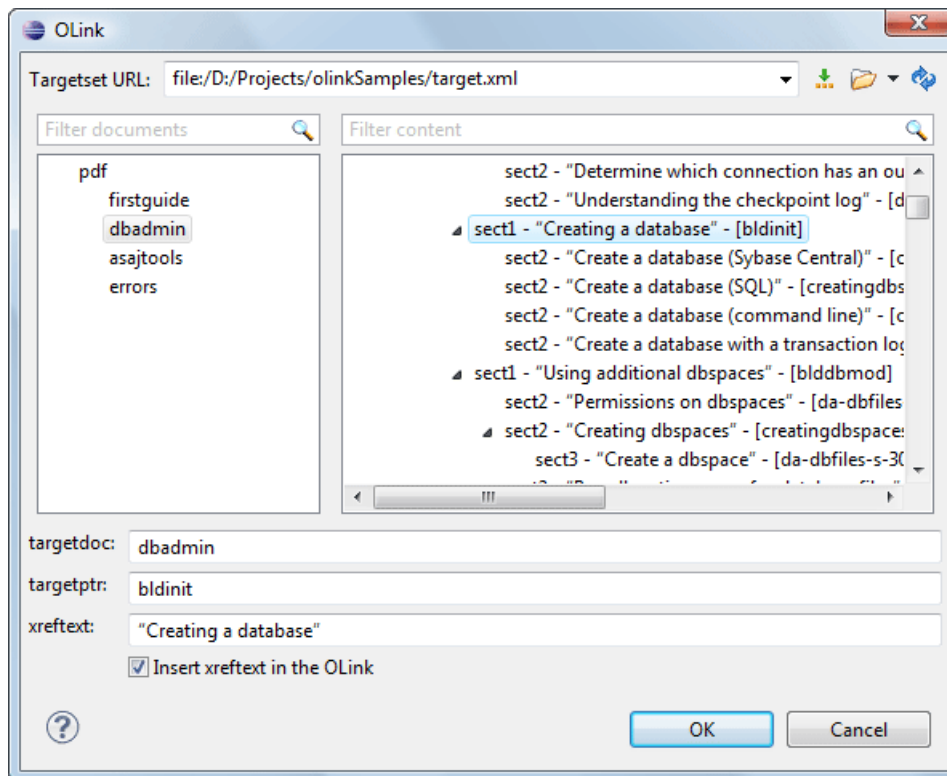


Figure 113: Insert OLink Dialog

6. Process each document for output.

That is done using a DocBook transformation scenario in which the URL of the target database document is set in the `target.database.document` parameter. The DocBook XSL stylesheets know how to resolve `olinks` in the output files using the value of this parameter.

The DocBook 5 Document Type

A file is considered to be a DocBook 5 document when the namespace is `http://docbook.org/ns/docbook`.

DocBook 5 documents use a Relax NG and Schematron schema located in `/${frameworks}/docbook/5.0/rng/docbookxi.rng`, where `/${frameworks}` is a subdirectory of the Oxygen XML Developer plugin install directory.

The XML catalog is stored in `/${frameworks}/docbook/5.0/catalog.xml`.

Transformation Scenarios

Default transformation scenarios allow you to transform DocBook 5 documents to HTML, HTML Chunk, PDF, XHTML, XHTML Chunk, WebHelp (experimental) and EPUB.

DocBook to EPUB Transformation

The EPUB specification recommends the use of *OpenType* fonts (recognized by their `.otf` file extension) when possible. To use a specific font:

- first you need to declare it in your CSS file, like:


```
@font-face {
font-family: "MyFont";
font-weight: bold;
```

```
font-style: normal;
src: url(fonts/MyFont.otf);
}
```

- tell the CSS where this font is used. To set it as default for h1 elements, use the `font-family` rule as in the following example:

```
h1 {
font-size:20pt;
margin-bottom:20px;
font-weight: bold;
font-family: "MyFont";
text-align: center;
}
```

- in your DocBook to EPUB transformation, set the `epub.embedded.fonts` parameter to `fonts/MyFont.otf`. If you need to provide more files, use comma to separate their file paths.

 **Note:** The `html.stylesheet` parameter allows you to include a custom CSS in the output EPUB.

Templates

Default templates are available in the *New File wizard* and can be used for easily creating a skeletal form of a DocBook 5 book or article. These templates are stored in the `${frameworks}/docbook/templates/DocBook 5` folder.

Here are some of the DocBook 5 templates available when creating *new documents from templates*.

- **Article**
- **Article with MathML**
- **Article with SVG**
- **Article with XInclude**
- **Book**
- **Book with XInclude**

Inserting olink Links in Docbook Documents

An `olink` is a type of link between two Docbook XML documents.

The `olink` element is the equivalent for linking outside the current Docbook document. It has the attribute `targetdoc` for the document ID that contains the target element and the attribute `targetptr` for the ID (the value of an `id` or `xml:id` attribute) of the target element. The combination of those two attributes provides a unique identifier to locate cross references.

For example, the *Administrator Guide* is a book with the document ID `MailAdminGuide` and it contains a chapter about user accounts like the following:

```
<chapter id="user_accounts">
<title>Administering User Accounts</title>
<para>blah blah</para>
...
```

You can form a cross reference to that chapter by adding an `olink` in the *User Guide* like the following:

```
You may need to update your
<olink targetdoc="MailAdminGuide" targetptr="user_accounts">user accounts
</olink>
when you get a new machine.
```

1. Decide what documents are included in the domain for cross referencing.

An ID should be assigned to each document that will be referenced with an `olink`. Usually it is added as an `id` or `xml:id` attribute to the root element of the document. A document ID is a string that is unique for each document

in your collection. For example the documentation may include a user's guide, an administrator's guide, and a reference document. These could have simple IDs like `ug`, `ag`, and `ref` or more specific IDs like `MailUserGuide`, `MailAdminGuide`, and `MailReference`.

2. Decide the output hierarchy.

For creating links between documents, the relative locations of the output documents must be known. Generally the HTML files for multiple documents are output to different directories if chunking is used. Before going further you must decide the names and locations of the HTML output directories for all the documents from the domain. Each directory will be represented by an element `<dir name="directory_name">` in the target database document. In the example from the next step the hierarchy is `documentation/guides/mailuser`, `documentation/guides/mailadmin`, `documentation/guides/reference`.

3. Create the target database document.

Each collection of documents has a master target database document that is used to resolve all `olinks` from that collection. The target database document is an XML file that is created once. It provides a framework that pulls in the target data for each document. The database document is static and all the document data is pulled in dynamically. An example is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE targetset
  SYSTEM "file:///tools/docbook-xsl/common/targetdatabase.dtd" [
<!ENTITY ugtargets SYSTEM "file:///doc/userguide/target.db">
<!ENTITY agtargetes SYSTEM "file:///doc/adminguide/target.db">
<!ENTITY reftargets SYSTEM "file:///doc/man/target.db">
]>
<targetset>
  <targetsetinfo>
    Description of this target database document,
    which is for the examples in olink doc.
  </targetsetinfo>

  <!-- Site map for generating relative paths between documents -->
  <sitemap>
    <dir name="documentation">
      <dir name="guides">
        <dir name="mailuser">
          <document targetdoc="MailUserGuide"
            baseuri="userguide.html">
            &ugtargetes;
          </document>
        </dir>
        <dir name="mailadmin">
          <document targetdoc="MailAdminGuide">
            &agtargetes;
          </document>
        </dir>
      </dir>
      <dir name="reference">
        <dir name="mailref">
          <document targetdoc="MailReference">
            &reftargetes;
          </document>
        </dir>
      </dir>
    </sitemap>
  </targetset>
```

An example of a target .db file:

```

<!DOCTYPE div
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">

  <ttitle>Administering User Accounts</ttitle>
  <xrefxref>How to administer user accounts</xrefxref>
  <div element="part" href="#d5e4" number="I">
    <ttitle>First Part</ttitle>
    <xrefxref>Part I, "First Part"</xrefxref>
    <div element="chapter" href="#d5e6" number="1">
      <ttitle>Chapter Title</ttitle>
      <xrefxref>Chapter 1, Chapter Title</xrefxref>
      <div element="sect1" href="#src_chapter" number="1"
targetptr="src_chapter">
        <ttitle>Section1 Title</ttitle>
        <xrefxref>xreflabel_here</xrefxref>
      </div>
    </div>
  </div>
</div>

```

4. Generate the target data files.

These files are the target .db files from the above example of target database document. They are created with the same DocBook transformation scenario as the HTML or XHTML output. The XSLT parameter called `collect.xref.targets` must be set to the value `yes`. The default name of a target data file is `target.db` but it can be changed by setting an absolute file path in the XSLT parameter `targets.filename`.

5. Insert olink elements in the DocBook XML documents.

When a DocBook XML document is edited in Author mode Oxygen provides the **Insert OLink** action on the toolbar. This action allows selecting the target of an olink from the list of all possible targets from a specified target database document. In the following image the target database document is called `target.xml`.

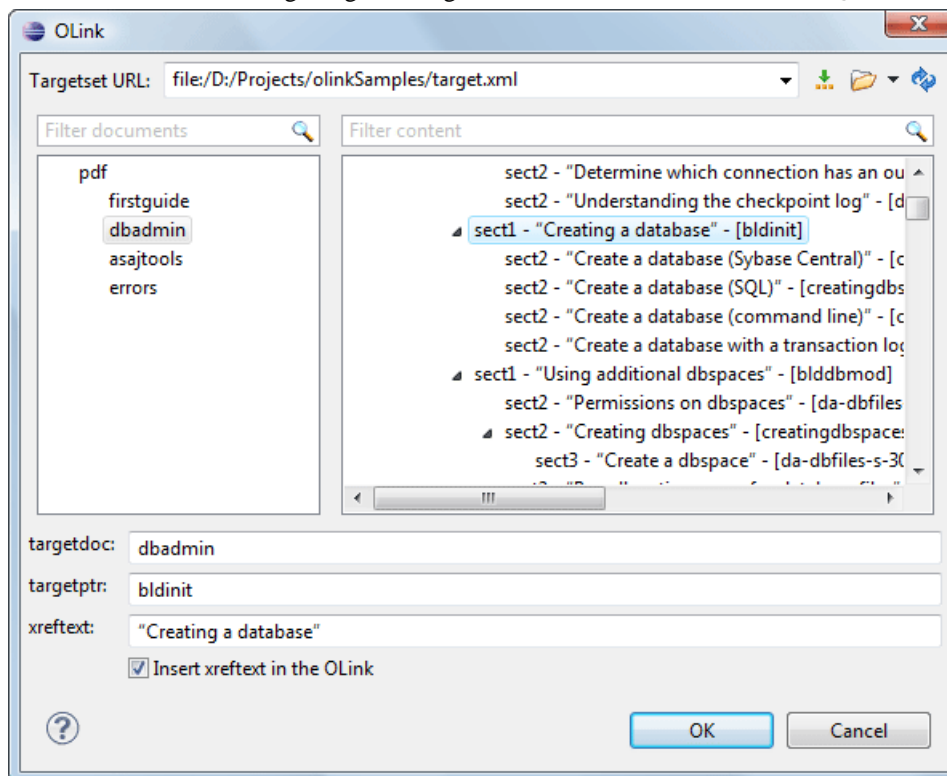


Figure 114: Insert OLink Dialog

6. Process each document for output.

That is done using a DocBook transformation scenario in which the URL of the target database document is set in the `target.database.document` parameter. The DocBook XSL stylesheets know how to resolve `olinks` in the output files using the value of this parameter.

The DocBook Targetset Document Type

DocBook *Targetset* documents are used to resolve cross references with DocBook `olinks`'s.

A file is considered to be a *Targetset* when the root name is `targetset`.

This type of documents use a DTD and schema located in

`${frameworks}/docbook/xsl/common/targetdatabase.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Developer plugin install directory.

Templates

There is a default template for *Targetset* documents in the `${frameworks}/docbook/templates/Targetset` folder. It is available when creating *new documents from templates*.

- **Docbook Targetset - Map** - New Targetset Map.

The DITA Topics Document Type

The Darwin Information Typing Architecture (DITA) is an XML-based architecture for authoring, producing, and delivering technical information. It divides content into small, self-contained topics that can be reused in different deliverables. The extensibility of DITA permits organizations to define specific information structures and still use standard tools to work with them.

A file is considered to be a DITA topic document when either of the following occurs:

- the root element name is one of the following: `concept`, `task`, `reference`, `dita`, `topic`
- PUBLIC ID of the document is one of the PUBLIC ID's for the elements above
- the root element of the file has an attribute named `DITAArchVersion` attribute from the “`http://dita.oasis-open.org/architecture/2005/`” namespace. This enhanced case of matching is only applied when the **Enable DTD processing** option from the is enabled.

The default schema used for DITA topic documents is located in `${frameworks}/dita/dtd/ditabase.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Developer plugin install directory.

The default XML catalog is `${frameworks}/dita/catalog.xml`.

Transformation Scenarios

The following default transformation scenarios are available for DITA Topics:

- **DITA XHTML** - Transforms a DITA topic to XHTML using DITA Open Toolkit 1.5.4;
- **DITA PDF (Idiom FO Plugin)** - Transforms a DITA topic to PDF using the DITA Open Toolkit 1.5.4 and the Apache FOP engine.

Templates

The default templates available for DITA topics are stored in `${frameworks}/dita/templates/topic` folder. They can be used for easily creating a DITA `concept`, `reference`, `task` or `topic`.

Here are some of the DITA templates available when creating *new documents from templates*.

- **DITA - Composite** - New DITA Composite
- **DITA - Composite with MathML** - New DITA Composite with MathML

- **DITA - Concept** - New DITA Concept
- **DITA - General Task** - New DITA Task
- **DITA - Glossentry** - New DITA Glossentry
- **DITA - Glossgroup** - New DITA Glossgroup
- **DITA - Machinery Task** - New DITA Machinery Task
- **DITA - Reference** - New DITA Reference
- **DITA - Task** - New DITA Task
- **DITA - Topic** - New DITA Topic
- **DITA - Learning Assessment** - New DITA Learning Assessment (learning specialization in DITA 1.2)
- **DITA - Learning Content** - New DITA Learning Content (learning specialization in DITA 1.2)
- **DITA - Learning Summary** - New DITA Learning Summary (learning specialization in DITA 1.2)
- **DITA - Learning Overview** - New DITA Learning Overview (learning specialization in DITA 1.2)

The DITA Map Document Type

DITA maps are documents that collect and organize references to DITA topics to indicate the relationships among the topics. They can also serve as outlines or tables of contents for DITA deliverables and as build manifests for DITA projects.

Maps allow scalable reuse of content across multiple contexts. They can be used by information architects, writers, and publishers to plan, develop, and deliver content.

A file is considered to be a DITA map document when either of the following occurs:

- root element name is one of the following: `map`, `bookmap`
- public id of the document is `-//OASIS//DTD DITA Map` or `-//OASIS//DTD DITA BookMap`.
- the root element of the file has an attribute named `class` which contains the value `map/map` and a `DITAArchVersion` attribute from the `http://dita.oasis-open.org/architecture/2005/` namespace. This enhanced case of matching is only applied when the **Enable DTD processing** option from the [Document Type Detection option page](#) is enabled.

The default schema used for DITA map documents is located in `${frameworks}/dita/DITA-OT/dtd/map.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Developer plugin install directory.

The default XML catalog is stored in `${frameworks}/dita/catalog.xml`.

Transformation Scenarios

The following default transformations are available:

- Predefined transformation scenarios allow you to transform a DITA Map to PDF, XHTML, WebHelp, EPUB and CHM files.
- **Run DITA OT Integrator** - Use this transformation scenario if you want to integrate a DITA OT plugin. This scenario runs an ANT task that integrates all the plug-ins from `DITA-OT/plugins` directory.
- **DITA Map Metrics Report** - Use this transformation scenario if you want to generate a DITA Map statistics report containing information like:
 - the number of processed maps and topics;
 - content reuse percentage;
 - number of elements, attributes, words, and characters used in the entire DITA Map structure;
 - DITA conditional processing attributes used in the DITA Maps;
 - words count;
 - information types like number of containing maps, bookmaps, or topics.

Many more output formats are available by clicking the **New** button. The transformation process relies on DITA Open Toolkit 1.5.4.

Templates

The default templates available for DITA maps are stored in `${frameworks}/dita/templates/map` folder. They can be used for easily creating a DITA map and bookmark files.

Here are some of the DITA Map templates available when creating *new documents from templates*:

- **DITA Map - Bookmark** - New DITA Bookmark
- **DITA Map - Map** - New DITA Map
- **DITA Map - Learning Map** - New DITA learning and training content specialization map
- **DITA Map - Learning Bookmark** - New DITA learning and training content specialization bookmark
- **DITA Map - Eclipse Map** - New DITA learning and training content specialization bookmark

The XHTML Document Type

The Extensible HyperText Markup Language, or XHTML, is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax.

A file is considered to be a XHTML document when the root element name is a `html`.

The schema used for these documents is located in `${frameworks}/xhtml/dtd/xhtml11-strict.dtd`, where `frameworks` is a subdirectory of the Oxygen XML Developer plugin install directory.

There are three default catalogs for XHTML document type:

- `${frameworks}/xhtml/dtd/xhtmlcatalog.xml`
- `${frameworks}/xhtml11/dtd/xhtmlcatalog.xml`
- `${frameworks}/xhtml11/schema/xhtmlcatalog.xml`

Transformation Scenarios

The following default transformation scenarios are available for XHTML:

- **XHTML to DITA concept** - Converts an XHTML document to a DITA concept document
- **XHTML to DITA reference** - Converts an XHTML document to a DITA reference document
- **XHTML to DITA task** - Converts an XHTML document to a DITA task document
- **XHTML to DITA topic** - Converts an XHTML document to a DITA topic document

Templates

Default templates are available for XHTML. They are stored in `${frameworksDir}/xhtml/templates` folder and they can be used for easily creating basic XHTML documents.

Here are some of the XHTML templates available when creating *new documents from templates*.

- **XHTML - 1.0 Strict** - New Strict XHTML 1.0
- **XHTML - 1.0 Transitional** - New Transitional XHTML 1.0
- **XHTML - 1.1 DTD Based** - New DTD based XHTML 1.1
- **XHTML - 1.1 DTD Based + MathML 2.0 + SVG 1.1** - New XHTML 1.1 with MathML and SVG insertions
- **XHTML - 1.1 Schema based** - New XHTML 1.1 XML Schema based

The TEI ODD Document Type

The **Text Encoding Initiative - One Document Does it all (TEI ODD)** is a TEI XML-conformant specification format that allows creating a custom TEI P5 schema in a literate programming fashion. A system of XSLT stylesheets called *Roma* was created by the TEI Consortium for manipulating the ODD files.

A file is considered to be a TEI ODD document when either of the following occurs:

- the file extension is `.odd`
- the document's namespace is `http://www.tei-c.org/ns/1.0`

The schema used for these documents is located in

`${frameworks}/tei/xml/tei/custom/schema/relaxng/brown_odds.rng`, where `${frameworks}` is a subdirectory of the Oxygen XML Developer plugin install directory.

There are two default catalogs for TEI ODD document type:

- `${frameworks}/tei/xml/tei/custom/schema/catalog.xml`
- `${frameworks}/tei/xml/tei/schema/catalog.xml`

Transformation Scenarios

The following default transformations are available:

- **TEI ODD XHTML** - Transforms a TEI ODD document into an XHTML document
- **TEI ODD PDF** - Transforms a TEI ODD document into a PDF document using the Apache FOP engine
- **TEI ODD EPUB** - Transforms a TEI ODD document into an EPUB document
- **TEI ODD DOCX** - Transforms a TEI ODD document into a DOCX document
- **TEI ODD ODT** - Transforms a TEI ODD document into an ODT document
- **TEI ODD RelaxNG XML** - Transforms a TEI ODD document into a RelaxNG XML document
- **TEI ODD to DTD** - Transforms a TEI ODD document into a DTD document
- **TEI ODD to XML Schema** - Transforms a TEI ODD document into an XML Schema document
- **TEI ODD to RelaxNG Compact** - Transforms a TEI ODD document into an RelaxNG Compact document

Templates

There is only one default template which is stored in the `${frameworks}/tei/templates/TEI_ODD` folder and can be used for easily creating a basic TEI ODD document. This template is available when creating [new documents from templates](#).

- **TEI ODD** - New TEI ODD document

The TEI P4 Document Type

The **Text Encoding Initiative (TEI) Guidelines** is an international and interdisciplinary standard that enables libraries, museums, publishers, and individual scholars to represent a variety of literary and linguistic texts for online research, teaching, and preservation.

A file is considered to be a TEI P4 document when either of the following occurs:

- the root's local name is `TEI . 2`
- the document's public id is `-/TEI P4`

The DTD schema used for these documents is located in `${frameworks}/tei/tei2xml.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Developer plugin install directory.

There are two default catalogs for TEI P4 document type:

- `${frameworks}/tei/xml/teip4/schema/dtd/catalog.xml`
- `${frameworks}/tei/xml/teip4/custom/schema/dtd/catalog.xml`

Transformation Scenarios

The following default transformations are available:

- **TEI HTML** - Transforms a TEI document into a HTML document
- **TEI P4 -> TEI P5 Conversion** - Convert a TEI P4 document into a TEI P5 document
- **TEI PDF** - Transforms a TEI document into a PDF document using the Apache FOP engine

Templates

The default templates are stored in `/${frameworks}/tei/templates/TEI_P4` folder and they can be used for easily creating basic TEI P4 documents. These templates are available when creating [new documents from templates](#).

- **TEI P4 - Lite** - New TEI P4 Lite
- **TEI P4 - New Document** - New TEI P4 standard document

The TEI P5 Document Type

The TEI P5 document type is the same with that for TEI P4 with the following exceptions:

- A file is considered to be a TEI P5 document when the namespace is `http://www.tei-c.org/ns/1.0`.
- The schema is located in `/${frameworks}/tei/xml/tei/custom/schema/relaxng/tei_allPlus.rng`, where `/${frameworks}` is a subdirectory of the Oxygen XML Developer plugin install directory.
- A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `graphic` DITA element with the `url` attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

Transformation Scenarios

The following default transformations are available:

- **TEI P5 XHTML** - Transforms a TEI P5 document into a XHTML document
- **TEI P5 PDF** - Transforms a TEI P5 document into a PDF document using the Apache FOP engine
- **TEI EPUB** - Transforms a TEI P5 document into an EPUB output. The EPUB output will contain any images referenced in the TEI XML document.
- **TEI DOCX** - Transforms a TEI P5 document into a DOCX (OOXML) document. The DOCX document will contain any images referenced in the TEI XML document.
- **TEI ODT** - Transforms a TEI P5 document into an ODT (ODF) document. The ODT document will contain any images referenced in the TEI XML document.

Templates

The default templates are stored in `/${frameworks}/tei/templates/TEI_P5` folder and they can be used for easily creating basic TEI P5 documents. These templates are available when creating [new documents from templates](#).

- **TEI P5 - All** - New TEI P5 All
- **TEI P5 - Bare** - New TEI P5 Bare
- **TEI P5 - Lite** - New TEI P5 Lite
- **TEI P5 - Math** - New TEI P5 Math
- **TEI P5 - Speech** - New TEI P5 Speech
- **TEI P5 - SVG** - New TEI P5 with SVG extensions
- **TEI P5 - XInclude** - New TEI P5 XInclude aware

The EPUB Document Type

Three distinct frameworks support the EPUB document type:

- **NCX** - A declarative global navigation definition.
- **OCF** - The Open Container Format(OCF) defines a mechanism by which all components of an Open Publication Structure(OPS) can be combined into a single file-system entity.
- **OPF** - The Open Packaging Format(OPF) defines the mechanism by which all components of a published work conforming to the Open Publication Structure(OPS) standard including metadata, reading order and navigational information are packaged into an OPS Publication.

Chapter 6

Grid Editor

Topics:

- [Layouts: Grid and Tree](#)
- [Navigating the Grid](#)
- [Specific Grid Actions](#)
- [Drag and Drop in the Grid Editor](#)
- [Copy and Paste in the Grid Editor](#)
- [Bidirectional Text Support in the Grid Editor](#)

In the grid editor the XML document is displayed as a structured grid of nested tables in which the text content can be modified by non technical users without editing directly the XML tags. The tables can be expanded and collapsed with a mouse click to show or hide the elements of the document as needed. The document structure can also be changed easily with drag and drop operations on the grid components. The tables can be zoomed using **(Ctrl - +)**, **(Ctrl - -)**, **(Ctrl - 0)** or **(Ctrl - mouse wheel)**.

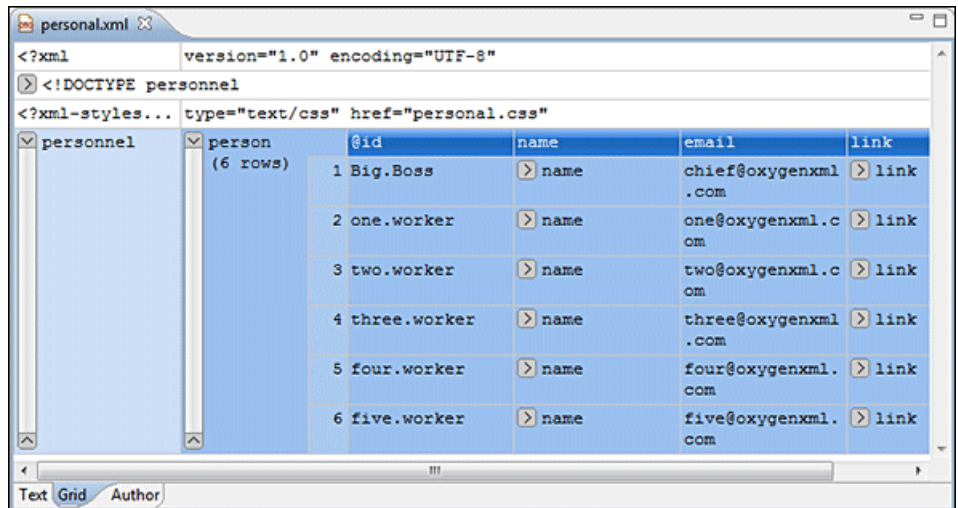


Figure 115: The Grid Editor

You can switch between the text tab and the grid tab of the editor panel with the buttons **Text** and **Grid** available at the bottom of the editor panel. .

If the edited document is associated with a schema (DTD, XML Schema, Relax NG, etc.), the editor offers content completion for the element and attributes names and values. If you choose to insert an element that has required content, the subtree of needed elements and attributes will automatically be included.

To display the content completion popup you have to start editing, for example by double clicking the cell. Pressing **(Ctrl - Space)** will also display the popup.

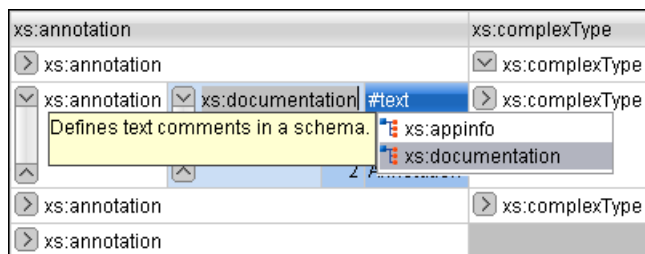


Figure 116: Content Completion in Grid Editor

Layouts: Grid and Tree

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

@id	first	last
1 10001	Jhon	Doe
2 10002	Mark	Ewing
3 10003	Dave	Flint

Figure 117: Grid Layout

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

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

Figure 118: Tree Layout

or the contextual menu > **Grid mode/Tree mode**.

Navigating the Grid

At first the content of a document opened in the grid tab is collapsed, only the root element and its attributes being shown. The grid disposition of the node names and values is very similar to a web form or a dialog. The same set of key shortcuts used to select dialog components is also available in the grid page. For instance moving to the next editable value in a table row is done using the **(Tab)** key. Moving to the previous cell employs using the **(Shift-Tab)** key. Changing a value involves pressing the **(Enter)** key or start typing the new value directly, and, when the editing is finished, pressing **(Enter)** again to commit the data into the document.

The arrow keys and the **(Page Up/Down)** keys can be used for navigation. By pressing **(Shift)** while using these keys you can create a selection zone. To add other nodes that are not close to this zone, you can use the mouse and the **(Ctrl)** key (**(Command)** on Mac OS X).



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

- Ctrl - Up scrolls the grid upwards
- Ctrl - Down scrolls the grid downwards
- Ctrl - Left scrolls the grid to the left
- Ctrl - Right scrolls the grid to the right

An arrow sign displayed to the left of the node name indicates that this node has child nodes. You can click this sign to display the children. The expand/collapse actions can be also invoked by pressing the **(NumPad Plus)** and **(NumPad Minus)** keys. The same

expand/collapse actions can be accessed from the submenu **Expand/Collapse** 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 are available in the **Document** menu and on the **Grid** toolbar which is opened from menu **Window > Show Toolbar > Grid**.


Sorting a Table Column

You can sort the table by a specific column. The sorting can be either ascending or descending.

The icons for this pair of actions are:  

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

Inserting a Row in a Table

A new row can be added using the copy/paste row operation, or by invoking the  **Insert row** action from the **Table** contextual menu.

A shorter way of inserting a new row is to move the selection over the row header, and then to press **(Enter)**. The row header is the zone in the left of the row that holds the row number. The new row will be inserted below the selection.

Inserting a Column in a Table

You can insert a column after the selected one, using the  **Insert column** action from the **Table** contextual menu.

Clearing the Content of a Column

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

Adding Nodes

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


The sub-menus containing detailed actions are:

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

Duplicating Nodes

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

Refresh Layout

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

Start Editing a Cell Value

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

Stop Editing a Cell Value

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

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

Drag and Drop in the Grid Editor

The arrangement of the different sections in your XML document in the grid editor is made easy by the drag and drop features.

Using drag and drop you can:

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

These operations are available for single and multiple selection.

Note that while dragging, the editor paints guide-lines showing locations where the nodes can be dropped.

Nodes can also be dragged outside the grid editor and text from other applications can be dropped inside the grid. See [Copy and Paste in the Grid Editor](#) for details.

Copy and Paste in the Grid Editor

The selection in the grid is a bit complex relative to the selection in a text component. It consists of a current selected cell and additional selected cells. These additional cells are either hand picked by the user using the mouse, or are implied by the current selected cell. To be more specific, let's consider you click the name of the column - this becomes the current selected cell, 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. 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 copied nodes from the grid can also be pasted into the text editor or other applications. When copying from grid into the text editor or other text based applications the inserted string represents the nodes serialization. The nodes from tables can be copied using HTML or RTF in table format. The resulting cells contain only the concatenated values of the text nodes.

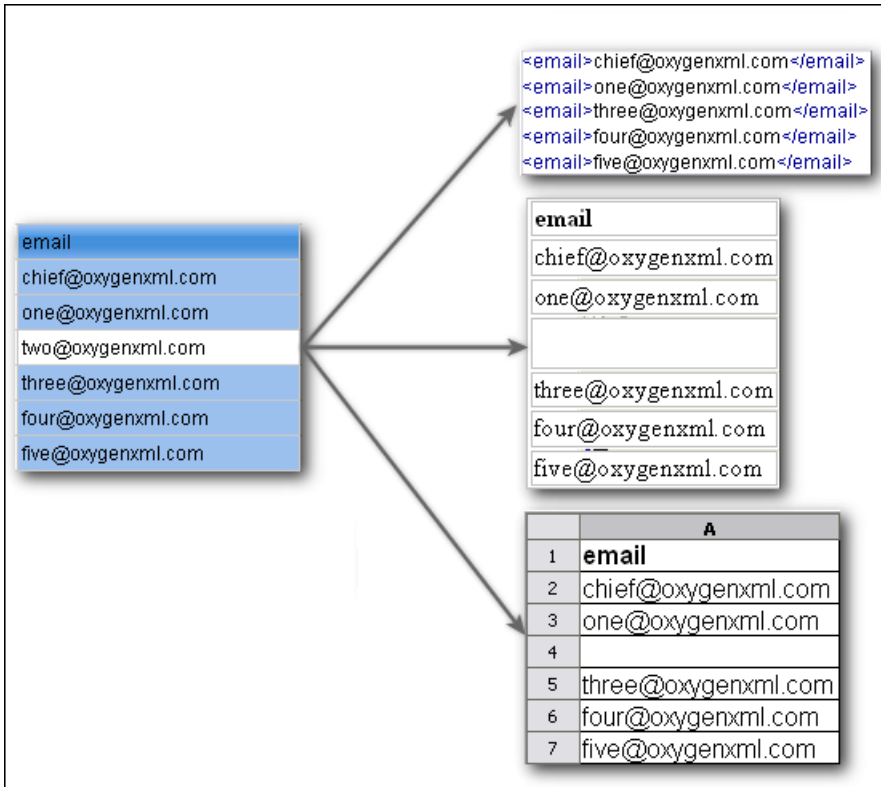


Figure 119: Copying from grid to other editors

In the grid editor you can paste wellformed xml content or tab separated values from other editors. If you paste xml content the result will be the insertion of the nodes obtained by parsing this content.

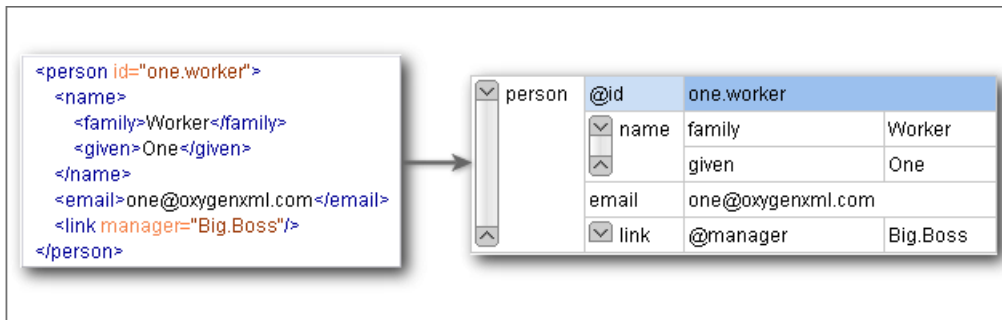


Figure 120: Copying XML data into grid

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


Id	Email
Id1	Email1
Id2	Email2
Id3	Email3

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

Figure 121: Copying tab separated values into grid

Bidirectional Text Support in the Grid Editor

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

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

<?xml	version="1.0" encoding="UTF-8"
sample	#text
(9 rows)	1 عندما يريد العالم أن يتكلم، فهو يتحدث بلغة يونيكود.
	2 Quan el món vol conversar, parla Unicode
	3 כאשר העולם רוצה לדבר, הוא מדבר ב-Unicode
	4 Ha a világ beszélni akar, azt Unicode-ul mondja
	5 Quando il mondo vuole comunicare, parla Unicode
	6 世界的に話すなら、Unicode です。
	7 세계를 향한 대화, 유니코드로 하십시오
	8 Når verden vil snakke, snakker den Unicode
	9 Når verda ønskjer å snakke, talar ho Unicode

Figure 122: Default left to right text orientation

<?xml	"version="1.0" encoding="UTF-8"
sample	#text
(9 rows)	1 عندما يريد العالم أن يتكلم، فهو يتحدث بلغة يونيكود.
	2 Quan el món vol conversar, parla Unicode
	3 כאשר העולם רוצה לדבר, הוא מדבר ב-Unicode
	4 Ha a világ beszélni akar, azt Unicode-ul mondja
	5 Quando il mondo vuole comunicare, parla Unicode
	6 世界的に話すなら、Unicode です。
	7 세계를 향한 대화, 유니코드로 하십시오
	8 Når verden vil snakke, snakker den Unicode
	9 Når verda ønskjer å snakke, talar ho Unicode

Figure 123: Right to left text orientation

Chapter 7

Transforming Documents

Topics:

- [Output Formats](#)
- [Transformation Scenario](#)
- [XSLT Processors](#)
- [XSL-FO Processors](#)
- [XProc Transformations](#)

XML is mainly used to store, carry, and exchange data, when you want to view the data in a more user friendly form, you must either use the [Author editor page](#), have an XML-compliant user agent or transform it to a format that can be read by other user agents. This process is known as transformation.

Status messages generated during transformation are displayed in the [Console view](#).

Output Formats

Within the current version of `Oxygen XML Developer plugin` you can transform your XML documents to the following formats without having to exit from the application:

- **PDF** - Adobe Portable Document Format (PDF) is a compact binary file format that can be viewed and printed by anyone, anywhere across a broad range of hardware and software using the free PDF Viewer from [Adobe](#).
- **PS** - PostScript is the leading printing technology from [Adobe](#) for high-quality, best-in-class printing solutions ranging from desktop devices to the most advanced digital presses, platemakers, and large format image setters in the world. PostScript files can be viewed using viewers such as GhostScript, but are more commonly created as a prepress format.
- **TXT** - Text files are Plain ASCII Text and can be opened in any text editor or word processor.
- **XML** - XML stands for eXtensible Markup Language and is a [W3C](#) standard markup language, much like HTML, which was designed to describe data. XML tags are not predefined in XML. You must define your own tags. XML uses a Document Type Definition (DTD), an XML Schema or a Relax NG schema to describe the data. XML with a DTD, XML Schema or Relax NG schema is designed to be self-descriptive. XML is not a replacement for HTML. XML and HTML were designed with different goals:
 - XML was designed to describe data and to focus on what data is.
 - HTML was designed to display data and to focus on how data looks.
 - HTML is about displaying information, XML is about describing information.
- **XHTML** - XHTML stands for eXtensible HyperText Markup Language, a [W3C](#) standard. XHTML is aimed to replace HTML. While almost identical to HTML 4.01, XHTML is a stricter and cleaner version of HTML. XHTML is HTML defined as an XML application.

For transformation to formats that are not listed above simply install the tool chain required to perform the transformation and process the xml files created with `Oxygen XML Developer plugin` in accordance with the processor instructions.

All formatting during a transformation is provided under the control of an Extensible Stylesheet (XSLT). Specifying the appropriate XSLT enables transformation to the above formats and preparation of output files for specific user agent viewing applications, including:

- **HTML** - HTML stands for Hyper Text Markup Language and is a [W3C Standard](#) for the World Wide Web. HTML is a text file containing small markup tags. The markup tags tell the Web browser how to display the page. An HTML file must have an `htm` or `html` file extension. An HTML file can be created using a simple text editor.
- **HTML Help** - [Microsoft HTML Help](#) is the standard help system for the Windows platform. Authors can use HTML Help to create online help for a software application or to create content for a multimedia title or Web site. Developers can use the HTML Help API to program a host application or hook up context-sensitive help to an application.
- **JavaHelp** - JavaHelp software is a full-featured, platform-independent, extensible help system from [Sun Microsystems/Oracle](#) that enables developers and authors to incorporate online help in applets, components, applications, operating systems, and devices. JavaHelp is a free product and the binaries for JavaHelp can be redistributed.
- **Eclipse Help** - Eclipse Help is the help system incorporated in the [Eclipse platform](#) that enables Eclipse plugin developers to incorporate online help in their plugins.

Many other target formats are possible, these are the most popular. The basic condition for transformation to any format is that your source document is well-formed. Always, make sure that the XSL used for the transformation is the right one according to the desired output format and with the input source definition. For example, if you want to transform to HTML format using a DocBook html stylesheet, your source xml document should respect the DocBook DTD.

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an output document by using special formatting vocabulary.

XSL consists of three parts:

- **XSL Transformations (XSLT)** - XSLT is a language for transforming XML documents.
- **XML Path (XPath) Language** - XPath is an expression language used by XSLT to access or refer parts of an XML document. XPath is also used by the XML Linking specification.

- **XSL Formatting Objects (XSL:FO)** - XSL:FO is an XML vocabulary for specifying formatting semantics.


Oxygen XML Developer plugin supports XSLT/XPath version 1.0 using Saxon 6.5.5, Xalan, Xsltproc, MSXML (3.0, 4.0, .NET) and XSLT/XPath 2.0 by using Saxon 9.3.0.5 HE, Saxon 9.3.0.5 PE, and Saxon 9.3.0.5 EE. [The validation](#) is done also depending on the stylesheet version.

Transformation Scenario

Before transforming an XML document in Oxygen XML Developer plugin you must define a transformation scenario to apply to that document. A scenario is a set of values for various parameters defining a transformation. It is not related to any particular document but to a document type:


- **Scenarios that apply to XML files** - Such a scenario contains the location of an XSLT stylesheet that is applied on the edited XML document and other transform parameters.
- **Scenarios that apply to XSLT files** - Such a scenario contains the location of an XML document that the edited XSLT stylesheet is applied on and other transform parameters.
- **Scenarios that apply to XQuery files** - Such a scenario contains the location of an XML source that the edited XQuery file is applied on and other transform parameters. When the XML source is a native XML database the XML source field of the scenario is empty because the XML data is read with XQuery specific functions like `document ()`. When the XML source is a local XML file the URL of the file is specified in the XML input field of the scenario.
- **Scenarios that apply to SQL files** - Such a scenario specifies a database connection for the database server that will run the SQL file associated with the scenario. The data processed by the SQL script is located in the database.
- **Scenarios that apply to XProc files** - Such a scenario contains the location of an XProc script and other transform parameters.
- **DITA-OT scenarios** - Such a scenario provides the parameters for an Ant transformation that will execute a DITA-OT build script. Oxygen 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.

A scenario can be created at [document type](#) level or at global level. The scenarios defined at document type level are available only for the documents that match that document type. The global scenarios are available for any document.


In order to apply a transformation scenario one has to press the  **Apply Transformation Scenario** button from the **Transformation** toolbar.

Batch Transformation

A transform action can be applied on a batch of files [from the Project view's contextual menu](#) without having to open the files:

-  **Apply Transformation Scenario** - Applies the transformation scenario associated to each of the selected files. If the currently processed file does not have an associated transformation scenario then a warning is displayed in the **Warnings** view.
- **Transform with...** - Allows you to select one transformation scenario to be applied to each one of the currently selected files.

`xsl:message` output information is collected and displayed in the [Results View](#). All entries in the Results View point to the location of the code that triggered them.

 **Note:** When you trigger a batch transformation, the output messages from the previous one are deleted.




Built-in Transformation Scenarios

If the **Apply Transformation Scenario** button from the **Transformation** toolbar is pressed and there is no scenario associated with the edited document but this contains an `xml-stylesheet` processing instruction referring to a XSLT stylesheet (commonly used to display the document in Internet browsers), then Oxygen XML Developer plugin will prompt the user and offer the option to associate the document with a default scenario based on this processing instruction.

The default scenario contains in the **XSL URL** field the URL from the `href` attribute of the processing instruction. This scenario will have the **Use xml-stylesheet declaration** checkbox set by default, will use Saxon as transformation engine, will perform no FO processing and will store the result in a file with the same URL as the edited document except the extension which will be changed to `html`. The name and path will be preserved because the output file name is specified with the help of two *editor variables*: `#{cfd}` and `#{cfn}`.

Oxygen XML Developer plugin comes with preconfigured built-in scenarios for usual transformations that enable the user to quickly obtain the desired output. All you need to do is to associate one of the built-in scenarios with the current edited document and then apply the scenario with just one click.

Defining a New Transformation Scenario

The **Configure Transformation Scenario** dialog is used to associate a scenario from the list of all scenarios with the edited document by selecting an entry from the list. The dialog is opened by pressing the  **Configure Transformation Scenario** button on the **Transformation** toolbar of the document view. Once selected, the scenario will be applied with only one click on the  **Apply Transformation Scenario** from the same toolbar. Pressing the  **Apply Transformation Scenario** button before associating a scenario with the edited document will invoke first the **Configure Transformation Scenario** dialog and then apply the selected scenario.

The **Configure Transformation Scenario** dialog can be opened using one of the methods previously presented or by selecting **XML > Configure transformation scenario**. (**Alt+Shift+T C**) (**Cmd+Alt+T C on Mac OS**).

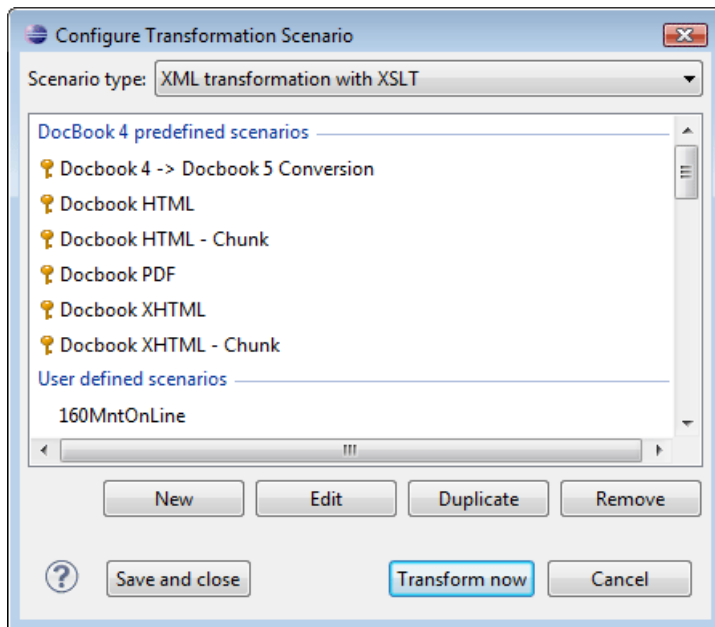


Figure 124: Configure Transformation Scenario Dialog

The **Scenario type** allows you to choose what type of user defined transformation scenario is displayed:

- **All** - No filtering. All user-defined scenarios are displayed.
- **XML transformation with XSLT** - Transformation scenarios that apply an XSLT stylesheet over an XML.
- **XML transformation with XQuery** - Transformation scenarios that apply an XQuery over an XML.
- **DITA OT transformation** - Transformation scenarios that use the DITA Open Toolkit (DITA-OT) to transform XML content into an output format.
- **ANT transformation** - Transformation scenarios that execute ANT scripts.
- **XSLT transformation** - Transformation scenarios that apply an XSLT stylesheet over an XML file.
- **XProc transformation** - Transformation scenarios that execute XProc XML pipelines.
- **XQuery transformation** - Represents a transformation that consists in applying an XQuery over an XML.
- **SQL transformation** - Executes an SQL over a database.

If you want an XSLT scenario select as **Scenario type** either **XML transformation with XSLT** or **XSLT transformation** then complete the dialog as follows:

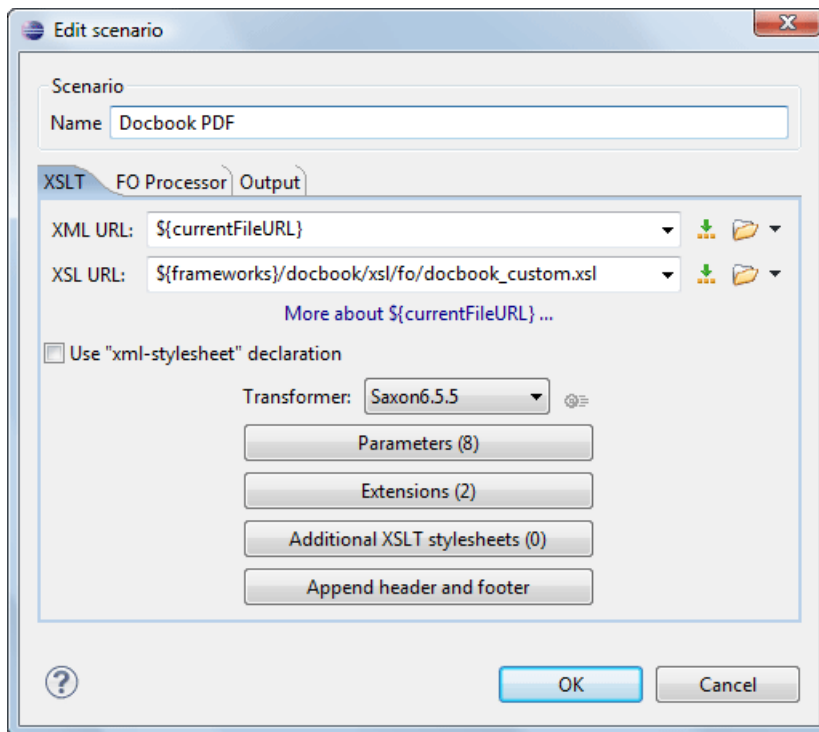



Figure 125: The Configure Transformation Dialog - XSLT Tab

- **XML URL** - Specifies an input XML file to be used for the transformation. Please note that this URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file will be used directly.
 - ☞ **Note:** If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, then the XML input of the transformation is passed to that URI resolver.
 - ☞ **Note:** If the transformer engine is one of the built-in XSLT 2.0 engines and *the name of an initial template is specified in the scenario* then the **XML URL** field can be empty. The **XML URL** field can also be empty in case of *external XSLT processors*. In all other cases a non-empty XML URL value is mandatory.

The following buttons are shown immediately after the input field:

- 📄 **Insert Editor Variables** - Opens a pop-up menu allowing 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 allowing to select a local file for the text field.
- 🌐 **Browse for remote file** - Opens an URL browser dialog allowing to select a remote file for the text field.
- 📦 **Browse for archived file** - Opens a zip archive browser dialog allowing to select a file from a zip archive that will be inserted in the text field.
-
- 📄 **Open in editor** - Opens the file with the path specified in the text field in an editor panel.
- **XSL URL** - Specifies an input XSL file to be used for the transformation. Please note that this URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file will be used directly. The set of browsing buttons described above for the XML URL field are also available for the XSL URL field.
- **Use "xml-stylesheet" declaration** - Use the stylesheet declared with an `xml-stylesheet` declaration instead of the stylesheet specified in the **XSL URL** field. By default this checkbox is not selected and the transformation applies the XSLT stylesheet specified in the **XSL URL** field. If it is checked the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction.

- **Transformer** - This combo box contains all the transformer engines available for applying the stylesheet. These are the built-in engines and *the external engines defined in the user preferences*. If you want to change the default selected engine just select other engine from the drop down list of the combo box. For XSLT/XQuery files only, if no validation scenario is associated, the transformer engine will also be used in validation process, if it has validation support.
- **Parameters** - Opens *the dialog for configuring the XSLT parameters*. In this dialog you set any global XSLT parameters of the main stylesheet set in the **XSL URL** field or of the additional stylesheets set with the button **Additional XSLT stylesheets**. If the XSLT transformer engine is custom defined this dialog cannot be used to configure the parameters sent to the custom engine. In this case you can copy all parameters from the dialog using the contextual menu actions and edit the custom XSLT engine to include in the command line the necessary parameters.
- **Append header and footer** - Opens a dialog for specifying an URL for a header HTML file added at the beginning of the result of an HTML transformation and a URL for a footer HTML file added at the end of the HTML result of the transformation.
- **Additional XSLT stylesheets** - Opens *the dialog for adding XSLT stylesheets* which are applied on the result of the main stylesheet specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document.
- **Extensions** - Opens *the dialog for configuring the XSLT/XQuery extension jars or classes* which define extension Java functions or extension XSLT elements used in the XSLT/XQuery transformation.
-  **Advanced options** - Configure advanced options specific for the Saxon HE / PE / EE engine. They are the same options as *the ones set in the user preferences* but they are configured as a specific set of transformation options for each transformation scenario. By default if you do not set a specific value in the transformation scenario each advanced option has the same value as the global option with the same name *set in the user preferences*.

The advanced options include two options that are not available globally in the user preferences: the initial XSLT template and the initial XSLT mode of the transformation. They are Saxon specific options that allow imposing the name of the first XSLT template that starts the XSLT transformation or the initial mode of transformation.

The advanced options specific for Saxon HE / PE / EE are:

- **Mode ("-im")** - Specifies to the transformer the initial template mode
- **Template ("-it")** - Specifies the name of the initial template to the transformer. When specified, the XML input URL for the transformation scenario is optional.
- **Use a configuration file ("-config")** - The URL input points to a Saxon advanced options configuration file.
- **Version warnings ("-versmsg")** - Warns you when the transformation is applied to an XSLT 1.0 stylesheet.
- **Line numbering ("-l")** - Error line number is included in the output messages.
- **Debugger trace into XPath expressions (applies to debugging sessions)** - Instructs the *XSLT Debugger* to step into XPath expressions.
- **DTD validation of the source ("-dtd")** - The following options are available:
 - **On**, requests *DTD-based* validation of the source file and of any files read using the document() function;
 - **Off** (default setting) suppresses DTD validation.
 - **Recover**, performs DTD validation but treats the errors as non-fatal.

Note that any external DTD is likely to be read even if not used for validation, because DTDs can contain definitions of entities.

- **Recoverable errors ("-warnings")** - Policy for handling recoverable errors in the stylesheet: Allows you to choose how dynamic errors are handled. One of the following options can be selected:
 - **Recover silently ("silent")** ;
 - **Recover with warnings ("recover")** - default setting;
 - **Signal the error and do not attempt recovery ("fatal")**.
- **Strip whitespaces ("-strip")** - Strip whitespaces feature can be one of the following three options:
 - **All ("all")** - Strips all whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes in the source document.

- **Ignorable ("ignorable")** - Strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
- **None ("none")** - default setting. No whitespaces are stripped before further processing. However, whitespace will still be stripped if this is specified in the stylesheet using `xsl:strip-space`.
- **Optimization level ("-opt")** - Set optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization); currently all values other than 0 result in full optimization but this is likely to change in the future. The default is full optimization; this feature allows optimization to be suppressed in cases where reducing compile time is important, or where optimization gets in the way of debugging, or causes extension functions with side-effects to behave unpredictably. (Note however, that even with no optimization, the lazy evaluation may still cause the evaluation order to be not as expected.)

The advanced options available only in Saxon PE / EE are:

- **Allow calls on extension functions ("-ext")** - If checked, the stylesheet is allowed to call external Java functions. This does not affect calls on integrated extension functions, including Saxon and EXSLT extension functions. This option is useful when loading an untrusted stylesheet, perhaps from a remote site using an `http://` URL; it ensures that the stylesheet cannot call arbitrary Java methods and thereby gain privileged access to resources on your machine.

The advanced options available only in Saxon EE are:

- **Validation of the source file ("-val")** - Requests schema-based validation of the source file and of any files read using the `document()` or similar functions. Validation is available only with Saxon-EE, and this flag automatically switches on the `-sa` option. Available options:
 - **Schema validation ("strict")** - This mode requires an XML Schema and specifies that the source documents should be parsed with schema-validation enabled.
 - **Lax schema validation ("lax")** - This mode specifies if the source documents should be parsed with schema-validation enabled if an XML Schema is provided.
 - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.
- **Validation errors in the results tree treated as warnings ("-outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.
- **Write comments for non-fatal validation errors of the result document** - The validation messages are written (where possible) as a comment in the result document itself.
- **Initializer class** - Equivalent with the `-init` Saxon command line argument. The value is the name of a user-supplied class that implements the interface `net.sf.saxon.lib.Initializer`; this initializer will be 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.

When creating a scenario that applies to an XML file, Oxygen XML Developer plugin fills the **XML URL** field with the default variable `#{currentFileURL}`. This means the input for the transformation is taken from the currently edited file. You can modify this value to some other file path. This is the case when you are currently editing a section from a large document, but you want the transformation to be performed on the main document. You can specify in this case either a full absolute path: `file:/c:/project/docbook/test.xml` or a path relative to one of the editor variables, for example or the current file directory: `#{cfdu}/test.xml`.

When the scenario applies to XSL files, the field **XSL URL** initially contains `#{currentFile}` editor variable. Just like in the XML case, you can specify here the path to a master stylesheet. The path can be configured using the *editor variables* or the *custom editor variables*.

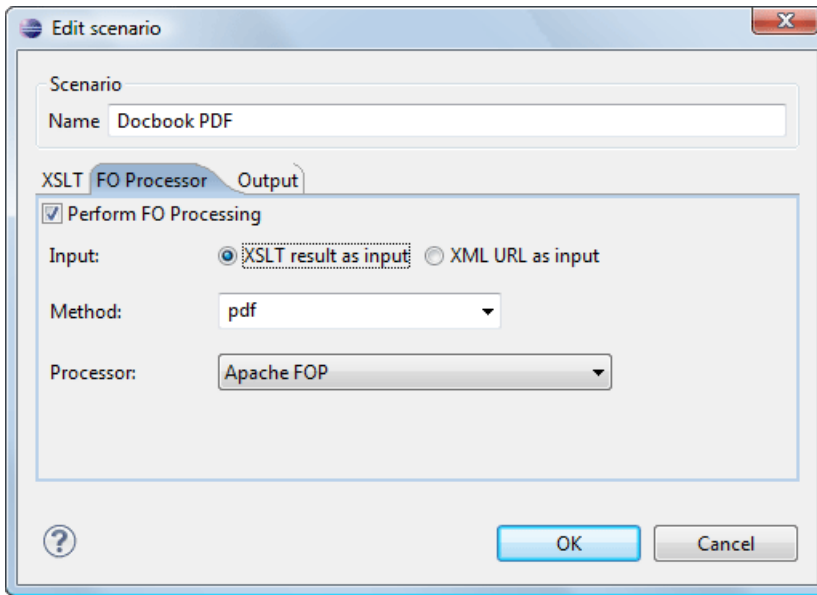


Figure 126: The Configure Transformation Dialog - FO Processor Tab

- **Perform FO Processing** - Enables or disables applying an FO processor (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.
- **XSLT result as input** - The FO processor is applied to the result of the XSLT transformation defined on the XSLT tab of the dialog.
- **XML URL as input** - The FO processor is applied to the input XML file.
- **Method** - The output format of the FO processing. Available options depend on the selected processor type.
- **Processor** - The FO processor, which can be the built-in Apache FOP processor or an *external processor*.

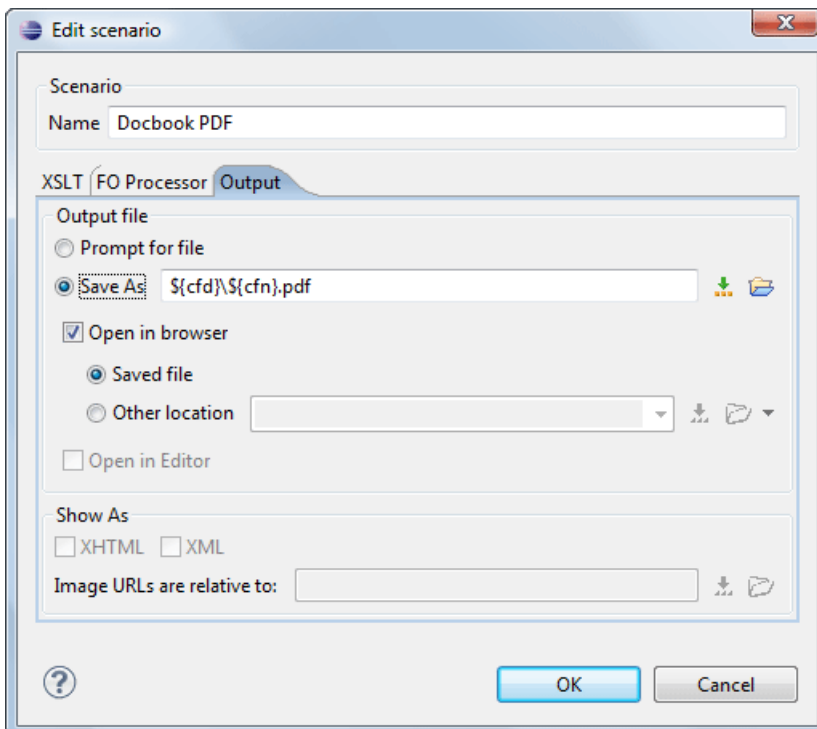


Figure 127: The Configure Transformation Dialog - Output Tab

- **Prompt for file** - At the end of the transformation a file browser dialog will be displayed for specifying the path and name of the file which will store the transformation result.
- **Save As** - The path of the file where the transformation result will be stored. The path can include *special Oxygen XML Developer plugin editor variables* or *custom editor variables*.
- **Open in Browser/System Application** - If enabled, Oxygen XML Developer plugin opens automatically the transformation result in a system application associated with the type of that result (HTML/XHTML, PDF, text) file.
 - 👉 **Note:** If you already set the **Default Internet browser** option in the **Global** preferences page, it will take precedence over the default system application settings.
 - 👉 **Note:** Go to **Window > Preferences > General > Web Browser** to set the web browser that will be used for displaying HTML/XHTML pages.
- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Developer plugin should open automatically at the end of the transformation the file specified in the **Save As** text field.
- **Other location** - When **Open in System Application** is selected, this button can be used to specify that Oxygen XML Developer plugin should not open the file specified in the **Save As** text field, it should open the file specified in the text field of the **Other location** radio button. The file path can include *special Oxygen XML Developer plugin editor variables* or *custom editor variable*.
- **Open in editor** - When this is checked the transformation result set in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type: if the result is an XML file it is opened with the built-in XML editor, if it is an XSL-FO file it is opened with the built-in FO editor, etc.
- **Show As XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked Oxygen XML Developer plugin will display 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 will result in a very long time necessary for rendering the transformation result in the XHTML result viewer panel. This drawback appears due to the built-in Java XHTML browser implementation. In this situations if you wish to see the XHTML result of the transformation you should use an external browser by checking the **Open in browser** checkbox.
- **Show As XML** - If this is checked Oxygen XML Developer plugin will display the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlight* specific for XML documents.
- **Image URLs are relative to** - If **Show As XHTML** is checked this text field specifies the path used to resolve image paths contained in the transformation result.

XSLT Stylesheet Parameters

The global parameters of the XSLT stylesheet used in the transformation scenario are configured from the dialog available from the **Parameters** button of the **Configure Transformation** dialog:

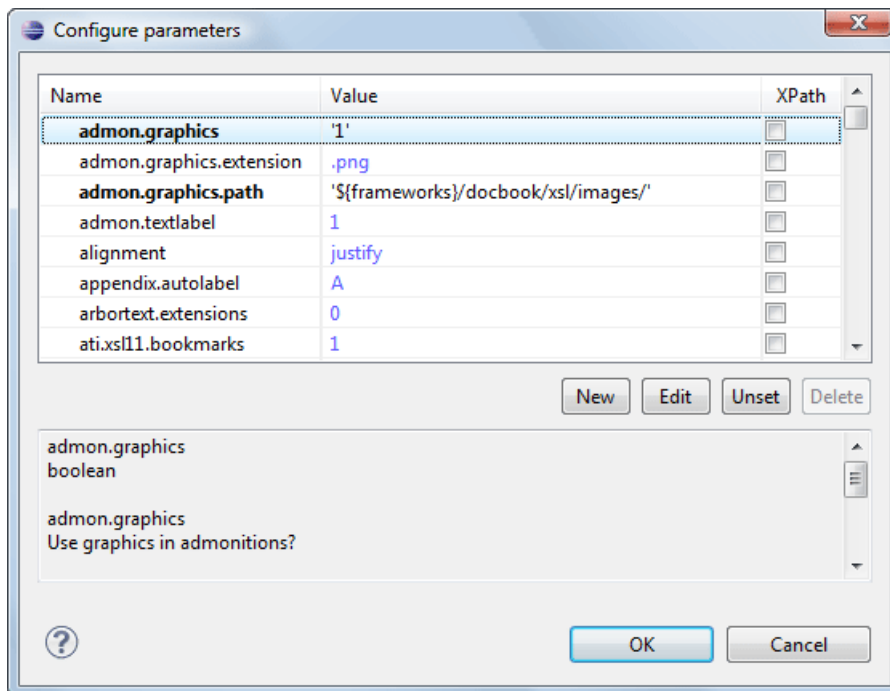


Figure 128: Configure parameters dialog

The table presents all the parameters of the XSLT stylesheet, all imported and included stylesheets and all *additional stylesheets* with their current values. The following font type and color conventions are used:

- blue font values are the defaults collected from the stylesheet;
- black font values and bold font names indicate edited parameters.

If a parameter value was not edited then the table presents its default value. The bottom panel presents the default value of the parameter selected in the table, a description of the parameter if available and the system ID of the stylesheet that declares it.

For example setting the value of a parameter having a declared namespace like:

```
<xsl:param name="p:param" xmlns:p="namespace">default</xsl:param>
```

use the following expression in the **Name** column of the **Parameters** dialog:

```
{namespace}param
```

If the **XPath** column is checked, the parameter value is evaluated as an XPath expression before starting the XSLT transformation.

For example you can use expressions like:

```
doc('test.xml')//entry  
//person[@atr='val']
```



Note:

1. The **doc** function solves the argument relative to the XSL stylesheet location. You can use full paths or editor variables like **\$(cfdu)** (current file directory) to specify other locations: `doc('${cfdu}/test.xml')//*`
2. You cannot use XSLT Functions. Only XPath functions are allowed.

The following actions are available for managing parameters:

- **New** - Adds a new parameter to the list.
- **Edit** - Edits the value of the selected parameter.
- **Unset** - Resets the selected parameter to its default value. Available only for parameters with set values.
- **Delete** - Removes the selected parameter from the list. It is enabled only for parameters added to the list with the **New** button.

The editor variables displayed at the bottom of the dialog (*frameworks*, *home*, *cfid*, etc) can be used in the values of the parameters to make them independent of the location of the XSLT stylesheet or the XML document. To prompt for values at runtime, use the *ask('user-message', param-type, 'default-value' ?)* editor variable.

Additional XSLT Stylesheets



The list of additional XSLT stylesheets can be edited in the dialog opened by the button **Additional XSLT Stylesheets** from the **Configure Transformation** dialog.

- **Add** - Adds a stylesheet in the **Additional XSLT stylesheets** list using a file browser dialog. You can type an editor variable in the file name field of the browser dialog. The name of the stylesheet will be added in the list after the current selection.
- **Remove** - Deletes the selected stylesheet from the **Additional XSLT stylesheets** list.
- **Up** - Moves the selected stylesheet up in the list.
- **Down** - Moves the selected stylesheet down in the list.

The path specified in the URL text field can include *special Oxygen XML Developer plugin editor variables*.

XSLT/XQuery Extensions

The **Libraries** dialog is used to specify the jars and classes containing extension functions called from the XSLT/XQuery file of the current transformation scenario.

An extension function called from the XSLT or XQuery file of the current transformation scenario will be searched in the specified extensions in the order of the list displayed in the dialog. For changing the order of the items the user must select the item that must be moved to other position in the list and press the  up and  down buttons.

Duplicate a Transformation Scenario

Use the following procedure to create a transformation scenario.

1. Go to menu **XML > Configure Transformation Scenario (Alt+Shift+T C) ((Cmd+Alt+T C on Mac OS))** to open the **Configure Transformation** dialog.
2. Click the **Duplicate Scenario** button of the dialog to create a copy of the current scenario.
3. Click in the **Name** field and type a new name.
 - a) You can choose to save the scenarios at project level by setting the **Project Scenarios** setting.
4. Click **OK** or **Transform Now** to save the scenario.

Ant Transformations

An Ant scenario is associated usually with an Ant build script. Oxygen XML Developer plugin runs an Ant 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.

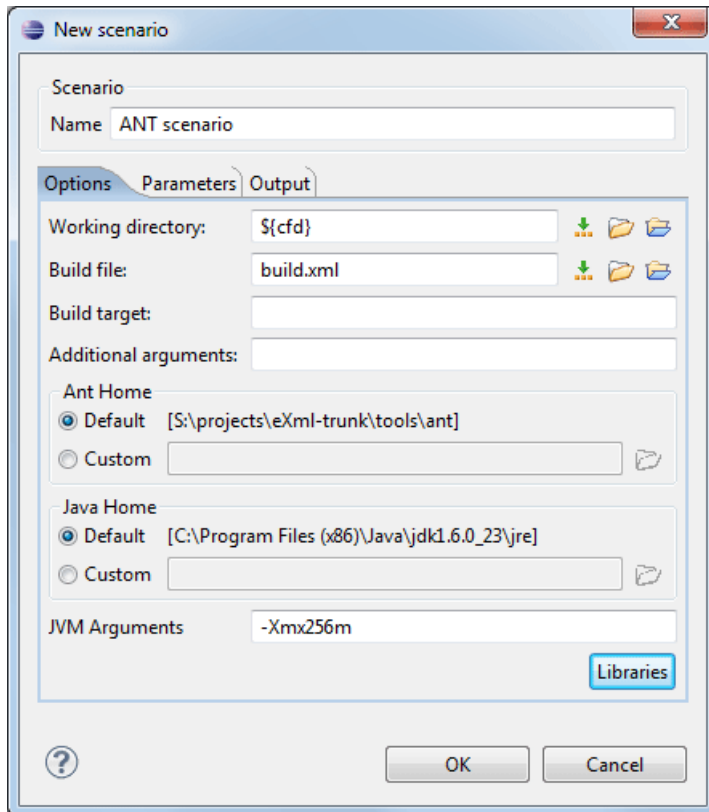


Figure 129: Ant scenario - Options tab

The following parameters are available on the **Options** tab:

- **Working directory** - Path of the current directory of the Ant external process. An *editor variable* can be inserted in this text box using the small green arrow button (↕).
- **Build file** - Ant script file that is the input of the Ant external process. An *editor variable* can be inserted in this text box using the small green arrow button (↕).
- **Build target** - Optionally a build target from the Ant script file can be specified. If no target is specified the Ant target that is specified as default in the Ant script file will be executed.
- **Additional arguments** - Additional command-line arguments to be passed to the Ant transformation (for example -verbose).
- **Ant Home** - Path to the Ant installation to run the transformation. By default it is the Ant installation version 1.8.2 that is bundled with Oxygen XML Developer plugin . A custom Ant installation can also be set.
- **Java Home** - The path to the Java Virtual Machine that runs the Ant transformation. By default it is the Java Virtual Machine that is bundled with Oxygen XML Developer plugin . A custom Java virtual machine can also be set.
- **JVM Arguments** - This parameter allows you to set specific parameters to the Java Virtual Machine used by Ant. By default it is set to -Xmx256m which means the transformation process is allowed to use 256 megabytes of memory. Sometimes, when performing a large DITA map transformation you may want to increase the memory allocated to the Java Virtual Machine from the default value (256 MB) to a higher value, like 512 MB. In this way, you can avoid running out of memory (**OutOfMemoryError**) when running an Ant process.
- **Libraries** - This button allows adding to the classpath of the Ant process any external libraries that are not bundled with Ant (that is they are not built-in Ant libraries).

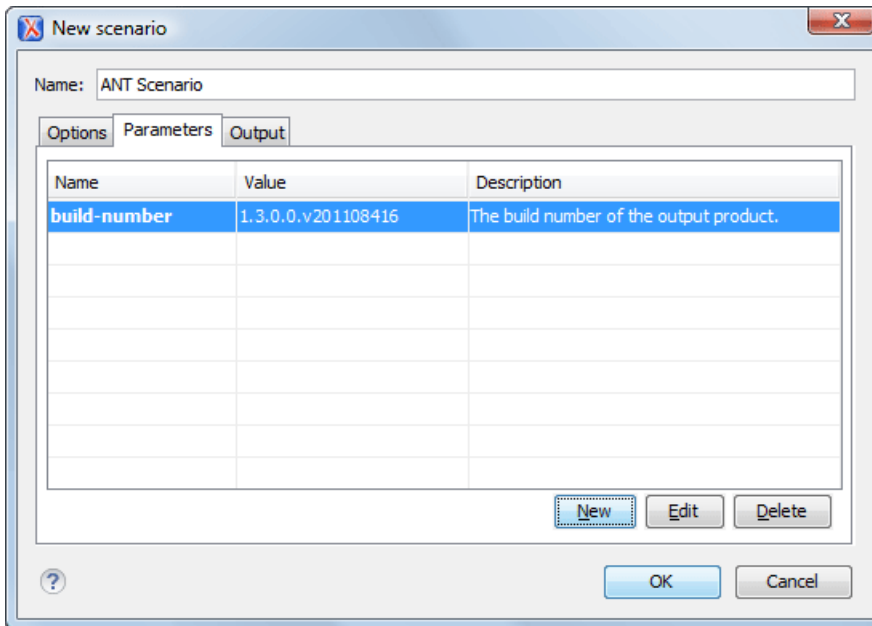


Figure 130: Ant scenario - Parameters tab

On the **Parameters** tab the buttons **New**, **Edit**, and **Delete** can be used to set the parameters which will be accessible as Ant properties in the Ant build script.

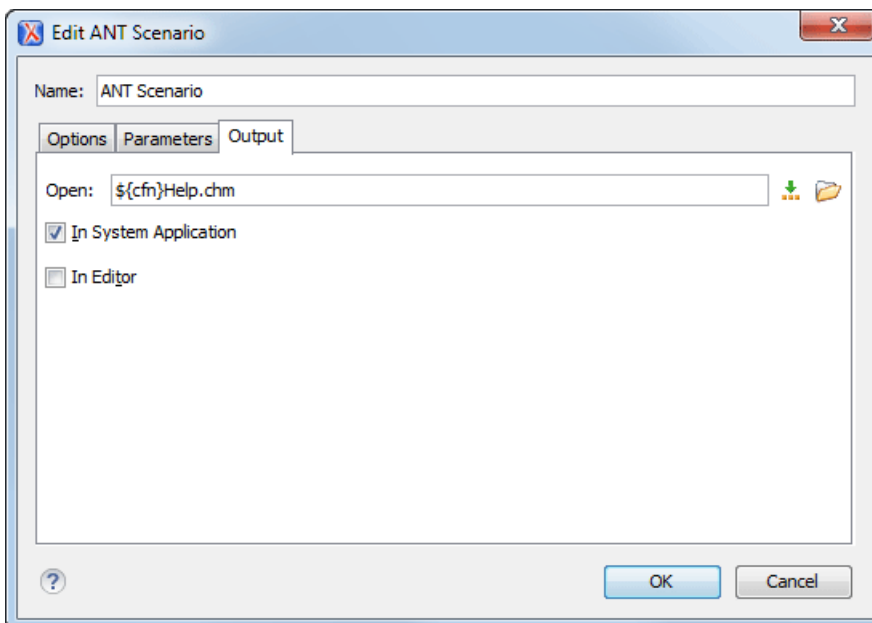


Figure 131: Ant scenario - Output tab

On the **Output** tab the following details can be configured:

- the file to open automatically when the transformation is finished in the **Open** text box, usually the output file of the Ant process; an *editor variable* can be inserted in this text box using the small green arrow button (↕);
- if the file specified in the **Open** text box will be opened in the system application that is set in the operating system as the default application for that type of files (for example the *Acrobat Reader* application for `.pdf` files.);
- if the file specified in the **Open** text box will be opened in Oxygen XML Developer plugin ; for example if it is an `.xml` file it will be opened automatically in *the XML editor panel*, if it is a `.zip` file or an `.epub` file it will be opened in *the Archive Browser view*, etc.;

Transformation Scenarios View

The list of transformation scenarios may be easier to manage for some users as a list presented in a dockable and floating view called **Transformation Scenarios**.

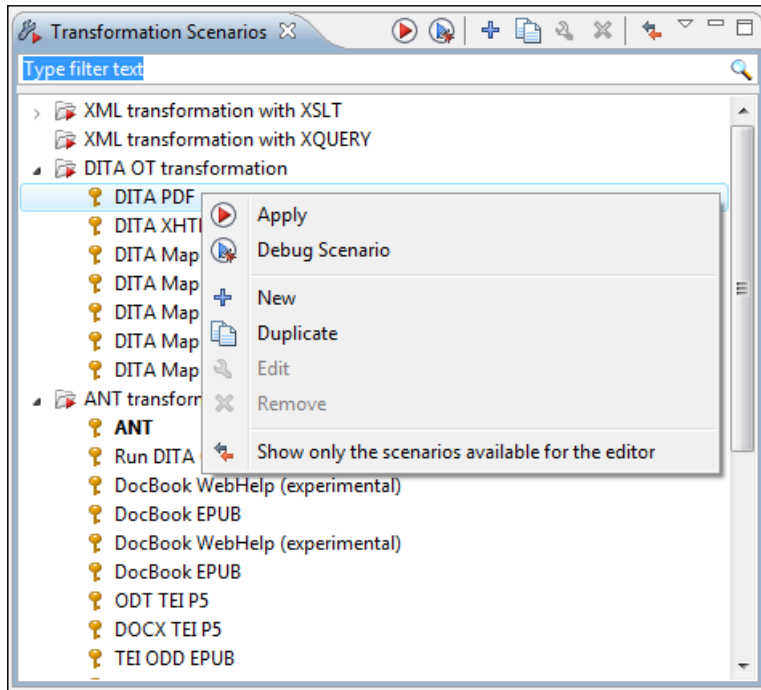










Figure 132: The Scenarios view

The actions available on the right click menu allow the same operations as in *the dialog Configure Transformation Scenario*:

-  **Apply** - Runs the current selected transformation scenario from the list.
-  **Debug Scenario** - Switches to the Debugger perspective and initialize it with the parameters from the scenario: the XML input, the XSLT or XQuery input, the transformation engine, the XSLT parameters.
- **+** **New** - Creates a new transformation scenario.
-  **Duplicate** - Adds a new scenario to the list that is a duplicate of the current scenario. It is useful for creating a new scenario that is the same as an existing one but needs some changes.
-  **Edit** - Opens the dialog for editing the parameters of a transformation scenario.
-  **Remove** - Removes the current scenario from the list. This action is also available on the **Delete** key.
-  **Show all scenarios / Show only the scenarios available for the editor** - A toggle action that switches between two modes:
 - one that lists all transformation scenarios;
 - one that shows only the transformation scenarios specified in the *document type* corresponding to the current editor.

 **Note:** Global scenarios (the scenarios that are not specified in a document type) are always displayed in the view regardless of the state of this action.

 **Note:** When no document is opened in the current editor, the view contains all the transformation scenarios.

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 comes with the following XSLT processors:

- **Xalan 2.7.1** - [Xalan-Java](#) is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements XSL Transformations (XSLT) Version 1.0 and XML Path Language (XPath) Version 1.0.
- **Saxon 6.5.5** - [Saxon 6.5.5](#) is an XSLT processor, which implements the Version 1.0 XSLT and XPath with a number of powerful extensions. This version of Saxon also includes many of the new features that were first defined in the XSLT 1.1 working draft, but for conformance and portability reasons these are not available if the stylesheet header specifies `version="1.0"`.
- **Saxon 9.3.0.5 Home Edition (HE), Professional Edition (PE)** - [Saxon-HE/PE](#) implements the basic conformance level for XSLT 2.0 and XQuery 1.0. The term *basic XSLT 2.0 processor* is defined in the draft XSLT 2.0 specifications: it is a conformance level that requires support for all features of the language other than those that involve schema processing. The HE product remains open source, but removes some of the more advanced features that were present in Saxon-PE.
- **Saxon 9.3.0.5 Enterprise Edition (EE)** - [Saxon EE](#) is the schema-aware edition of Saxon and it is one of the built-in processors of Oxygen 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 specification or according to the W3C XML Schema 1.1 one. This can be [configured in Preferences](#).

Besides the above list Oxygen XML Developer plugin supports the following processors:

- **Xsltproc (libxslt)** - [Libxslt](#) is the XSLT C library developed for the Gnome project. `Libxslt` is based on `libxml2` the XML C library developed for the Gnome project. It also implements most of the EXSLT set of processor-portable extensions functions and some of Saxon's evaluate and expressions extensions. The `libxml2` version included in Oxygen XML 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 into the distribution kit of the stand-alone version for Windows and Mac OS X. Because there are differences between different Linux distributions, on Linux you must install `Libxslt` on your machine as a separate application and set the `PATH` variable to contain the `Xsltproc` executable.

If you do not have the `Libxslt` library already installed, you should copy the following files from Oxygen XML Developer plugin stand-alone installation directory to the root of the `com.oxygenxml.editor_13.2.0` plugin:

- on Windows: `xsltproc.exe`, `zlib1.dll`, `libxslt.dll`, `libxml2.dll`, `libexslt.dll`, `iconv.dll`
- on Linux: `xsltproc`, `libexslt.so.0`, `libxslt.so.1`, `libxml2.so.2`
- on Mac OS X: `xsltproc.mac`, `libexslt`, `libxslt`, `libxml`

The `Xsltproc` processor can be configured from the [XSLTPROC options page](#).



Caution: Known problem: file paths containing spaces are not handled correctly in the LIBXML processor. For example the built-in XML catalog files of the predefined document types (DocBook, TEI, DITA, etc) are not handled by LIBXML if Oxygen XML 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` subdirectory of the installation directory which in this case contains at least a space character.

- **MSXML 3.0/4.0** - [MSXML 3.0/4.0](#) is available only on Windows 2000, Windows NT and Windows XP platforms. It can be used for [transformation](#) and [validation of XSLT stylesheets](#) .

Oxygen XML Developer plugin uses the Microsoft XML parser through its command line tool `msxsl.exe`.

Because `msxsl.exe` is only a wrapper, Microsoft Core XML Services (MSXML) must be installed on the computer otherwise you will get a corresponding warning. You can get the latest Microsoft XML parser from [Microsoft web-site](#)

- **MSXML .NET** - *MSXML .NET* is available only on Windows NT4, Windows 2000 and Windows XP platforms. It can be used for *transformation* and *validation of XSLT stylesheets* .

Oxygen XML 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 other XSLT transformation engines* than *the ones which come with the Oxygen XML Developer plugin distribution*. Such an external engine can be used for XSLT transformations within Editor perspective, and is available in the list of engines in *the dialog for editing transformation scenarios* . However it cannot be used in the XSLT Debugger perspective.

The output messages of a custom processor are displayed in an output view at the bottom of the 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 referred in the message.

Configuring the XSLT Processor Extensions Paths

The Xalan and Saxon processors support the use of extension elements and extension functions. Unlike a literal result element, which the stylesheet simply transfers to the result tree, an extension element performs an action. The extension is usually used because the XSLT stylesheet fails in providing adequate functions to the user for accomplishing a more complex task.

Samples on how to use extensions can be found at:

- for Xalan - <http://xml.apache.org/xalan-j/extensions.html>
- for Saxon 6.5.5 - <http://saxon.sourceforge.net/saxon6.5.5/extensions.html>
- for Saxon 9.3.0.5 - <http://www.saxonica.com/documentation/extensibility/intro.xml>

In order to set an XSLT processor extension (a directory or a jar file), you have to use the *Extensions button of the scenario edit dialog*. The old way of setting an extension (using the parameter `-Dcom.oxygenxml.additional.classpath`) was deprecated and you should use the extension mechanism of the XSLT transformation scenario.

XSL-FO Processors

This section explains how to apply XSL-FO processors when transforming XML documents to various output formats in Oxygen 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. *FOP* is implemented as a Java application that reads a formatting object tree and renders the resulting pages to a specified output.

To include PNG images in the final PDF document you need the *JIMI* or *JAI* libraries. For TIFF images you need the *JAI* library. For PDF images you need the *fop-pdf-images* library. These libraries are not bundled with Oxygen 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:
${oxygenInstallDir}/lib/xalan.jar:${oxygenInstallDir}/lib/
serializer.jar:
${oxygenInstallDir}/lib/resolver.jar:${oxygenInstallDir}/lib/
fop-pdf-images-1.3.jar:
${oxygenInstallDir}/lib/PDFBox-0.7.3.jar"
org.apache.fop.cli.Main -fo ${fo} -${method} ${out}
```

You need to add to the classpath *JimiProClasses.zip* for *JIMI* and *jai_core.jar*, *jai_codec.jar* and *mliwrapper_jai.jar* for *JAI*. For the *JAI* package you can include the directory containing the native libraries (*mli_jai.dll* and *mli_jai_mmx.dll* on Windows) in the *PATH* system variable.

The Mac OS X version of the *JAI* library can be downloaded from <http://www.apple.com/downloads/macosx/apple/java3dandjavaadvancedimagingupdate.html>. In order to use it, install the downloaded package.

Other FO processors can be configured in *the Preferences dialog*.

Add a Font to the Built-in FOP - The Simple Version

If the font that must be set to Apache FOP is one of the fonts that are installed in the operating system you should follow the next steps for creating and setting a FOP configuration file that looks for the font that it needs in the system fonts. It is a simplified version of *the procedure for setting a custom font in Apache FOP*.

1. Register the font in FOP configuration. (not necessary in case of DITA PDF transformations, see next step)
 - a) Create a FOP configuration file that specifies that FOP should look for fonts in the installed fonts of the operating system.

```
<fop version="1.0">
  <renderers>
    <renderer mime="application/pdf">
      <fonts>
        <auto-detect/>
      </font>
    </renderer>
  </renderers>
</fop>
```

```

    </fonts>
  </renderer>
</renderers>
</fop>

```

b) Set the FOP configuration file in **Preferences**.

Go to menu **Options > Preferences > XML > XSLT/FO/XQuery > FO Processors** and enter the path of the FOP configuration file in the **Configuration file for the built-in FOP** text field.

2. Set the font on the document content.

This is done usually with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

- For DocBook documents you can start with the predefined scenario called **DocBook PDF**, [edit the XSLT parameters](#) and set the font name (in our example the font family name is **Arial Unicode MS**) to the parameters `body.font.family` and `title.font.family`.
- For TEI documents you can start with the predefined scenario called **TEI PDF**, [edit the XSLT parameters](#) and set the font name (in our example **Arial Unicode MS**) to the parameters `bodyFont` and `sansFont`.
- For DITA transformations using DITA-OT you should use an IDIOM FOP transformation and modify the following two files:
 - `${frameworks}/dita/DITA-OT/demo/fo/cfg/fo/font-mappings.xml` - the `font-face` element included in each element `physical-font` having the attribute `char-set="default"` must contain the name of the font (**Arial Unicode MS** in our example)
 - `${frameworks}/dita/DITA-OT/demo/fo/fop/conf/fop.xconf` - an element `auto-detect` must be inserted in the element `fonts` which is inside the element `renderer` having the attribute `mime="application/pdf"`:


```

<renderer mime="application/pdf">
  . . .
  <fonts>
    <auto-detect/>
  </fonts>
  . . .
</renderer>

```

Add a Font to the Built-in FOP

If an XML document is transformed to PDF using the built-in Apache FOP processor but it contains some Unicode characters that cannot be rendered by the default PDF fonts then a special font that is capable to render these characters must be configured and embedded in the PDF result.

 **Important:** If the special font that must be set to Apache FOP is installed in the operating system there is a simple way of telling FOP to look for the font. See [the simplified procedure for adding a font to FOP](#).

1. Locate the font.

First, you have to find out the name of a font that has the glyphs for the special characters you used. One font that covers the majority of characters, including Japanese, Cyrillic and Greek, is Arial Unicode MS.

On Windows the fonts are located into the `C:\Windows\Fonts` directory. On Mac they are placed in `/Library/Fonts`. To install a new font on your system is enough to copy it in the `Fonts` directory.

2. Generate a font metrics file from the font file.

- Open a terminal.
- Change the working directory to the `Oxygen XML Developer plugin` install directory.
- Create the following script file in the `Oxygen XML Developer plugin` installation directory.

For Mac OS X and Linux create a file `ttfConvert.sh`:

```
#!/bin/sh
export LIB=lib
export CMD=java -cp
"$LIB/fop.jar:$LIB/avalon-framework-4.2.0.jar:$LIB/xercesImpl.jar"
export CMD=$CMD org.apache.fop.fonts.apps.TTFReader
export FONT_DIR='/Library/Fonts'
$CMD $FONT_DIR/Arialuni.ttf Arialuni.xml
```

For Windows create a file `ttfConvert.bat`:

```
set LIB=lib
set CMD=java -cp
"%LIB%\fop.jar;%LIB%\avalon-framework-4.2.0.jar;%LIB%\xercesImpl.jar"
set CMD=%CMD% org.apache.fop.fonts.apps.TTFReader
set FONT_DIR=C:\Windows\Fonts
%CMD% %FONT_DIR%\Arialuni.ttf Arialuni.xml
```

The paths specified in the file are relative to the Oxygen XML Developer plugin installation directory so if you decide to create it in other directory you have to change the file paths accordingly.

The `FONT_DIR` can be different on your system. Make sure it points to the correct font directory. If the Java executable is not in the `PATH` you will have to specify the full path of the executable.

If the font has bold and italic variants, you will have to convert those too. For this you add two more lines to the script file:

- for Mac OS X and Linux:

```
$CMD $FONT_DIR/Arialuni-Bold.ttf Arialuni-Bold.xml
$CMD $FONT_DIR/Arialuni-Italic.ttf Arialuni-Italic.xml
```

- for Windows:

```
%CMD% %FONT_DIR%\Arialuni-Bold.ttf Arialuni-Bold.xml
%CMD% %FONT_DIR%\Arialuni-Italic.ttf Arialuni-Italic.xml
```

d) Execute the script.

On Linux and Mac OS X you should execute the command `sh ttfConvert.sh` from the command line. On Windows you should run the command `ttfConvert.bat` from the command line or double click on the file `ttfConvert.bat`.

3. Register the font in FOP configuration. (not necessary in case of DITA PDF transformations, see next step)

a) Create a FOP configuration file that specifies the font metrics file for your font.

```
<fop version="1.0">
  <base>./</base>
  <font-base>file:/C:/path/to/FOP/font/metrics/files/</font-base>
  <source-resolution>72</source-resolution>
  <target-resolution>72</target-resolution>
  <default-page-settings height="11in" width="8.26in"/>
  <renderers>
    <renderer mime="application/pdf">
      <filterList>
        <value>flate</value>
      </filterList>
    <fonts>
      <font metrics-url="Arialuni.xml" kerning="yes"
        embed-url="file:/Library/Fonts/Arialuni.ttf">
```

```

        <font-triplet name="Arialuni" style="normal"
            weight="normal"/>
    </font>
</fonts>
</renderer>
</renderers>
</fop>

```

The `embed-url` attribute points to the font file to be embedded. You have to specify it using the URL convention. The `metrics-url` attribute points to the font metrics file with a path relative to the base element. The triplet refers to the unique combination of name, weight, and style (italic) for each variation of the font. In our case is just one triplet, but if the font had variants, you would have to specify one for each variant. Here is an example for Arial Unicode if it had italic and bold variants:

```

<fop version="1.0">
  ...
  <fonts>
    <font metrics-url="Arialuni.xml" kerning="yes"
        embed-url="file:/Library/Fonts/Arialuni.ttf">
      <font-triplet name="Arialuni" style="normal"
        weight="normal"/>
    </font>
    <font metrics-url="Arialuni-Bold.xml" kerning="yes"
        embed-url="file:/Library/Fonts/Arialuni-Bold.ttf">
      <font-triplet name="Arialuni" style="normal"
        weight="bold"/>
    </font>
    <font metrics-url="Arialuni-Italic.xml" kerning="yes"
        embed-url="file:/Library/Fonts/Arialuni-Italic.ttf">
      <font-triplet name="Arialuni" style="italic"
        weight="normal"/>
    </font>
  </fonts>
  ...
</fop>

```

More details about the FOP configuration file are available on <http://xmlgraphics.apache.org/fop/0.93/configuration.html> the FOP website.

b) Set the FOP configuration file in **Preferences**.

Go to menu **Options > Preferences > XML > XSLT/FO/XQuery > FO Processors** and enter the path of the FOP configuration file in the **Configuration file for the built-in FOP** text field.

4. Set the font on the document content.

This is done usually with XSLT stylesheet parameters and depends on the document type processed by the stylesheet. For DocBook documents you can start with the predefined scenario called **DocBook PDF**, *edit the XSLT parameters* and set the font name (in our example **Arialuni**) to the parameters `body.font.family` and `title.font.family`. For TEI documents you can start with the predefined scenario called **TEI PDF**, *edit the XSLT parameters* and set the font name (in our example **Arialuni**) to the parameters `bodyFont` and `sansFont`. For DITA transformations using DITA-OT you should use an IDIOM FOP transformation and modify the following two files:

- `${frameworks}/dita/DITA-OT/demo/fo/cfg/fo/font-mappings.xml` - the `font-face` element included in each element `physical-font` having the attribute `char-set="default"` must contain the name of the font (*Arialuni* in our example)

- `${frameworks}/dita/DITA-OT/demo/fo/fop/conf/fop.xconf` - an element `font` must be inserted in the element `fonts` which is inside the element `renderer` having the attribute `mime="application/pdf"`:

```
<renderer mime="application/pdf">
  . . .
  <font>
    <font metrics-url="Arialuni.xml" kerning="yes"
      embed-url="file:/Library/Fonts/Arialuni.ttf">
      <font-triplet name="Arialuni" style="normal"
        weight="normal"/>
    </font>
  </font>
. . .
</renderer>
```

XProc Transformations

This section explains how to configure and run XProc transformations in Oxygen XML Developer plugin .

XProc Transformation Scenario

A sequence of transformations described by an XProc script can be executed with an XProc transformation scenario. The scenario specifies the following parameters of the transformation:

- the URL of the XProc script
- the XProc engine
- the input ports
- the output ports

On the **XProc** tab of the scenario edit dialog you can select the URL of the XProc script and the XProc engine. The engine can be the built-in *Calabash* engine or a custom engine *configured in the Preferences dialog*.

On the **Inputs** tab of the dialog you can configure each port used in the XProc script for reading input data. Each input port has an assigned name in the XProc script. The XProc engine reads data from the URLs specified in the **URLs** list. The *built-in editor variables* and the *custom editor variables* can be used to specify these URLs.

On the *Parameters* tab you can specify the parameters available on each port.

Each port where the output of the XProc transformation is sent is associated with an URL on the **Outputs** tab of the dialog. The *built-in editor variables* and the *custom editor variables* can be used for specifying this URL.

The 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 and the content of the document(s) that correspond to the selected output port on the right side. If the **Open results in editor** option is selected, the XProc transformation result will be opened automatically in an editor panel. By selecting the **Open in System Application** option, you can specify a file that will be opened at the end of the XProc transformation in the system application associated with that file type.

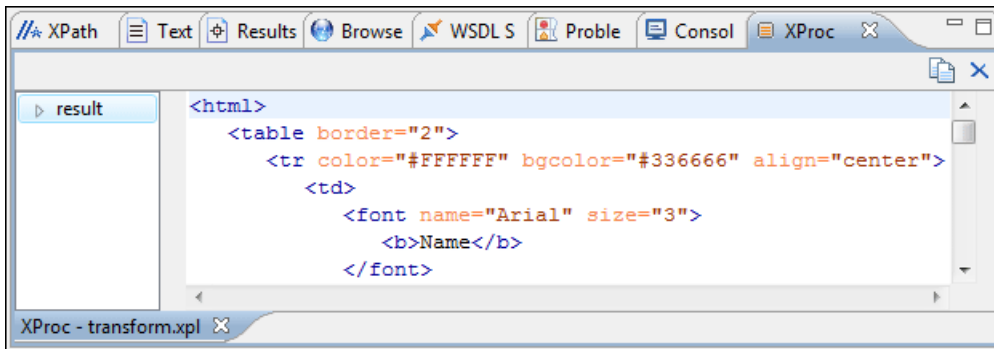


Figure 133: XProc Transformation results view

Integration of an External XProc Engine

The Javadoc documentation of the XProc API is available for download from the application website as a zip file: [xprocAPI.zip](#). In order to create an XProc integration project you can follow the next steps:

1. Take the `oxygen.jar` from `[Oxygen-install-folder]/lib` and put it in the `lib` folder of your project.
2. Implement the `ro.sync.xml.transformer.xproc.api.XProcTransformerInterface` interface.
3. Create a new Java archive (jar) from the classes you created.
4. Create a new `engine.xml` file according with the `engine.dtd` file. The attributes of the `engine` element have the following meanings:
 1. `name` - The name of the XProc engine.
 2. `description` - A short description of the XProc engine.
 3. `class` - The complete name of the class that implements `ro.sync.xml.transformer.xproc.api.XProcTransformerInterface`.
 4. `version` - The version of this integration.
 5. `engineVersion` - The version of the integrated engine.
 6. `vendor` - The name of the vendor / implementor.
 7. `supportsValidation` - `true` if the engine supports validation, `false` otherwise.

The `engine` element has only one child, `runtime`. The `runtime` element contains several `library` elements who's name attribute contains the relative or absolute location of the libraries necessary to run this integration.

5. Create a new folder with the name of the integration in the `[Oxygen-install-folder]/lib/xproc`.
6. Put there the `engine.xml`, and all the libraries necessary to run the new integration.

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 explains possible ways of running an XPath expression on an XML document.

What is XPath

XPath is a language for addressing specific parts of an XML document. XPath, like the Document Object Model (DOM), models an XML document as a tree of nodes. An XPath expression is a mechanism for navigating through and selecting nodes from the XML document. An XPath expression is in a way analogous to a Structured Query Language (SQL) query used to select records from a database.

There are different types of nodes, including element nodes, attribute nodes and text nodes. XPath defines a way to compute a string-value for each type of node.

XPath defines a library of standard functions for working with strings, numbers and boolean expressions.


Some examples:

- `child::*` - Selects all children of the root node.
- `./name` - Selects all elements having the name "name", descendants of the current node.
- `/catalog/cd[price>10.80]` - Selects all the `cd` elements that have a price element with a value larger than 10.80.

To find out more about XPath, the following URL is recommended: <http://www.w3.org/TR/xpath>.

Oxygen's XPath Console

To use XPath effectively requires an understanding of *the XPath Core Function Library*. The Oxygen XML XPath expression field of the current editor toolbar can be used to aid you in XML document development.

In Oxygen XML an XPath 1.0 or XPath 2.0 expression is typed and executed on the current document from the menu **XML > XPath (Ctrl+Shift+X)** (**Cmd+Shift+X on Mac OS**) or from the toolbar button . Both XPath 2.0 basic and XPath 2.0 schema aware expressions can be executed in the XPath console. XPath 2.0 schema aware also takes into account the Saxon EE *XML Schema version* option. The XPath console features a syntax highlight mechanism that allows you to better identify the components of the XPath expression. You can customize the color scheme from the *Syntax Highlight* options page.

The *content completion assistant* that helps entering XPath expressions in attributes of XSLT stylesheets elements is also available in the XPath console. It offers context dependent proposals according with the cursor position in the edited document. The set of XPath functions proposed by the assistant also depends on the XPath version selected from the drop-down menu of the XPath button (1.0 or 2.0).

In the following figure the content completion assistant offers element names from the current document and all XPath 2.0 functions:

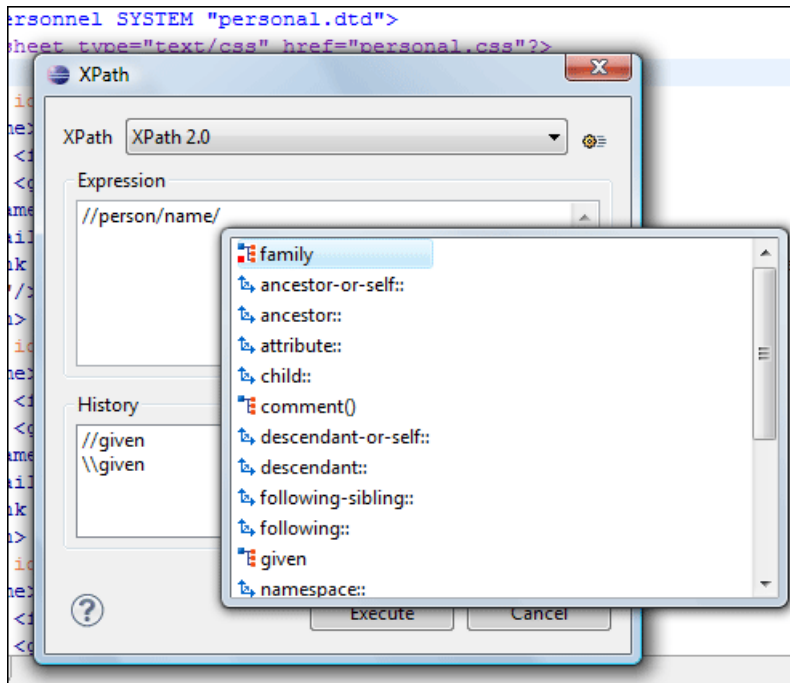


Figure 134: Content Completion in the XPath console

The evaluation of the XPath expression tries to resolve the locations of documents referred in the expression through the *XML catalogs* which are *configured in Preferences* and *the current XInclude preferences*. An example is evaluating the `collection(URIofCollection)` function (XPath 2.0). If you need to resolve the references from the files returned by the `collection()` function with an XML catalog set up in the Oxygen XML Developer plugin preferences you have to specify the class name of the XML catalog enabled parser for parsing these collection files. The class name is `ro.sync.xml.parser.CatalogEnabledXMLReader` and you specify it like this:

```
let $docs := collection(iri-to-uri(
  "file:///D:/temp/test/XQuery-catalog/mydocsdire?recurse=yes;select=*.xml;
  parser=ro.sync.xml.parser.CatalogEnabledXMLReader" ))
```

The results of an XPath query are *displayed in a view* that allows grouping them as a tree. Clicking a record in the result list highlights the matched nodes within the text editor panel with a character level precision. Results are returned in a format that is a valid XPath expression:

```
- /node[value]/node[value]/node[value] -
```

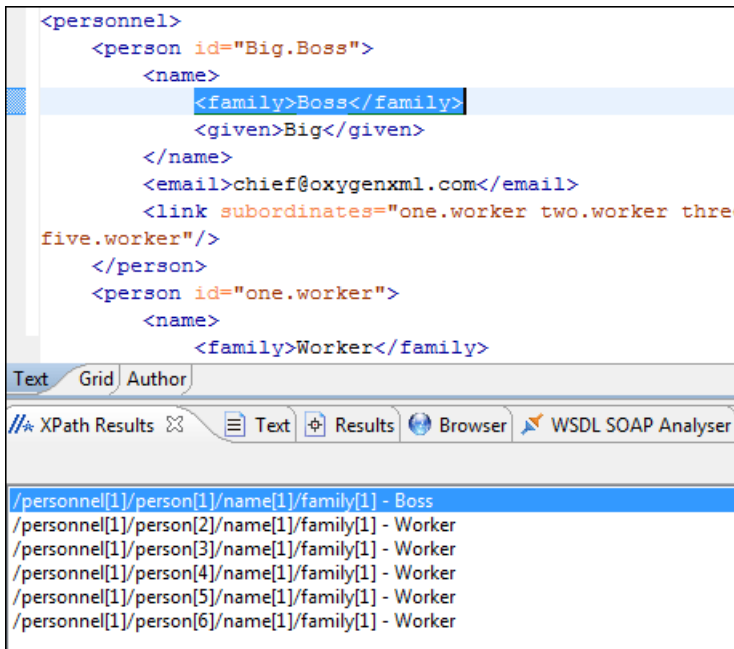


Figure 135: XPath results highlighted in editor panel with character precision

When using the *grid editor*, clicking a result record will highlight the entire node.

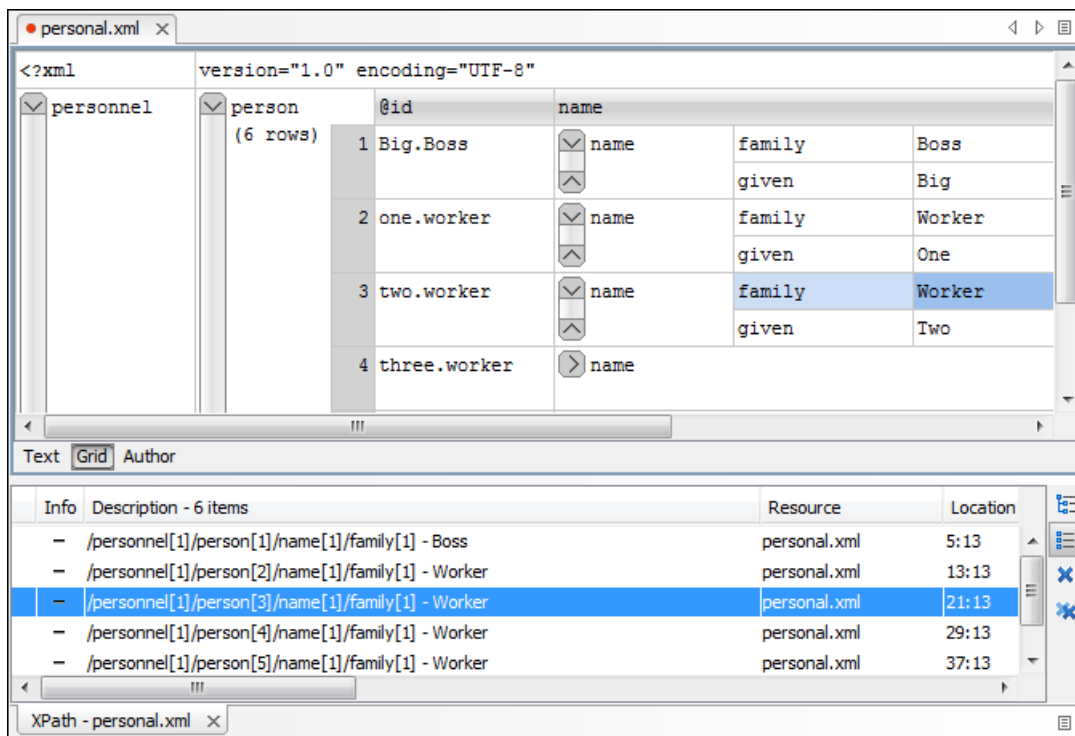


Figure 136: XPath results highlighted in the Grid Editor

The popup menu of the history list of the XPath dialog contains **Remove** and **Remove All** actions for removing expressions from the history list.

XPath Utilization with DocBook DTD

The following examples are taken from a DocBook book based on the DocBook XML DTD. The book contains a number of chapters. To return all the chapter nodes of the book you should enter `//chapter` into the XPath expression field then press **(Enter)**. This will return all the chapter nodes of the DocBook book, in the results view. Each record when clicked will locate and highlight the corresponding chapter element and all its children nodes.

If you want to find all `example` nodes contained in the `sect2` nodes of a DocBook XML document you should use the following XPath expression: `//chapter/sect1/sect2/example`. For each `example` node found in any `sect2` node, a result will be added to the results view.

For example if one of the results of the previous XPath query is:

```
- /chapter[1]/sect1[3]/sect2[7]/example[1]
```

it means that in the edited file, first chapter, third section level 1, seventh section level 2, the found example node is the first in the section.



Important: If the document defines a default namespace then Oxygen XML Developer plugin will bind this namespace to the first free prefix from the list: `default`, `default1`, `default2`, etc. For example if the document defines the default namespace `xmlns="something"` and the prefix `default` is not associated with another namespace then you can match tags without prefix in an XPath expression typed in the XPath console by using the prefix `default`. For example to find all the `level` elements when the root element defines a default namespace you should execute in the XPath console the expression: `//default:level`.

To define default mappings between prefixes that can be used in the XPath console and namespace URIs [go to the XPath Options user preferences panel](#) and enter the mappings in the **Default prefix-namespace mappings** table. The same preferences panel allows the configuration of the default namespace used in XPath 2.0 expressions entered into the XPath toolbar and if different results tabs should be created for each executed XPath query.

To apply an XPath expression relative to the element on which the caret is positioned use the following actions:

- **XML editor contextual menu > XML Document > Copy XPath (Ctrl+Shift+.)** (also available on the context menu of the main editor panel) to copy the XPath expression of the current element or attribute to the clipboard
- **Paste** action of the contextual menu of the XPath console to paste this expression in the console
- add your relative expression in the console and execute the resulting complete expression

The popup menu available in the **Expression** panel of the XPath expressions dialog offers the usual edit actions: **Cut**, **Copy**, **Paste**, **Select All**.

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 a new 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 component 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 9.3.0.5 PE or Saxon 9.3.0.5 EE
- the edited file has a validation scenario associated that use as validation engine Saxon 9.3.0.5 PE or Saxon 9.3.0.5 EE
- the validation engine specified in *Preferences* is Saxon 9.3.0.5 PE or Saxon 9.3.0.5 EE.

If the Saxon namespace (<http://saxon.sf.net>) is mapped to a prefix 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 will display all the XQuery functions from that namespace. When the default namespace is mapped to a prefix the XQuery functions from this namespace offered by content completion are also prefixed, only the function name being used otherwise.

The content completion popup window presents all the variables and functions from both the edited XQuery file and its imports.

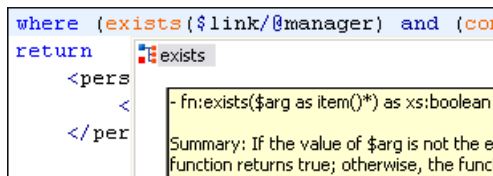


Figure 137: XQuery Content Completion

XQuery Outline View

The XQuery document structure is presented in the **XQuery Outline** view. The outline tree presents the list of all the components (namespaces, imports, variables, and functions) from both the edited XQuery file and its imports. It allows quick access to a component by knowing its name. It can be opened from the **Window** > **Show View** > **Other** > **oXygen** > **Outline** menu action.

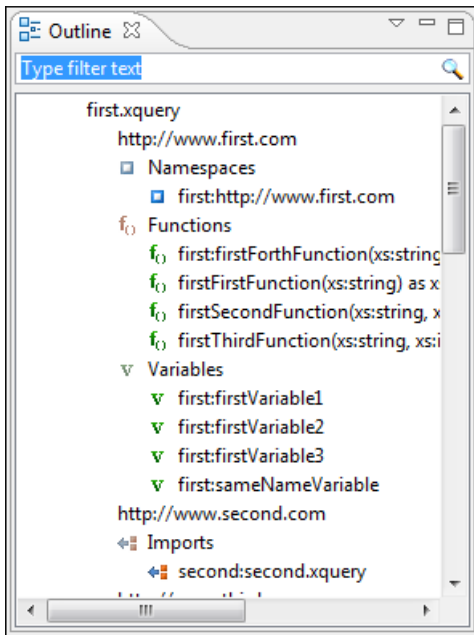


Figure 138: XQuery Outline View

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 local components** - Displays components defined in the current file only.
- **Group by location/namespace/type** - Allows you to group the components by location, namespace, and type. When grouping by namespace, the main XQuery module namespace is 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

A node can be dragged and dropped in the editor area for quickly inserting XQuery expressions.

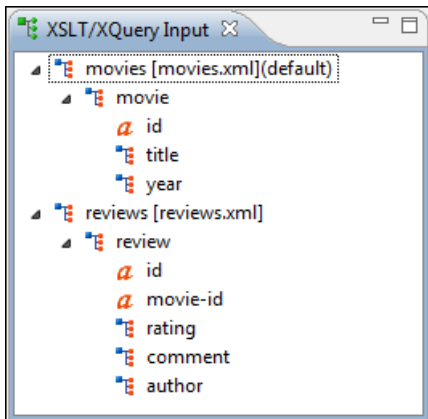


Figure 139: XQuery Input view

Create FLWOR by drag and drop

For the following XML documents:

```
<movies>
  <movie id="1">
    <title>The Green Mile</title>
    <year>1999</year>
  </movie>
  <movie id="2">
    <title>Taxi Driver</title>
    <year>1976</year>
  </movie>
</movies>
```

and

```
<reviews>
  <review id="100" movie-id="1">
    <rating>5</rating>
    <comment>It is made after a great Stephen King
book.
    </comment>
    <author>Paul</author>
  </review>
  <review id="101" movie-id="1">
    <rating>3</rating>
    <comment>Tom Hanks does a really nice
acting.</comment>
    <author>Beatrice</author>
  </review>
  <review id="104" movie-id="2">
    <rating>4</rating>
    <comment>Robert De Niro is my favorite
actor.</comment>
    <author>Maria</author>
  </review>
</reviews>
```

and the following XQuery:

```

let $review := doc("reviews.xml")
for $movie in doc("movies.xml")/movies/movie
let $movie-id := $movie/@id
return
<movie id="{ $movie/@id }">
{ $movie/title }
{ $movie/year }
<maxRating>
{
}
</maxRating>
</movie>

```

if you drag the **review** element and drop it between the braces a popup menu will be displayed.



Figure 140: XQuery Input drag and drop popup menu

Select **FLWOR rating** and the result document will be:

```

37 {
38     for $review in doc("reviews.xml")/reviews/review
39     return
40     where ((compare($rev/rating/text(), string($minRating)) eq 0)
41           and ($rev/@movie-id = $movie/@id))
42     return $rev/author
43 }

```

Figure 141: XQuery Input drag and drop result

XQuery Validation

With Oxygen XML Developer plugin you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.3.0.5 PE processor or the 9.3.0.5 EE, IBM DB2, eXist, Software AG Tamino, Berkeley DB XML or Documentum xDb (X-Hive/DB) 10 if you installed them. 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 syntactically check the expression without executing it. The errors that occurred in the document are presented in the messages view at the bottom of editor window, with a full description message. As with all error messages, if you click on one entry, the line where the error appeared is highlighted.

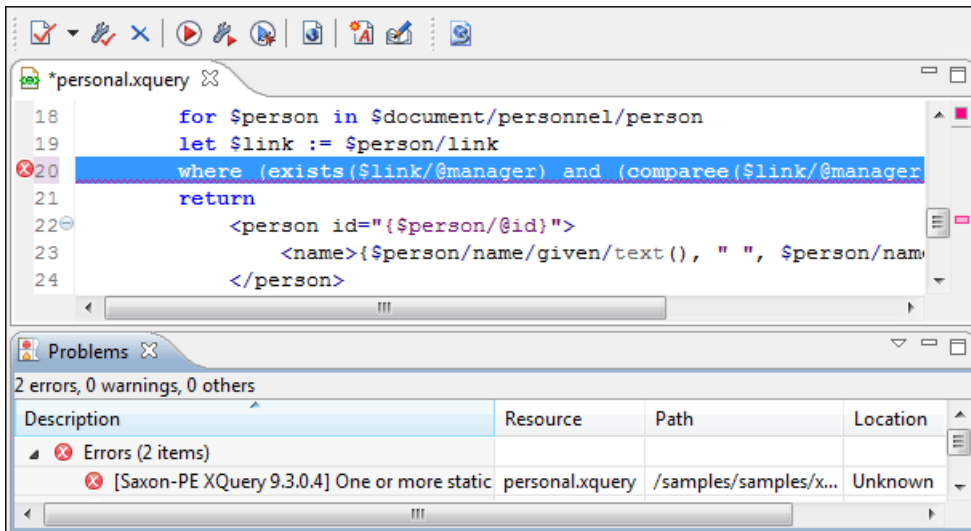


Figure 142: XQuery Validation

Please note that if you choose a processor that doesn't support XQuery validation you will receive a warning when trying to validate.

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 very similar to the XSL stylesheets in the sense they are both 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 dialog **Configure Transformation Scenario** or in *the Scenarios view*. The transformation can be performed on the XML document specified in the **XML URL** field, or, if this field is empty, the documents referred from the query expression. The parameters of XQuery transforms must be set in *the Parameters dialog*. Parameters that are in a namespace must be specified using the qualified name, for example a param parameter in the `http://www.oxygenxml.com/ns` namespace must be set with the name `{http://www.oxygenxml.com/ns}param`.

The transformation uses one of the Saxon 9.3.0.5 HE, Saxon 9.3.0.5 PE, Saxon 9.3.0.5 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 9.3.0.5 EE processor supports also XQuery 1.1 transformations. If *the option Enable XQuery 1.1 support* is enabled Saxon EE runs the XQuery transformations as XQuery 1.1.

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.

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select **XQuery API for Java(XQJ)** in the **Type** combo box.
5. Press the **Add** button to add XQJ API specific files.

You can manage the driver files using the **Add**, **Remove**, **Detect** and **Stop** buttons.

Oxygen XML Developer plugin will detect any implementation of `javax.xml.xquery.XQDataSource` and present it in **Driver class** field.

6. Select the most suited driver in the **Driver class** combo box.
7. Click the **OK** button to finish the data source configuration.

How to Configure an XQJ Connection

The steps for configuring an XQJ connection are the following:


1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for this connection.
4. Select one of the previously configured *XQJ data sources* in the **Data Source** combo box.
5. Fill-in the connection details.

The properties presented in the connection details table are automatically detected depending on the selected data source.

6. Click the **OK** button.

Display Result in Sequence View

The result of an XQuery executed on a database can be very large and sometimes only a part of the full result is needed. For avoiding the long time necessary for fetching the full result the **Present as a sequence** option should be selected in the **Output** tab of the dialog for editing the transformation scenario. This option fetches only the first chunk of the result. Clicking on 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 user option Size limit of Sequence view*. The  **XQuery options** button from the **More results available** label provides a quick access to this option by opening *the XQuery panel of the Preferences dialog* where the option can be modified.

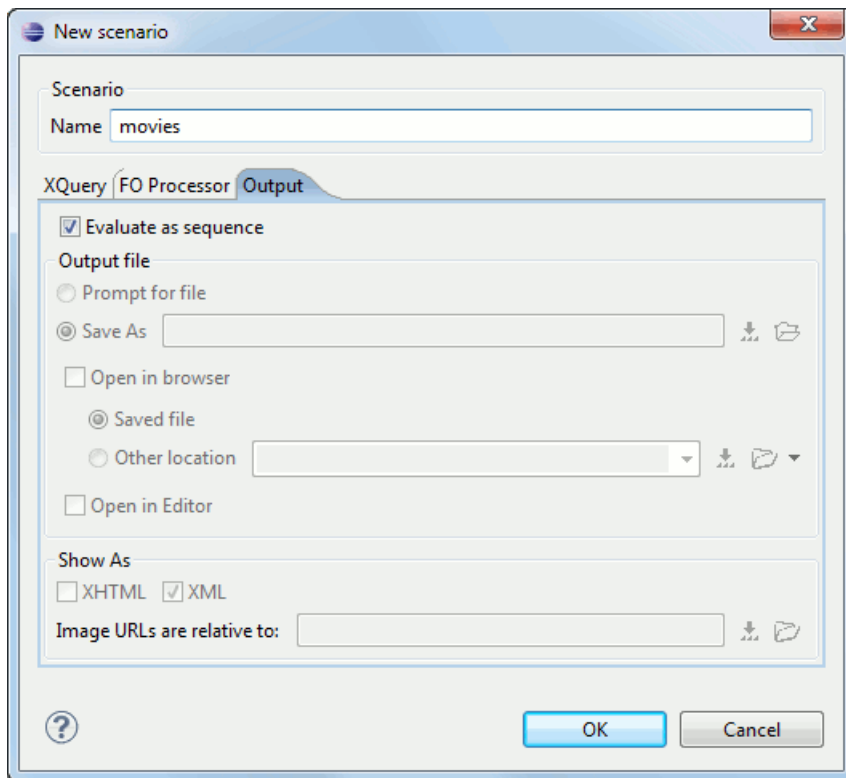


Figure 143: The XQuery transformation result displayed in Sequence view

A chunk of the XQuery transformation result is displayed in the **Sequence** view.

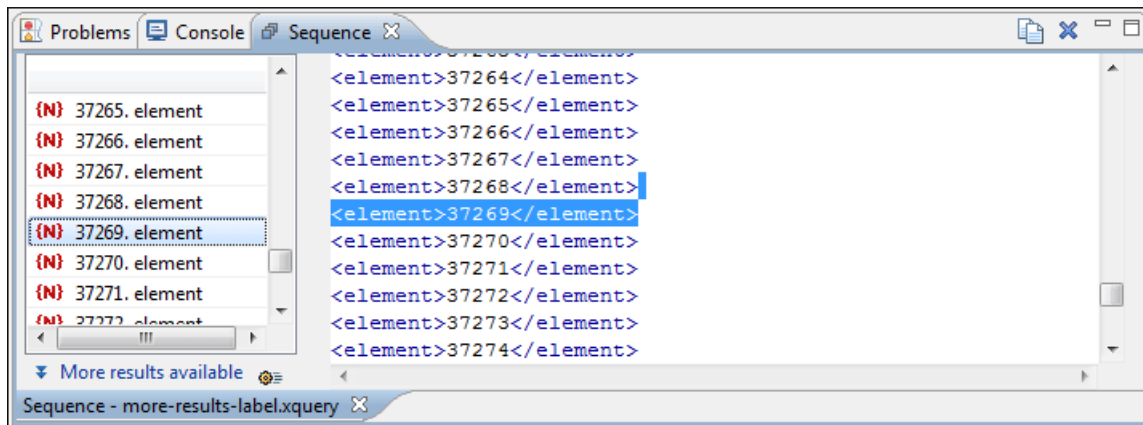


Figure 144: The XQuery transformation result displayed in Sequence view

Advanced Saxon HE/PE/EE Transform Options

The XQuery transformation scenario allows configuring advanced options specific for the Saxon HE (Home Edition) / PE (Professional Edition) / EE (Enterprise Edition) engine. They are the same options as *the ones set in the user preferences* but they are configured as a specific set of transformation options for each transformation scenario. The default values of the options in the transformation scenario are the values *set in the user preferences*. The advanced options specific for Saxon HE / PE / EE are:

- **Use a configuration file** - If checked, the specified Saxon configuration file will be used to determine the Saxon advanced options.
- **Recoverable errors** - Policy for handling recoverable errors in the stylesheet. Allows the user to choose how dynamic errors will be handled. Either one of the following options can be selected:

- recover silently
- recover with warnings
- signal the error and do not attempt recovery
- **Strip whitespaces** - Can have one of the following values:
 - **All** - Strips all whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the XML source document.
 - **Ignorable** - Strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the XML source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
 - **None** - Strips no whitespace before further processing.
- **Optimization level** - Allows optimization to be suppressed in cases where reducing compile time is important, or where optimization gets in the way of debugging, or causes extension functions with side-effects to behave unpredictably.
- **Allow calls on extension functions** - If checked, calling external Java functions is allowed.
- **Validation of the source file** - Available only for Saxon EE. It can have three values:
 - **Schema validation** - This mode requires an XML Schema and enables parsing the source documents with schema-validation enabled.
 - **Lax schema validation** - This mode enables parsing the source documents with schema-validation enabled if an XML Schema is provided.
 - **Disable schema validation** - This means parsing the source documents schema-validation disabled.
- **Validation errors in the results tree treated as warnings** - Available only for Saxon EE. If checked, all validation errors are treated as warnings, otherwise they are treated as fatal.
- **Enable XQuery 1.1 support** - If checked, Saxon EE runs the XQuery transformation with the XQuery 1.1 support.
- **Backup files updated by XQuery ("-backup:(on|off)")** - If checked, backup versions for any XML files updated with XQuery Update will be generated.

Updating XML Documents using XQuery

Using the bundled Saxon 9.3.0.5 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.

Just choose Saxon 9.3.0.5 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.

Using XQuery Update to modify a tag name in an XML file

```
rename node doc("books.xml")//publisher[1]//book[1] as "firstBook"
```

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 stylesheets and XQuery 1.0 documents including complex XPath 2.0 expressions. The interface presents simultaneous views of the source XML document, the XSLT/XQuery document and the result document. As you go step by step through the XSLT/XQuery document the corresponding output is generated step by step, and the corresponding position in the XML file is highlighted. At the same time, special views provide various types of debugging information and events useful to understand the transformation process.

The user is offered a rich set of features for testing and solving XSLT/XQuery problems:

- Support for XSLT 1.0 stylesheets (using Saxon 6.5.5 and Xalan XSLT engines), XSLT 2.0 stylesheets and XPath 2.0 expressions that are included in the stylesheets (using Saxon 9.3.0.5 XSLT engine) and XQuery 1.0 (using Saxon 9.3.0.5 XQuery engine).
- Stepping capabilities: step in, step over, step out, run, run to cursor, run to end, pause, stop.
- Output to source mapping between every line of output and 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 file and XSL file 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*

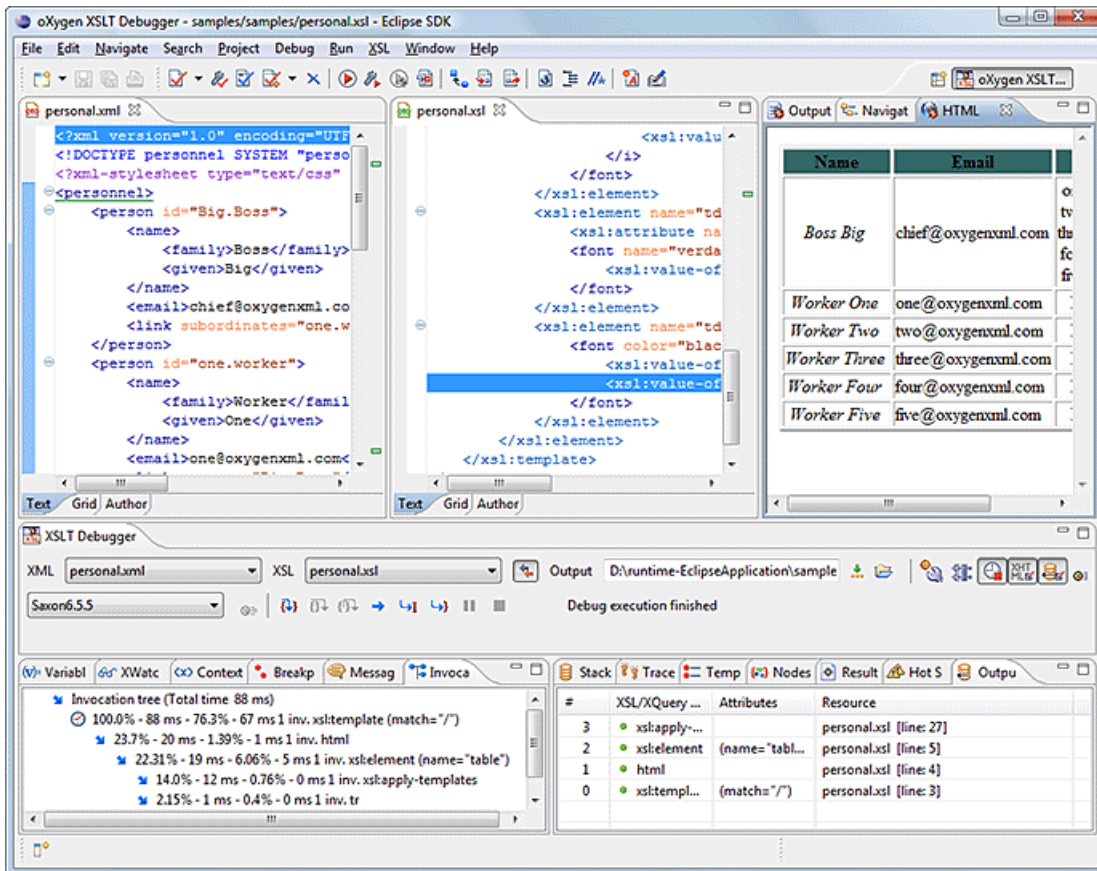


Figure 145: Debugger Mode Interface

XML documents and XSL stylesheets or XQuery documents that were opened in the Editor perspective are automatically sorted into the first two panes. When multiple files of each type are opened, the individual documents and stylesheets are separated using the familiar tab management system of the Editor perspective. Selecting a tab brings the document or stylesheet into focus and enables editing without the need to go back to the Editor perspective.

During debugging, the current execution node is highlighted in both document (XML) and XSLT/XQuery views.

Control Toolbar

The toolbar contains all the actions needed to configure and control the debug process. Below items are described as they appear in the toolbar from left to right.

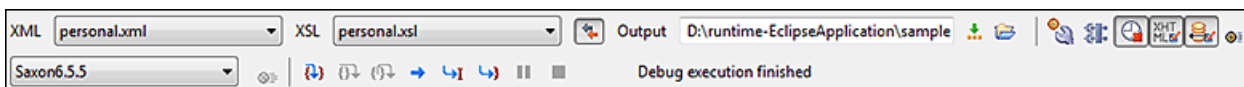























Figure 146: Control Toolbar

- **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 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. Enabled by default.
- **Output selector** - The selection represents the output file specified in the associated transformation scenario.

-  **XSLT / XQuery parameters** - XSLT / XQuery parameters to be used by the transformation.
-  **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. Please 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 9.3.0.5.
-  **Step into (F7)** - Starts the debugging process and runs until the next stylesheet node or the next XPath 2.0 expression step (next transformation step).
-  **Step over (F8 (Alt+F7 on Mac OS))** - Executes the current stylesheet node (including its sub-elements) and goes to the next node in document order (usually the next sibling of the current node) or to the next step of an XPath 2.0 expression.
-  **Step out (Shift + F7)** - Steps out to the parent node (equivalent to the [Step over](#) on the parent).
-  **Run** - Starts the debugging process. The execution of the process is paused when a breakpoint is encountered. (see [breakpoints](#)).
-  **Run to cursor ((Ctrl + F5)** - Starts the debugging process and runs until one of the following conditions occur: the line of cursor is reached, a valid breakpoint is reached or the execution ends.
-  **Run to end ((Alt + F5)** - Runs the transformation until the end, without taking into account enabled breakpoints (if any).
-  **Pause (Shift + F6)** - Interrupts the current transformation. This is useful for long transformations (DocBook for instance) when you want to find out what point the transformation has reached. The transformation can be resumed after.
-  **Stop (F6)** - Ends the transformation process.
- **Show current execution nodes** - Positions the cursor at the current debug context. Possible displayed states:
 - entering () or leaving () an XML execution node;
 - entering () or leaving () an XSL execution node;
 - entering () or leaving () an XPath execution node.

Information View

The information view is comprised of two panes that are used to display various types of information used to understand the transformation process. For each information type there is a corresponding tab. While running a transformation, relevant events are displayed in the various information views. This enables the developer to obtain a clear view of the transformation progress. Using the debug controls developers can easily isolate parts of stylesheet therefore they may be understood and modified. The information types include:

Left side information views

- [Context Node view](#)

- *XWatch view*
- *Breakpoints view*
- *Messages view* (XSLT only)
- *Variables view*

Right side information views

- *Stack view*
- *Trace view*
- *Templates view* (XSLT only)
- *Nodeset view*

Context Node View

The context node is valid only for XSLT debugging session and is a source node corresponding to the XSL expression being evaluated. It is also called the context of execution. The context node implicitly changes as the processor hits various steps (at the point where XPath expressions are evaluated). This node has the same value as evaluating '.' (dot) XPath expression in *XWatch view*. The value of the context node is presented as a tree in the view.

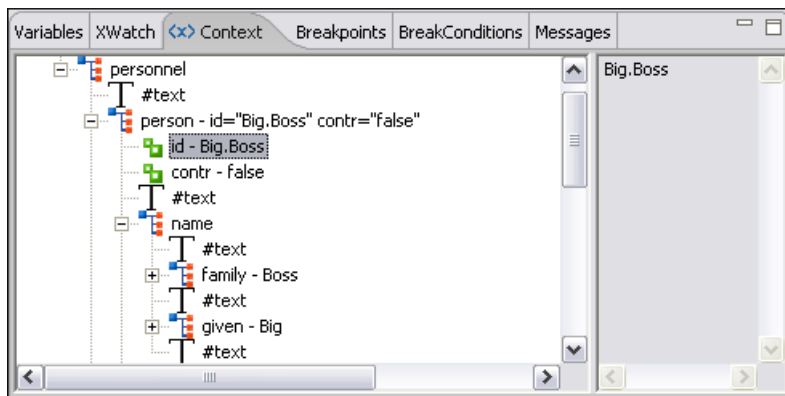


Figure 147: The Context node view

The context node is presented in a tree-like fashion. Nodes from a defined namespace bound to a prefix are displayed using the qualified name. If the namespace is not bound to a prefix, the namespace URI is presented before the node name. The value of the selected attribute or node is shown in the right side panel.

XPath Watch (XWatch) View

This view shows XPath expressions to be evaluated during debugging. Expressions are evaluated dynamically as the processor changes its source context.

When the XPath expression is typed in the **Expression** column the usual content completion assistant is activated and *supports the process of composing the expression just like in the XSLT editor*.

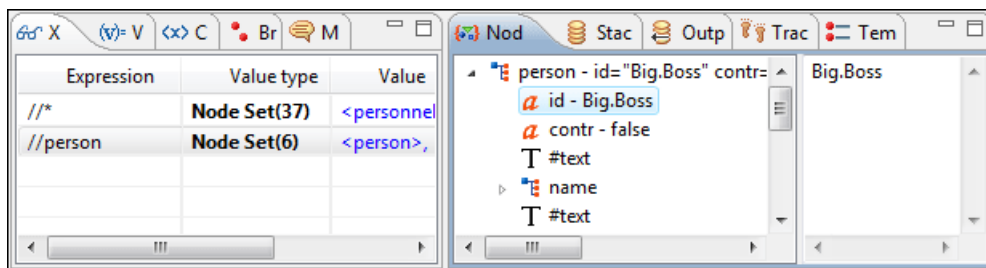



Figure 148: The XPath watch view

Table 1: XWatch columns

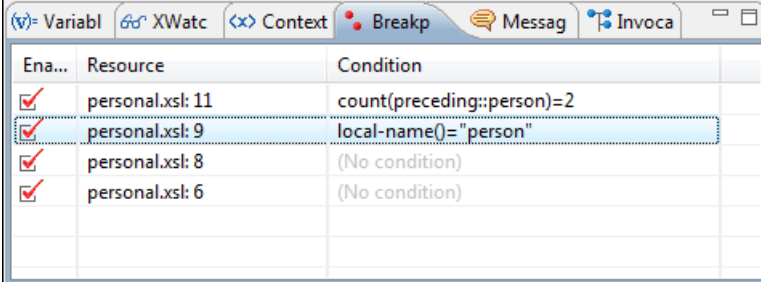
Column	Description
Expression	XPath expression to be evaluated (should be XPath 1.0 or 2.0 compliant).
Value	Result of XPath expression evaluation. Value has a type (see <i>the possible values</i> in the section <i>Variables View</i> on page 238). For Node Set results the number of nodes in the set is shown in parenthesis.

 **Important:** Remarks about working with the XWatch view:

- Expressions referring to variables names are not evaluated. In case of an XPath error, you get an Error line.
- The expression list is not deleted at the end of transformation (it is preserved between debugging sessions).
- To insert a new expression click the last line on the expression column and enter it. As alternative right click and select the **Add** action. Press **(Enter)** on the cell to add and evaluate.
- To delete an expression click on its **Expression** column and delete its content. As alternative right click and select the **Remove** action. Press **(Enter)** on the cell to commit changes.
- If the expression result type is a Node Set you can click on it (**Value** column) and you will see on the right side its value. (see *Nodeset view*).
- The **Copy**, **Add**, **Remove** and **Remove All** actions are offered in every row's contextual menu.

Breakpoints View

This view lists all breakpoints set on opened documents. Once you set a breakpoint it is automatically added in this list. Breakpoints can be set in XSLT/XQuery documents and in XML documents for XSLT debugging sessions. A breakpoint can have an associated break conditions which represent XPath expressions evaluated in the current debugger context. In order to be processed their evaluation result should be a boolean value. A breakpoint with an associated condition stops the execution of the Debugger only if the breakpoint condition is evaluated to **true**.



Ena...	Resource	Condition
<input checked="" type="checkbox"/>	personal.xml: 11	count(preceding::person)=2
<input checked="" type="checkbox"/>	personal.xml: 9	local-name()="person"
<input checked="" type="checkbox"/>	personal.xml: 8	(No condition)
<input checked="" type="checkbox"/>	personal.xml: 6	(No condition)

Figure 149: The Breakpoints View**Table 2: Breakpoints columns**

Column	Description
Enabled	If checked, the current condition is evaluated and taken into account.
Resource	Resource file and number of the line where the breakpoint is set.
Condition	XSLT/XQuery expression to be evaluated during debugging. The expression will be evaluated at every debug step.



Important: Not all set breakpoints are valid. You should check that your breakpoint is valid:

- For example if the breakpoint is set on an empty line or commented line or the line is not reached by the processor (no template to match it, line containing only an end tag), that breakpoint is invalid.
- Clicking a record highlights the breakpoint line into the document.
- The breakpoints list is not deleted at the end of transformation (it is preserved between debugging sessions).

The following actions are available on the table's contextual menu:

- **Go to** - Moves the cursor on the breakpoint's source.
- **Enable** - Enables the breakpoint.
- **Disable** - Disables the breakpoint. A disabled breakpoint will not be evaluated by the Debugger.
- **Add** - Allows you to add a new breakpoint and breakpoint condition.
- **Edit** - Allows you to edit an existing breakpoint.
- **Remove** - Deletes the selected breakpoint.
- **Enable all** - Enables all breakpoints.
- **Disable all** - Disables all breakpoints.
- **Remove all** - Removes all breakpoints.

Messages View

`xsl:message` instructions are one way to signal special situations encountered during transformation as well as a raw way of doing the debugging. This view is available only for XSLT debugging sessions and shows all `xsl:message` calls executed by the XSLT processor during transformation.

Message	Terminate	Resource
Message 1	no	personal.xml [line: 8]
Message 2	no	personal.xml [line: 12]
Message 3	no	personal.xml [line: 29]

Figure 150: The Messages View

Table 3: Messages columns

Column	Description
Message	Message content.
Terminate	Signals if processor terminates the transformation or not once it encounters the message (yes/no respectively)
Resource	Resource file where <code>xsl:message</code> instruction is defined and the message line number.

The following actions are available in the contextual menu:

- **Go to** - Highlight the XSL fragment that generated the message.
- **Copy Value** - Copies to clipboard message details (system ID, severity info, description, start location, terminate state).



Important: Remarks

- Clicking a record from the table highlights the `xsl:message` declaration line.
- Message table values can be sorted by clicking the corresponding column header. Clicking the column header switches the sorting order between: ascending, descending, no sort.

Stack View

This view shows the current execution stack of both source and XSLT/XQuery nodes. During transformation two stacks are managed: one of source nodes being processed and the other for XSLT/XQuery nodes being processed. Oxygen 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.

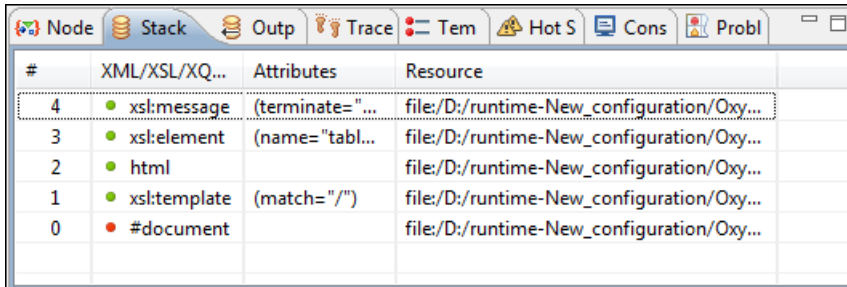


Figure 151: The Stack View

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 4: Stack columns

Column	Description
#	Order number, represents the depth of the node (0 is the stack base).
XML/XSLT/XQuery Node	Node from source or stylesheet document currently being processed. One particular stack node is the document root, noted as #document .
Attributes	Attributes of the node (a list of <code>id="value"</code> pairs).
Resource	Resource file where the node is located.

Important: Remarks:

- Clicking a record from the stack highlights that node's location inside resource.
- Using Saxon, the stylesheet elements are qualified with XSL proxy, while using Xalan you only see their names. (example: `xsl:template` using Saxon and `template` using Xalan).
- Only the Saxon processor shows element attributes.
- The Xalan processor shows also the built-in rules.

Output Mapping Stack View

This view is useful at the end of the transformation and shows the whole stack of XSLT templates/XQuery elements that generated a specific area of the output. It provides *context data in addition to the highlight of the XML element and the XSLT template/XQuery element* available in the editor panel and in the **Stack** view and **Trace** view.

#	XML/XSL/XQuery Node	Attributes	Resource
8	xsl:value-of	(select="email/text()")	file:/D:/runtime-Eclipse
7	font	(name="verdana" (size="3"))	file:/D:/runtime-Eclipse
6	xsl:element	(name="td")	file:/D:/runtime-Eclipse
5	xsl:element	(name="tr")	file:/D:/runtime-Eclipse
4	xsl:template	(match="//person")	file:/D:/runtime-Eclipse
3	xsl:apply-templates		file:/D:/runtime-Eclipse
2	xsl:element	(name="table")	file:/D:/runtime-Eclipse
1	html		file:/D:/runtime-Eclipse
0	xsl:template	(match="/")	file:/D:/runtime-Eclipse

Figure 152: The Output Mapping Stack view

The contextual menu contains one action: **Go to**, which moves the selection in the editor panel to the line containing the XSLT element that is displayed on the selected line from the view.

Table 5: Output Mapping Stack columns

Column	Description
#	The order number in the stack of XSLT templates/XQuery elements. Number 0 corresponds to the bottom of the stack in the status of the XSLT/XQuery processor. The highest number corresponds to the top of the stack.
XSL/XQuery Node	The name of an XSLT template/XQuery element that participated in the generation of the selected output area.
Attributes	The attributes of the XSLT template/XQuery node.
Resource	The name of the file containing the XSLT template/XQuery element.

Important: Remarks:

- Clicking a record highlights that XSLT template definition/XQuery element inside the resource (XSLT stylesheet file/XQuery file).
- Saxon only shows the applied XSLT templates having at least one hit from the processor. Xalan shows all defined XSLT templates, with or without hits.
- The table can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in XSLT rules.

Trace History View

Usually the XSLT/XQuery processors signal the following events during transformation:

- - Entering a source (XML) node.
- - Leaving a source (XML) node.
- - Entering a XSLT/XQuery node.
- - Leaving a XSLT/XQuery node.

The trace history catches all these events, so you can see how the process evolved. The red icon lines denote source nodes while the green icon lines denote XSLT/XQuery nodes.

It is possible to save the element trace in a structured XML document. The action is available on the context menu of the view. In this way you have the possibility to compare the trace results from different debug sessions.

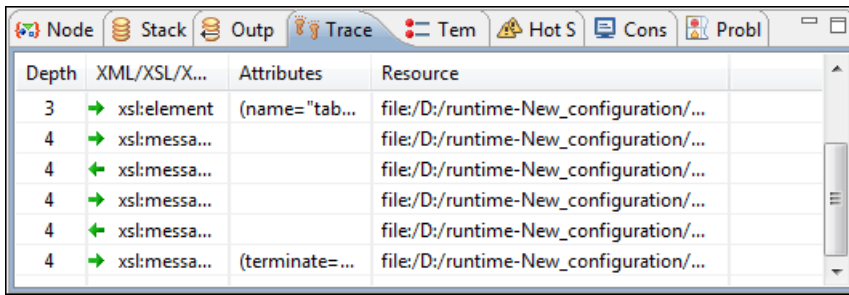


Figure 153: The Trace History View

The contextual menu contains the following actions:

- **Go to** - moves the selection in the editor panel to the line containing the XSLT element or XML element that is displayed on the selected line from the view;
- **Export to XML** - saves the entire trace list into XML format.

Table 6: Trace History columns

Column	Description
Depth	Shows you how deep the node is nested in the XML or stylesheet structure. The bigger the number, the more nested the node is. A depth 0 node is the document root.
XML/XSLT/XQuery Node	Represents the node from the processed source or stylesheet document. One particular node is the document root, noted as #document. Every node is preceded by an arrow that represents what action was performed on it (entering or leaving the node).
Attributes	Attributes of the node (a list of id= "value" pairs).
Resource	Resource file where the node is located.

Important: Remarks:

- Clicking a record highlights that node's location inside the resource.
- Only the Saxon processor shows the element attributes.
- The Xalan processor shows also the built-in rules.

Templates View

The `xsl:template` is the basic element for stylesheets transformation. This view is only available during XSLT debugging sessions and shows all `xsl:template` instructions used by the transformation. By seeing the number of hits for each of the templates you get an idea of the stylesheet coverage by template rules with respect to the input source.

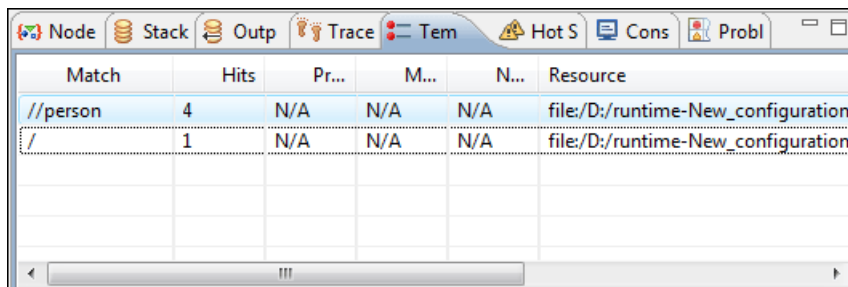


Figure 154: The Templates view

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 7: Templates columns

Column	Description
Match	The <code>match</code> attribute of the <code>xsl:template</code> .
Hits	The number of hits for the <code>xsl:template</code> . Shows how many times the XSLT processor used this particular template.
Priority	The template priority as established by XSLT processor.
Mode	The <code>mode</code> attribute of the <code>xsl:template</code> .
Name	The <code>name</code> attribute of the <code>xsl:template</code> .
Resource	The resource file where the template is located.

Important: Remarks:

- Clicking a record highlights that template definition inside the resource.
- Saxon only shows the applied templates having at least one hit from the processor. Xalan shows all defined templates, with or without hits.
- Template table values can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in rules.

Node Set View

This view is always used in relation with *The Variables view* and *the XWatch view*. It shows an XSLT node set value in a tree form. The node set view is updated as response to the following events:

- You click a variable having a node set value in one of the above 2 views.
- You click a tree fragment in one of the above 2 views.
- You click an XPath expression evaluated to a node set in one of the above 2 views.

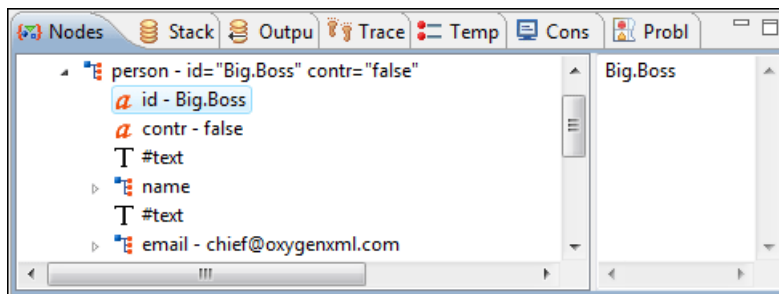


Figure 155: The Node Set view

The nodes / values set is presented in a tree-like fashion. Nodes from a defined namespace bound to a prefix are displayed using the qualified name. If the namespace is not bound to a prefix the namespace URI is presented before the node name. The value of the selected attribute or node is shown in the right side panel.

Important: Remarks:

- In case of longer values in the right side panel the interface shows three suspension points (...) at the end. A more detailed value is available as tooltip.
- Clicking a record highlights the location of that node into the source or stylesheet view.

Variables View

Variables and parameters play an important role during an XSLT/XQuery transformation. Oxygen XML Developer plugin uses the following icons to differentiate variables and parameters:

- **V** - Global variable.
- **{V}** - Local variable.
- **P** - Global parameter.
- **{P}** - Local parameter.

The following value types are available:

- **Boolean**
- **String**
- **Date** - XSLT 2.0 only.
- **Number**
- **Set**
- **Object**
- **Fragment** - Tree fragment.
- **Any**
- **Undefined** - The value was not yet set, or it is not accessible.

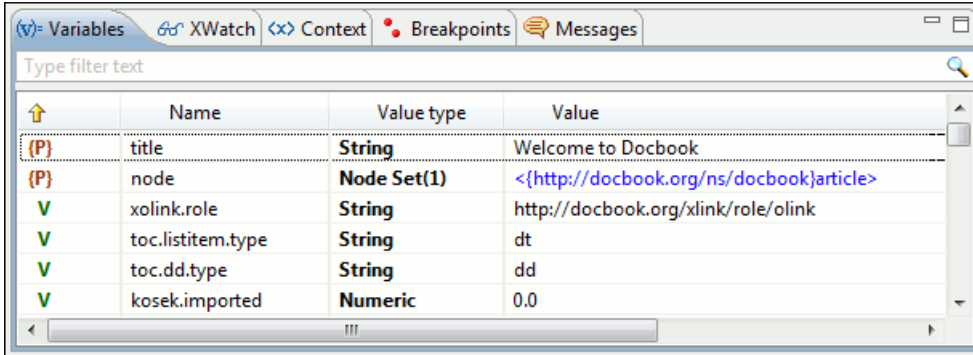
Note:

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.



	Name	Value type	Value
{P}	title	String	Welcome to Docbook
{P}	node	Node Set(1)	<{http://docbook.org/ns/docbook}article>
V	xolink.role	String	http://docbook.org/xlink/role/olink
V	toc.listitem.type	String	dt
V	toc.dd.type	String	dd
V	kosek.imported	Numeric	0.0

Figure 156: The Variables View

Table 8: Variables columns

Column	Description
Name	Name of variable / parameter.
Value type	Type of variable/parameter.
Value	Current value of variable / parameter.

The value of a variable (the **Value** column) can be copied to the clipboard for pasting it to other editor area with the action **Copy value** from the contextual menu of the table from the view. This is useful in case of long and complex values which are not easy to remember by looking at them once.

 **Important:** Remarks:

- Local variables and parameters are the first entries presented in the table.
- Clicking a record highlights the variable definition line.
- Variable values could differ depending on the transformation engine used or stylesheet version set.
- If the value of the variable is a node set or a tree fragment, clicking on it causes the *Node Set view* to be shown with the corresponding set of values.
- Variable table values can be sorted by clicking the corresponding column header. Clicking the column header switches between the orders: ascending, descending, no sort.

Multiple Output Documents in XSLT 2.0


For XSLT 2.0 stylesheets that store the output in more than one file by using the `xsl:result-document` instruction the content of the file created in this way is displayed dynamically while the transformation is running in an output view. There is one view for each `xsl:result-document` instruction so that the output of different instructions is not mixed but is presented in different views.









Working with the XSLT / XQuery Debugger

This section describes how to work with the debugger in the most common use cases.

Steps in a Typical Debug Process

To debug a stylesheet or XQuery document follow the procedure:

1. Open the source XML document and the XSLT/XQuery document.
2. If you are in the 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**
-  **Run to end**
-  **Pause**
-  **Stop**

8. Examine the information in the Information views to find the bug in the transformation process.

You may find [the procedure for determining the XSLT template/XQuery element that generated an output section](#) useful for fixing bugs in the transformation.

Using Breakpoints

The Oxygen XML 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 persistence between work sessions, they are saved at project level. You can set maximum 100 breakpoints per project.

Inserting Breakpoints

To insert a breakpoint, follow these steps:

1. Place your cursor on the line where you want the breakpoint to be in the XML source document or the XSLT / XQuery document.

You can set breakpoints on XML source only for XSLT debugging sessions.

2. Click the left side stripe of the editor panel.

Removing Breakpoints

Only one action must be executed for removing a breakpoint:


Click the breakpoint icon on the left side stripe of the editor panel. As alternative go to menu **Edit > Breakpoints > Remove All**.

Determining What XSLT / XQuery Expression Generated Particular Output


In order to quickly spot the XSLT templates or XQuery expressions with problems it is important to know what XSLT template in the XSLT stylesheet or XQuery expression in the XQuery document and what element in the source XML document generated a specified area in the output.

Some of the debugging capabilities, for example *Step in* can be used for this purpose. Using *Step in* you can see how output is generated and link it with the XSLT/XQuery element being executed in the current source context. However, this can become difficult on complex XSLT stylesheets or XQuery documents that generate a large output.

Output to source mapping is a powerful feature that makes this output to source mapping persistent. You can click on the text from the **Text** output view or **XHTML** output view and the editor will select the XML source context and the XSLT template/XQuery element that generated the text. Also inspecting the whole stack of XSLT templates/XQuery elements that determined the state of the XSLT/XQuery processor at the moment of generating the specified output area speeds up the debugging process.

1. Switch to the 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  **Debug scenario** . This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer

engine, the XSLT parameters, the transformer extensions, etc) will be saved back in the scenario when exiting from the Debugger perspective.

2. Select the source XML document in the XML source selector of *the Control toolbar*. In case of XQuery debugging without an implicit source choose the NONE value.
3. Select the XSLT / XQuery document in the XSLT / XQuery selector of *the Control toolbar*.
4. Select the XSLT / XQuery engine in the XSLT / XQuery engine selector of *the Control toolbar*.
5. Set XSLT / XQuery parameters from the button available on *the Control toolbar*.
6. Apply the XSLT stylesheet or XQuery transformation using the button  **Run to end** available on *the Control toolbar*.
7. Inspect the mapping by clicking a section of the output from the **Output** view of the Oxygen XML Developer plugin XSLT Debugger or Oxygen XML Developer plugin XQuery Debugger perspectives.

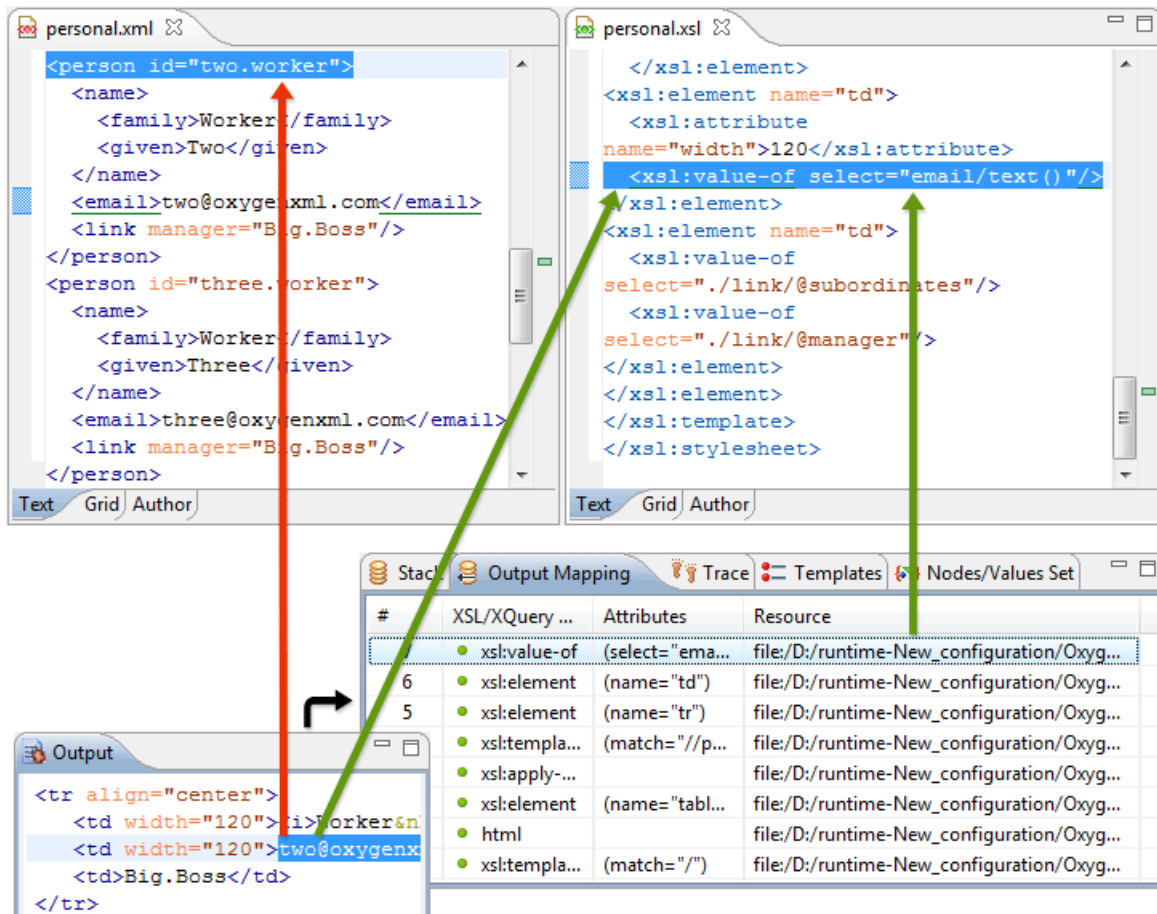


Figure 157: Text Output to Source Mapping

This action will highlight the XSLT / XQuery element and the XML source context. This XSLT template/XQuery element that is highlighted in the XSLT/XQuery editor represents only the top of the stack of XSLT templates/XQuery elements that determined the state of the XSLT/XQuery processor at the moment of generating the clicked output section. In case of complex transformations inspecting the whole stack of XSLT templates/XQuery elements speeds up the debugging process. This stack is available in *the Output Mapping Stack view*.

Debugging Java Extensions

The XSLT/XQuery debugger does not step into Java classes that are configured as XSLT/XQuery extensions of the transformation. For stepping into Java classes, inspecting variable values and setting breakpoints in Java methods you should set up a Java debug configuration in an IDE like the Eclipse SDK as described below.

1. Create a debug configuration.

- a) Set at least 256 MB as heap memory for the Java virtual machine (recommended 512 MB) by setting the `-Xmx` parameter in the debug configuration, for example `"-Xmx512m"`.
- b) Make sure the `[Oxygen-install-folder]/lib/oxygen.jar` file and your Java extension classes are on the Java classpath.

The Java extension classes should be the same classes that were *set as an extension* of the XSLT/XQuery transformation in the Oxygen debugging perspective.

- c) Set the class `ro.sync.exml.Oxygen` as the main Java class of the configuration.

The main Java class `ro.sync.exml.Oxygen` is located in the `oxygen.jar` file.

2. Start the debug configuration.

Now you can set breakpoints and inspect Java variables as in any Java debugging process executed in the selected IDE (Eclipse SDK, etc.).

Supported Processors for XSLT / XQuery Debugging

The following built-in XSLT processors are integrated in the debugger and can be selected in the *Control Toolbar*:

- Saxon 9.3.0.5 HE (Home Edition) - a limited version of the Saxon 9 processor, capable of running XSLT 1.0, XSLT 2.0 basic and XQuery 1.0 transformations, available in both the XSLT debugger and the XQuery one,
- Saxon 9.3.0.5 PE (Professional Edition) - capable of running XSLT 1.0 transformations, XSLT 2.0 basic ones and XQuery 1.0 ones, available in both the XSLT debugger and the XQuery one,
- Saxon 9.3.0.5 EE (Enterprise Edition) - a schema aware processor, capable of running XSLT 1.0 transformations, XSLT 2.0 basic ones, XSLT 2.0 schema aware ones and XQuery 1.0 ones, available in both the XSLT debugger and the XQuery debugger,
- Saxon 6.5.5 - capable of running only XSLT 1.0 transformations, available only in the XSLT debugger,
- Xalan 2.7.1 - capable of running only XSLT 1.0 transformations, available only in the XSLT debugger.

Chapter 10

Profiling XSLT Stylesheets and XQuery Documents

Topics:


- [Overview](#)
- [Viewing Profiling Information](#)
- [Working with XSLT/XQuery Profiler](#)

This chapter explains the user interface and how to use the profiler for finding performance problems in XSLT transformations and XQuery ones.

Overview

Whether you are trying to identify a performance issue that is causing your production XSLT/XQuery transformation to not meet customer expectations or you are trying to proactively identify issues prior to deploying your XSLT/XQuery transformation, using the XSLT/XQuery profiler feature is essential to helping you save time and ultimately ensure a better performing, more scalable XSLT/XQuery transformation.

The XSLT/XQuery profiling feature can use any available XSLT/XQuery processors that could be used for debugging and it is available from the debugging perspective.

Enabling and disabling the profiler is controlled by the  *Profiler button* from the *debugger control toolbar*. The XSLT/XQuery profiler is off by default. This option is not available during a debugger session so you should set it before starting the transformation.

Viewing Profiling Information

This section explains the views that display the profiling data collected by the profiles during the transformation.

Invocation Tree View

This view shows a top-down call tree representing how XSLT instructions or XQuery expressions are processed.

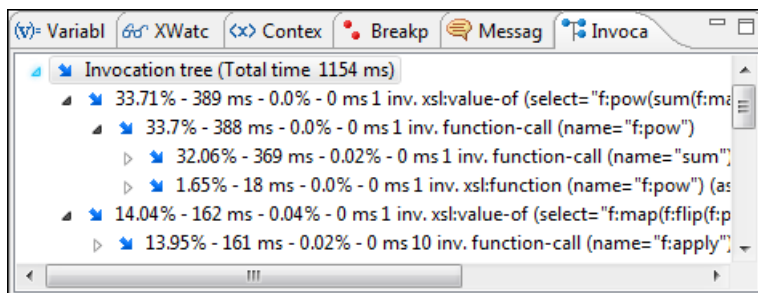




Figure 158: Invocation tree view

The entries in the invocation tree have different meanings which are indicated by the displayed icons:

-  - Points to a call whose inherent time is insignificant compared to its call tree time.
-  - Points to a call whose inherent time is significant compared to its call tree time (greater than 1/3rd of its call tree time).

Every entry in the invocation tree has textual information attached which depends on the [XSLT/XQuery profiler settings](#) :

- A percentage number of total time which is calculated with respect to either the root of the tree or the calling instruction.
- A total time measurement in milliseconds or microseconds. This is the total execution time that includes calls into other instructions.
- A percentage number of inherent time which is calculated with respect to either the root of the tree or the calling instruction.
- An inherent time measurement in milliseconds or microseconds. This is the inherent execution time of the instruction.
- An invocation count which shows how often the instruction has been invoked on this path.
- An instruction name which contains also the attributes description.

Hotspots View

This view shows a list of all instruction calls which lie above the threshold defined in the [XSLT/XQuery profiler settings](#) .

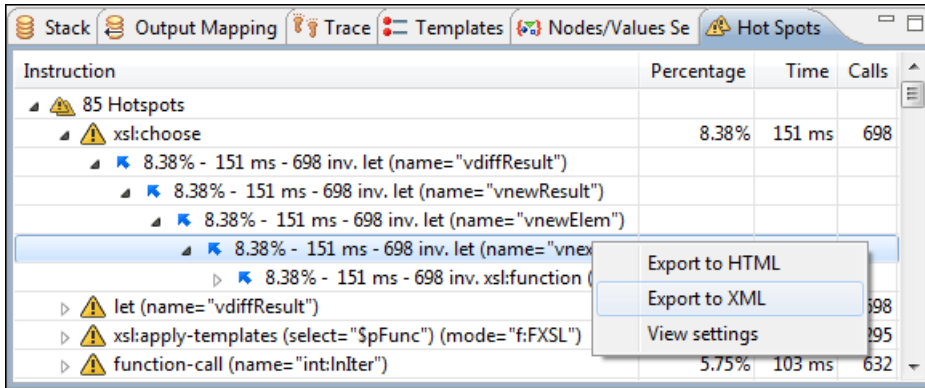



Figure 159: Hotspots View

By opening a hotspot instruction entry, the tree of back-traces leading to that instruction call are calculated and shown.


Every hotspot is described by the values from the following columns:

- The instruction name.
- The inherent time in milliseconds or microseconds of how much time has been spent in the hotspot together with a bar whose length is proportional to this value. All calls into this instruction are summed up regardless of the particular call sequence.
- The invocation count of the hotspot.

If you click on the  handle on the left side of a hotspot, a tree of back-traces will be shown.

Every entry in the backtrace tree has textual information attached to it which depends on the [XSLT/XQuery profiler settings](#):

- A percentage number which is calculated with respect either to the total time or the called instruction.
- A time measured in milliseconds or microseconds of how much time has been contributed to the parent hotspot on this path.
- An invocation count which shows how often the hotspot has been invoked on this path.

 **Note:** This is not the number of invocations of this instruction.

- An instruction name which contains also its attributes.

Working with XSLT/XQuery Profiler

Profiling activity is linked with debugging activity, so the first step in order to profile is to switch to debugging perspective and follow the corresponding procedure for debugging (see [Working with XSLT Debugger](#)).

Immediately after turning the profiler on two new information views are added to the current debugger [information views](#):

- [Invocation tree view](#) on left side
- [Hotspots view](#) on right side

Profiling data is available only after the transformation ends successfully.

Looking to the right side ([Hotspots view](#)), you can immediately spot the time the processor spent in each instruction. As an instruction usually calls other instructions the used time of the called instruction is extracted from the duration time of the caller (the hotspot only presents the inherent time of the instruction).

Looking to the left side ([Invocation tree view](#)), you can examine how style instructions are processed. This result view is also named call-tree, as it represents the order of style processing. The profiling result shows the duration time for each of the style-instruction including the time needed for its called children.

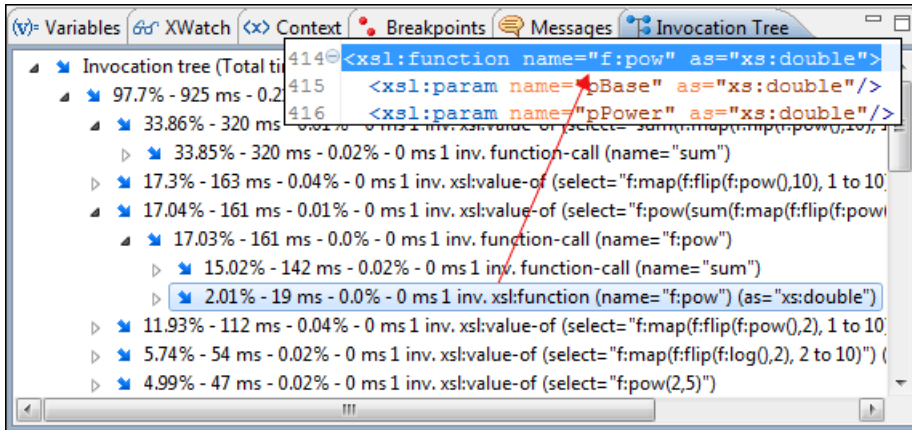


Figure 160: Source backmapping

In any of the above views you can use the backmapping feature in order to find the XSLT stylesheet or XQuery expression definition. Clicking on the selected item cause Oxygen XML 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 `frameworks/profiler/` of the Oxygen XML Developer plugin installation folder) so you can make your own report based on the profiling raw data.

If you like to change the *XSLT/XQuery profiler settings* you should right click on view, use the pop-up menu and choose the corresponding **View settings** entry.



Caution: Profiling exhaustive transformation may run into an OutOfMemory error due to the large amount of information being collected. If this is the case you can close unused projects when running the profiling or use high values for Java VM options `-Xms` and `-Xmx`. If this does not help you can shorten your source xml file and try again.

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 is unmounted.






Attention: The ZIP support needs the IBM437 character set to be properly installed in the Java Runtime Environment in order to be able to navigate / open ZIP archives. If you encounter an error message when expanding a ZIP archive about the JVM that is missing a charset then the JVM used to run Eclipse does not have the character set library properly installed.

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:

-  **New folder...** - Creates a folder as child of the selected folder in the browsed archive.
-  **New file...** - Creates a file as child of the selected folder in the browsed archive.
-  **Add files...** - Adds existing files as children of the selected folder in the browsed archive.






Note:

You can also add files in the archive by dragging them from the file browser or **Project view** and dropping them in the **Archive Browser** view.


-  **Delete** - Deletes the selected resource in the browsed archive.
-  **Archive Options...** - Opens the [Archive preferences page](#).

The following additional operations are available from the **Archive Browser** contextual menu:

-  **Open** - Opens a resource from the archive in the editor.
-  **New folder...** - Creates a folder as child of the selected folder in the browsed archive.
-  **New file...** - Creates a file as child of the selected folder in the browsed archive.
- **Add files...** - Adds existing files as children of the selected folder in the browsed archive.




Note: On Mac OS X, there is also available the **Add file...** action, which allows you to add one file at a time.

-  **Find/Replace in Files** - Allows you to search for and replace specific pieces of text inside the archive.
- **Cut** - Cut the selected archive resource
- **Copy** - Copy the selected archive resource
- **Paste** - Paste a file or folder into the archive



Note: You can add files in the archive by copying the files from the **Project view** and paste them into the **Archive view**.

- **Delete** - Remove a file or folder from archive
- **Copy location** - Copies the URL location of the selected resource.
-  **Refresh** - Refreshes the selected resource.
- **Properties** - Views properties for the selected resource.

Working with EPUB

EPUB is a free and open electronic book standard by the International Digital Publishing Forum (IDPF). It was designed for *reflowable content*, meaning that the text display can be optimized for the particular display device used by the reader of the EPUB-formatted book.

Oxygen XML Developer plugin opens EPUB files in the **Archive Browser** view, exposing all their internal bits and pieces:

- document content (XHTML and image files);
- packaging files;
- container files.

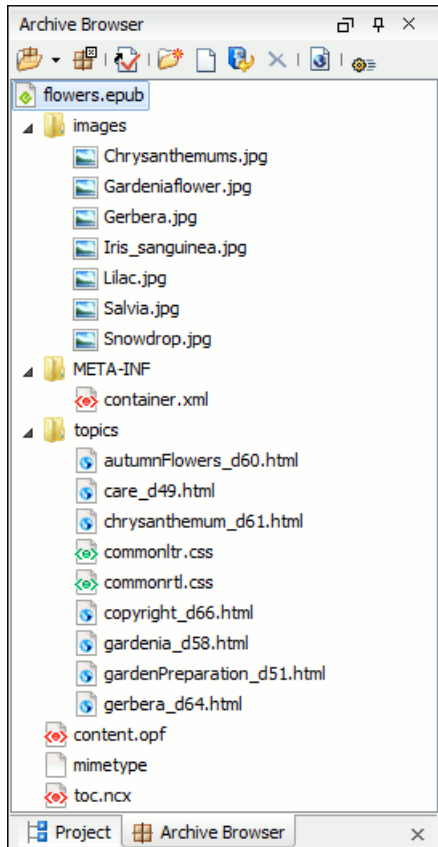






Figure 161: EPUB file displayed in the Archive Browser view

Here you can edit, delete and add files that compose the EPUB structure. To check that the EPUB file you are currently working is valid, invoke the  **Validate and Check for Completeness** action. To perform the operation, Oxygen XML Developer plugin uses the open-source *EpubCheck* validator which detects many types of errors, including OCF container structure, OPF and OPS mark-up, as well as internal reference consistency. All errors found during validation are displayed in a separate tab in the **Errors** view.

 **Note:** Invoke the  **Open in System Application** action to see how the EPUB is rendered in your system default EPUB reader application.

Create an EPUB

To begin writing an EPUB file from scratch, do the following:

1. Select **File > New (Ctrl+N)** or press the  **New** toolbar button.

2. Choose **EPUB Book** template. Click **Create**. Choose the name and location of the file. Click **Save**.
A skeleton EPUB file is saved on disk and open in the **Archive Browser** view.
3. Use the **Archive Browser** view specific actions to edit, add and remove resources from the archive.
4. Use the **Validate and Check for Completeness** action to verify the integrity of the EPUB archive.

Publish to EPUB

Oxygen XML Developer plugin comes with built-in support for publishing Docbook and DITA XML documents directly to EPUB.

1. Open the **Configure Transformation Scenario** dialog and choose a predefined transformation scenario.
2. Start the transformation scenario.

Editing Files From Archives

You can open and edit files directly from an archive using the **Archive Browser** view. When saving the file back to archive, you are prompted to choose if you want the application to make a backup copy of the archive before saving the new content. If you choose **Never ask me again**, you will not be asked again to make backup copies. You can re-enable the dialog pop-up from the [Archive preferences page](#).

Chapter 12

Working with Databases

Topics:

- [Relational Database Support](#)
- [Native XML Database \(NXD\) Support](#)
- [XQuery and Databases](#)
- [WebDAV Connection](#)

XML is a storage and interchange format for structured data and it is supported by all major database systems. Oxygen XML Developer plugin offers the means of managing the interaction with some of the widely used databases, both relational ones and Native XML Databases. By interaction, one should understand browsing, querying, SQL execution support, content editing, importing from databases, generating XML Schema from database structure.

Relational Database Support

Relational databases use a relational model and are based on tables linked by a common key. Oxygen XML Developer plugin offers support for the following relational databases: IBM DB2, JDBC-ODBC Bridge, MySQL, Microsoft SQL Server, Oracle 11g:

- browsing the tables of these types of database in the **Data Source Explorer** view
- executing SQL queries against them
- calling stored procedures with input and output parameters

Configuring Database Data Sources

This section describes the procedures for configuring the data sources for relational databases.

How to Configure an IBM DB2 Data Source

The steps for configuring a data source for connecting to an IBM DB2 server are the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

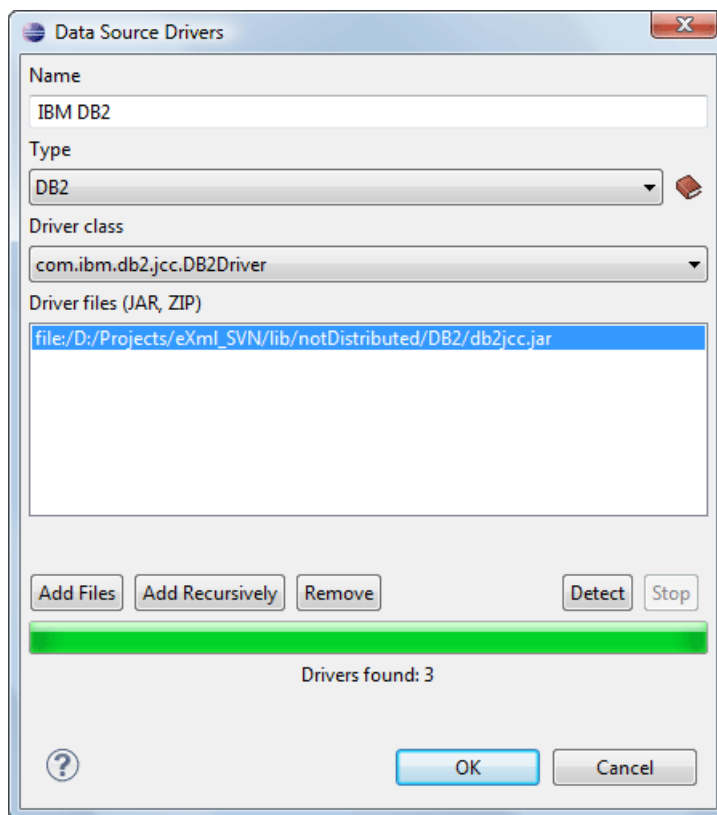


Figure 162: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **DB2** in the driver type combo box.
5. Add the driver files for IBM DB2 using the **Add** button.

The IBM DB2 driver files are:

- db2jcc.jar

- db2jcc_license_cisuz.jar
- db2jcc_license_cu.jar

In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing IBM DB2 databases in Oxygen XML Developer plugin .

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a Microsoft SQL Server Data Source

The steps for configuring a data source for connecting to a Microsoft SQL server are the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

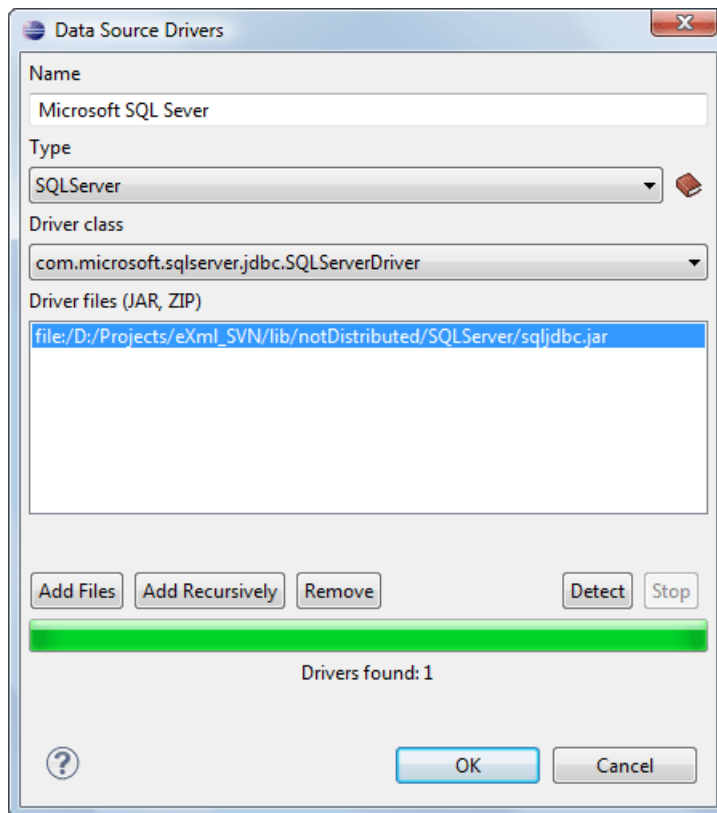


Figure 163: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **SQLServer** in the driver type combo box.
5. Add the Microsoft SQL Server driver file using the **Add** button.

The SQL Server driver file is called `sqljdbc.jar`. In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing Microsoft SQL Server databases in Oxygen XML Developer plugin .

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a Generic JDBC Data Source

Oxygen XML Developer plugin 's default configuration already contains a generic JDBC data source called **JDBC-ODBC Bridge**. Oxygen XML Developer plugin can display and edit XML data stored in PostgreSQL and Microsoft SQL Server databases accessible through a JDBC 4 driver. To do this, configure a **Generic JDBC** data source that uses a JDBC 4 driver. The following procedure shows you how to configure a generic JDBC data source:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

The following dialog is displayed:

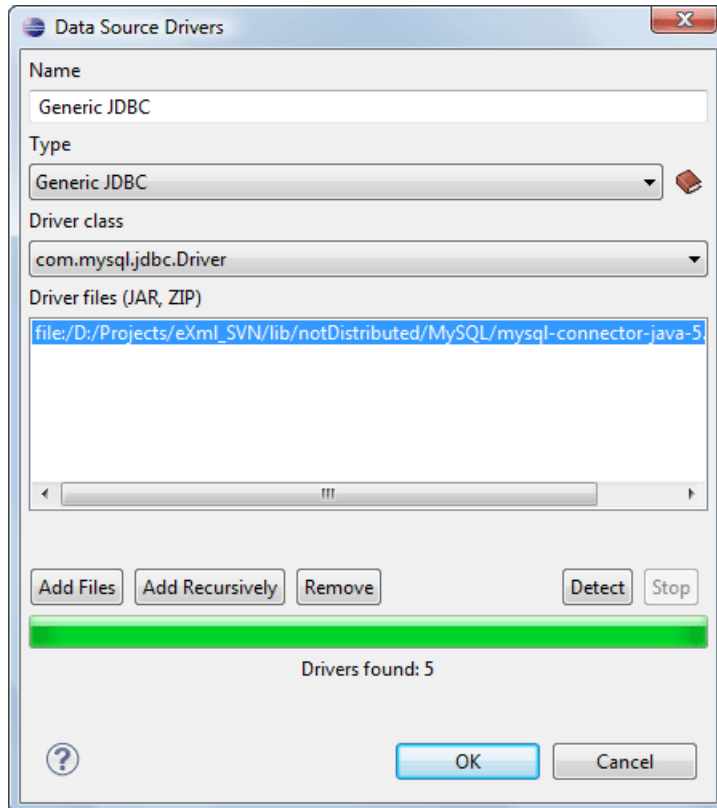


Figure 164: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **Generic JDBC** in the driver type combo box.
5. Add the driver file(s) using the **Add** button.
6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a MySQL Data Source

Previous versions of Oxygen XML Developer plugin (up to version 11.2) included a built-in type of data sources called **MySQL** and based on the JDBC driver for the MySQL 4 server. That type of data source is still available but is marked *outdated* because it does not support more recent versions of the MySQL server (starting from version 5.0) and it will be removed in a future version of Oxygen XML Developer plugin. For connecting to a MySQL server you should create a new data source of type Generic JDBC based on [the MySQL JDBC driver available from the MySQL website](#). The steps for configuring such a data source are the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

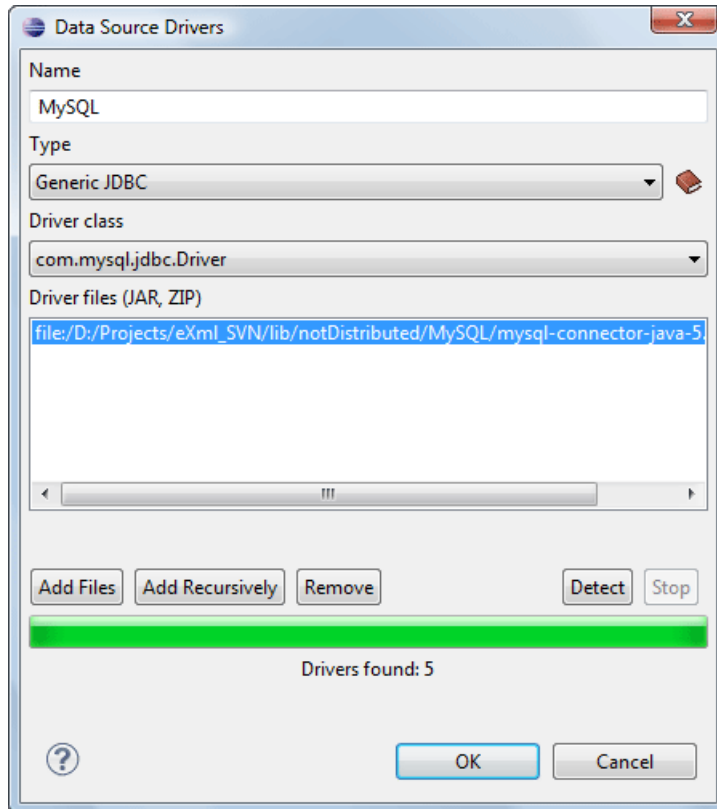


Figure 165: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **Generic JDBC** in the driver type combo box.
5. Add the MySQL 5 driver files using the **Add** button.

The driver file for the MySQL server is called `mysql-com.jar`. In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing MySQL databases in Oxygen XML Developer plugin .

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure an Oracle 11g Data Source

The steps for configuring a data source for connecting to an Oracle 11g server are the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

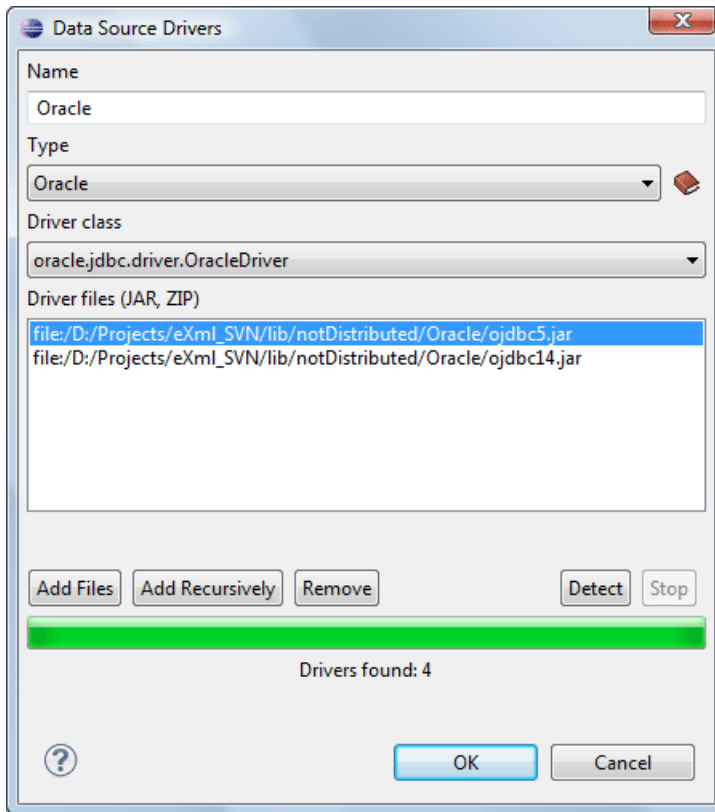


Figure 166: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **Oracle** in the driver type combo box.
5. Add the Oracle driver file using the **Add** button.

The Oracle driver file is called `ojdbc5.jar`. In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing Oracle databases in Oxygen XML Developer plugin .

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a PostgreSQL 8.3 Data Source

The steps for configuring a data source for connecting to a PostgreSQL server are the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.

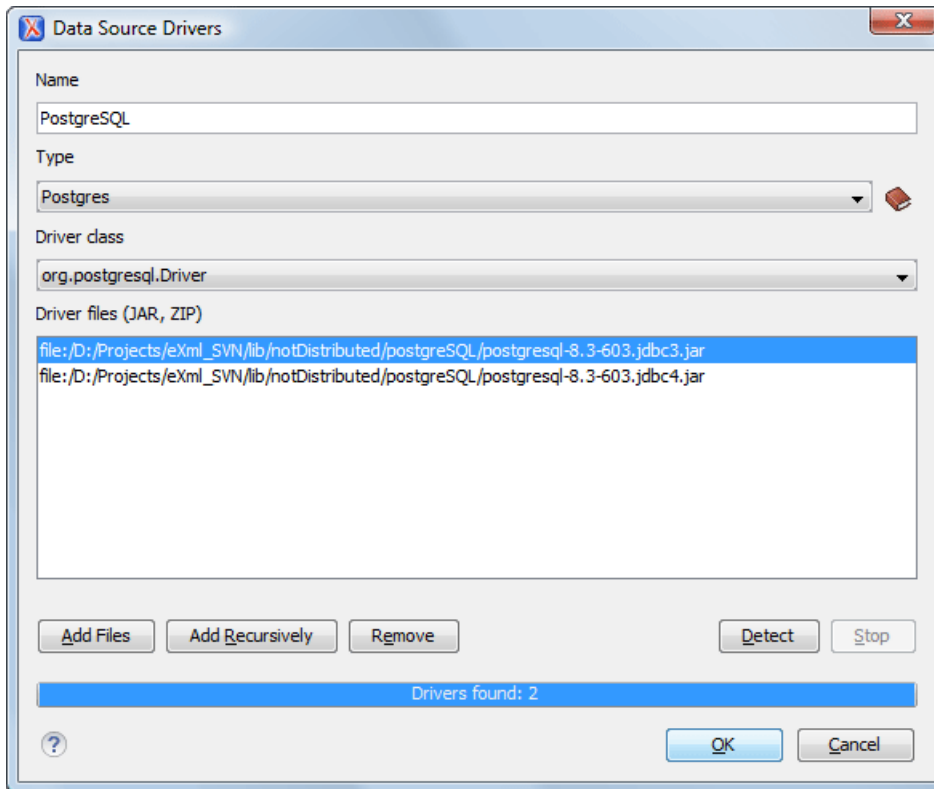


Figure 167: Data Source Drivers Configuration Dialog

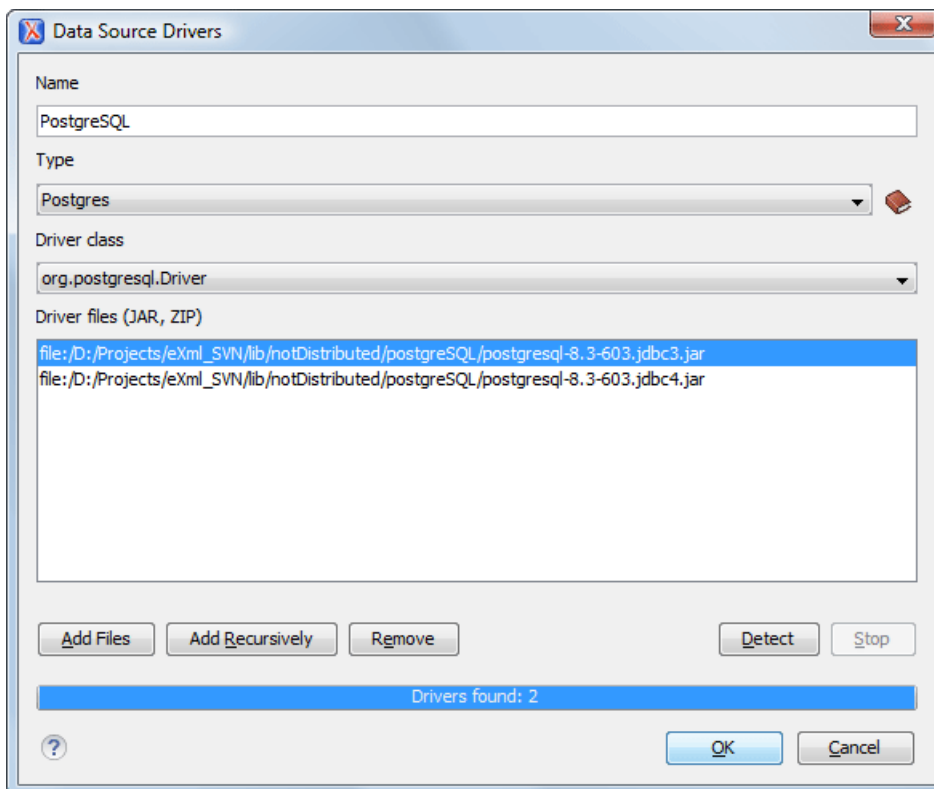


Figure 168: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.

4. Select **PostgreSQL** in the driver type combo box.
5. Add the PostgreSQL driver file using the **Add** button.

The PostgreSQL driver file is called `postgresql-8.3-603.jdbc3.jar`. In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing PostgreSQL databases in Oxygen XML Developer plugin .

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

Configuring Database Connections

This section describes the procedures for configuring the connections for relational databases:

How to Configure an IBM DB2 Connection

Available in the Enterprise edition only.

The steps for configuring a connection to an IBM DB2 server are the following:

1. Go to menu **Preferences > Data Sources**.
2. In the **Connections** panel click the **New** button.

The dialog for configuring a database connection will be displayed.

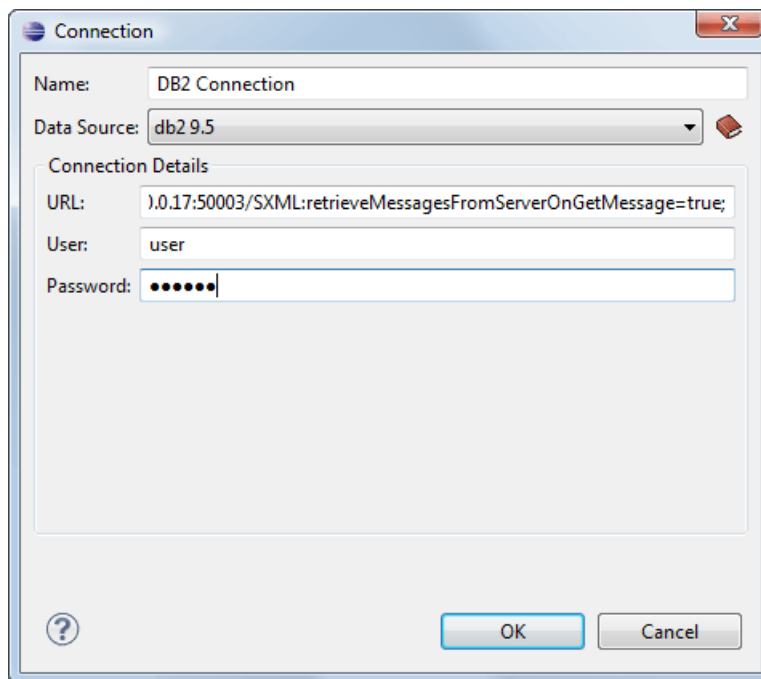


Figure 169: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select an IBM DB2 data sources in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL to the installed IBM DB2 engine.
 - b) Fill-in the user name to access the IBM DB2 engine.
 - c) Fill-in the password to access the IBM DB2 engine.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure a JDBC-ODBC Connection

The steps for configuring a connection to an ODBC data source are the following:

1. Go to menu **Preferences > Data Sources**.
2. In the **Connections** panel click the **New** button.

The dialog for configuring a database connection will be displayed.

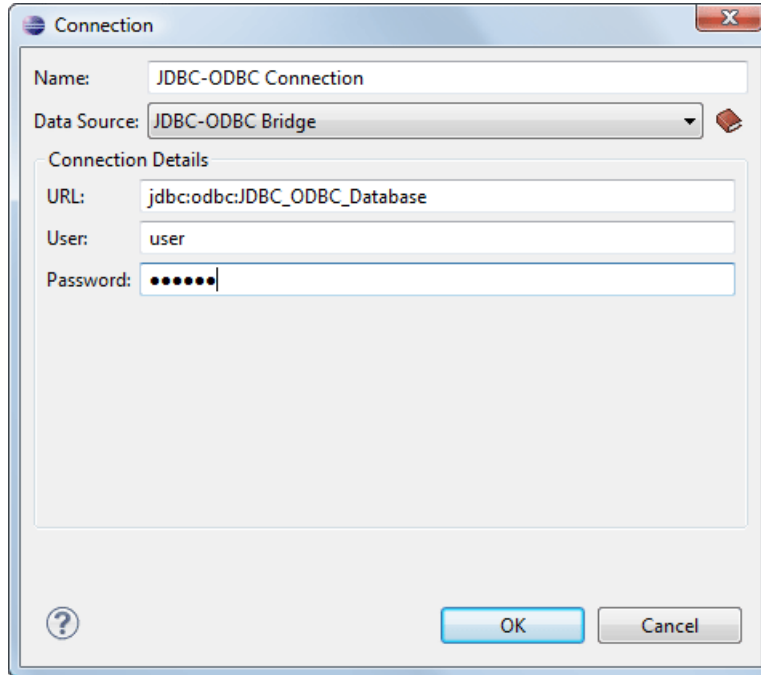


Figure 170: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select JDBC-ODBC bridge in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the ODBC source.
 - b) Fill-in the user name of the ODBC source.
 - c) Fill-in the password of the ODBC source.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure a Microsoft SQL Server Connection

Available in the Enterprise edition only.

The steps for configuring a connection to a Microsoft SQL Server server are the following:

1. Go to menu **Preferences > Data Sources**.
2. In the **Connections** panel click the **New** button.

The dialog for configuring a database connection will be displayed.

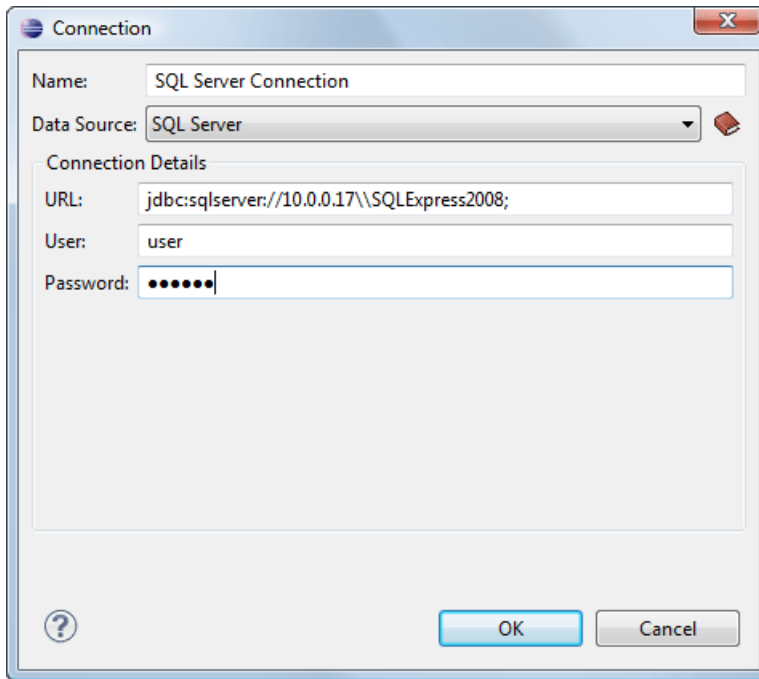


Figure 171: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select a SQL Server data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the SQL Server server.
If you want to connect to the server using Windows integrated authentication you must add `;integratedSecurity=true` to the end of the URL, so the URL will look like:

```
jdbc:sqlserver://localhost:1433;databaseName=test;integratedSecurity=true
```

- b) Fill-in the user name for the connection to the SQL Server.
 - c) Fill-in the password for the connection to the SQL Server.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure a MySQL Connection

The steps for configuring a connection to a MySQL server are the following:

1. Go to menu **Preferences > Data Sources**.
2. In the **Connections** panel click the **New** button.

The dialog for configuring a database connection will be displayed.

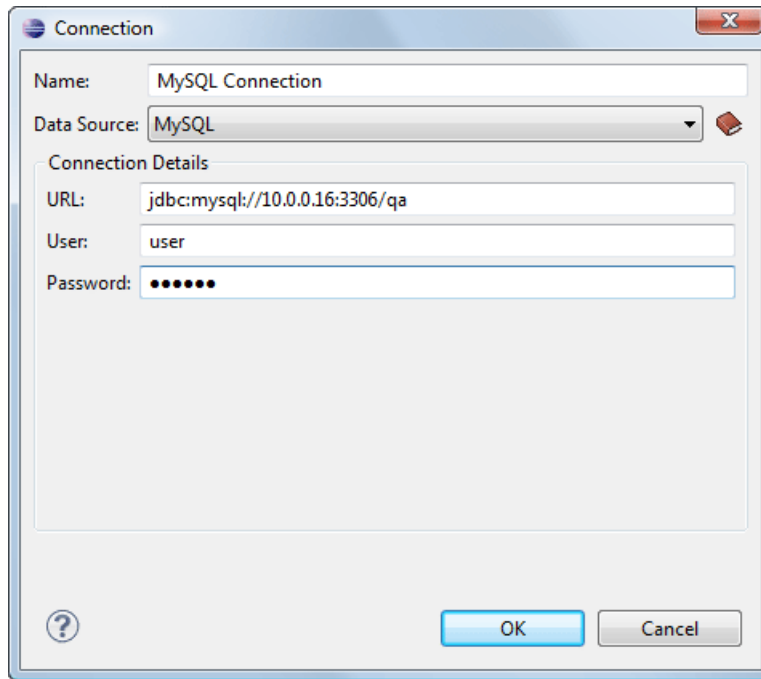


Figure 172: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select a MySQL data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the MySQL server.
 - b) Fill-in the user name for the connection to the MySQL server.
 - c) Fill-in the password for the connection to the MySQL server.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure a Generic JDBC Connection

The steps for configuring a connection to a generic JDBC database are the following:

1. Go to menu **Preferences > Data Sources**.
2. In the **Connections** panel click the **New** button.

The dialog for configuring a database connection will be displayed.
3. Enter a unique name for the connection.
4. Select a generic JDBC data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the generic JDBC database, with the following format: `jdbc: <subprotocol>: <subname>`.
 - b) Fill-in the user name for the connection to the generic JDBC database.
 - c) Fill-in the password for the connection to the generic JDBC database.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure an Oracle 11g Connection

Available in the Enterprise edition only.

The steps for configuring a connection to an Oracle 11g server are the following:

1. Go to menu **Preferences > Data Sources**.
2. In the **Connections** panel click the **New** button.

The dialog for configuring a database connection will be displayed.

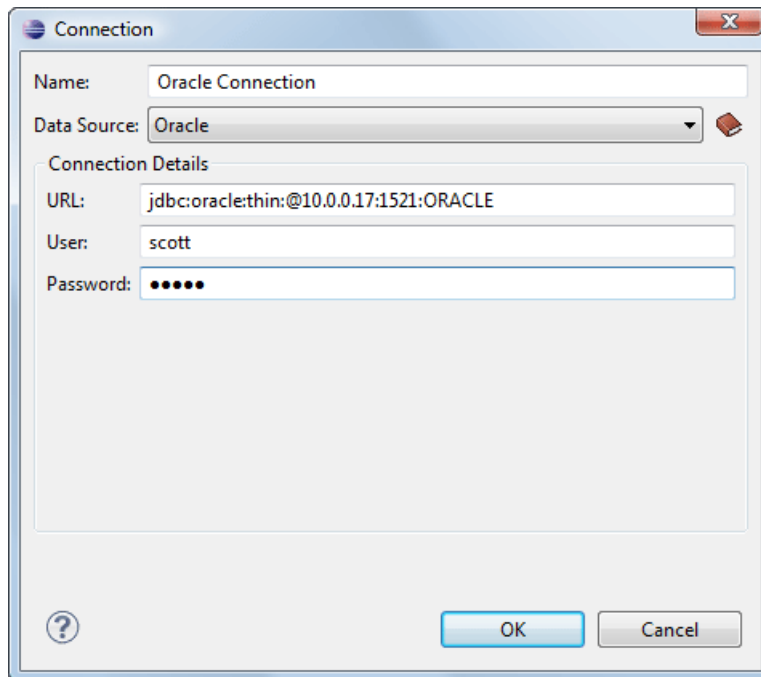


Figure 173: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select an Oracle 11g data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the Oracle server.
 - b) Fill-in the user name for the connection to the Oracle server.
 - c) Fill-in the password for the connection to the Oracle server.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure a PostgreSQL 8.3 Connection

Available in the Enterprise edition only.

The steps for configuring a connection to a PostgreSQL 8.3 server are the following:

1. Go to menu **Preferences > Data Sources**.
2. In the **Connections** panel click the **New** button.

The dialog for configuring a database connection will be displayed.

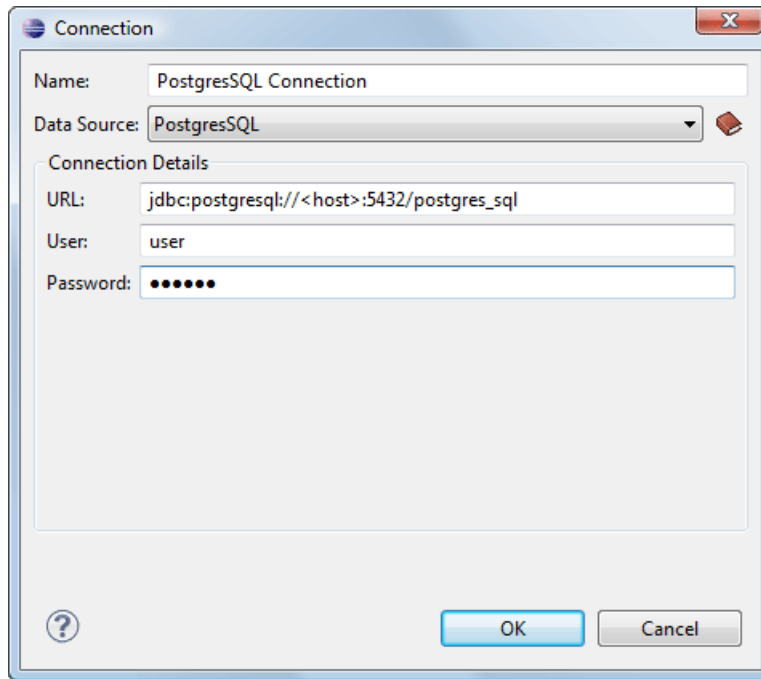


Figure 174: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select a PostgreSQL 8.3 data source in the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Fill-in the URL of the PostgreSQL 8.3 server.
 - b) Fill-in the user name for the connection to the PostgreSQL 8.3 server.
 - c) Fill-in the password for the connection to the PostgreSQL 8.3 server.
6. Click the **OK** button to finish the configuration of the database connection.

Resource Management

This section explains the resource management actions for relational databases.

Data Source Explorer View

This view presents in a tree-like fashion the database connections configured from menu **Options > Preferences > Data Sources**. You can connect to a database simply by expanding the connection node. The database structure can be expanded up to column level. Oxygen XML Developer plugin supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.

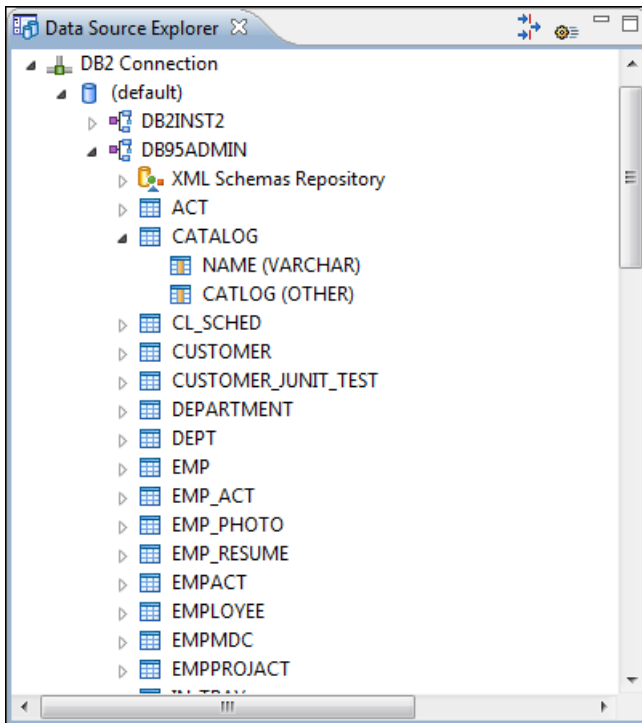















Figure 175: Data Source Explorer View

The following objects are displayed by the **Data Source Explorer** view:



-  **Connection**
-  **Catalog (Collection)**
-  **XML Schema Repository**
-  **XML Schema Component**
-  **Schema**
-  **Table**
-  **System Table**
-  **Table Column**

A  **collection** (called *catalog* in some databases) is a hierarchical container for  **resources** and further sub-collections. There are two types of resources:


-  **XML resource** - an XML document or a document fragment, selected by a previously executed XPath query.
-  **non XML resource**



 **Note:** For some connections you can add or move resources into container by dragging them from **Project view**, the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) or from another database container.

The following actions are available in the view's toolbar:


- The  **Filters** button opens the **Data Sources / Table Filters Preferences page**, allowing you to decide which table types will be displayed in the **Data Source Explorer** view.
- The  **Configure Database Sources** button opens the **Data Sources preferences page** where you can configure both data sources and connections.

Actions Available at Connection Level in Data Source Explorer View

The contextual menu of a  **Connection** node of the tree from the **Data Source Explorer** view contains the following actions:


-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Disconnect** - Closes the current database connection. If a table is already open, you are warned to close it before proceeding.
-  **Configure Database Sources** - Opens the **Data Sources preferences page** where you can configure both data sources and connections.


Actions Available at Catalog Level in Data Source Explorer View

The contextual menu of a  **Catalog** node of the tree from the **Data Source Explorer** view contains the following actions:


-  **Refresh** - Performs a refresh of the selected node's subtree.




Actions Available at Schema Level in Data Source Explorer View

The contextual menu of a  **Schema** node of the tree from the **Data Source Explorer** view contains the following actions:

-  **Refresh** - Performs a refresh of the selected node's subtree.

Actions Available at Table Level in Data Source Explorer View


The contextual menu of a  **Table** node of the tree from the **Data Source Explorer** view contains the following actions:


-  **Refresh** - Performs a refresh of the selected node's subtree.
-  **Edit** - Opens the selected table in the **Table Explorer** view.
-  **Export to XML** - Opens the **Export Criteria** dialog (a thorough description of this dialog can be found in the [Import from Database](#) chapter).


XML Schema Repository Level

This section explains the actions available at XML Schema Repository level.

Oracle's XML Schema Repository Level

The Oracle database supports XML schema repository (XSR) in the database catalogs. The contextual menu of a  **XML Schema Repository** node of the tree from the **Data Source Explorer** view contains the following actions:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Register** - Opens a dialog for adding a new schema file in the XML repository. To add an XML Schema, enter the schema URI and location on your file system. Local scope means that the schema will be visible only to the user who registers it. Global scope means that the schema is public.


 **Note:** Registering a schema may involve dropping/creating types. Hence you need type-related privileges such as DROP TYPE, CREATE TYPE, and ALTER TYPE. You need privileges to delete and register the XML schemas involved in the registering process. You need all privileges on XMLType tables that conform to the registered schemas. For XMLType columns, the ALTER TABLE privilege is needed on corresponding tables. If there are schema-based XMLType tables or columns in other database schemas, you need privileges such as the following:


- CREATE ANY TABLE
- CREATE ANY INDEX
- SELECT ANY TABLE
- UPDATE ANY TABLE

- INSERT ANY TABLE
- DELETE ANY TABLE
- DROP ANY TABLE
- ALTER ANY TABLE
- DROP ANY INDEX

To avoid having to grant all these privileges to the schema owner, Oracle recommends that the registration be performed by a DBA if there are XML schema-based XMLType table or columns in other users' database schemas.

IBM DB2's XML Schema Repository Level



The contextual menu of a  **XML Schema Repository** node of the tree from the **Data Source Explorer** view contains the following actions:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Register** - Opens a dialog for adding a new schema file in the XML Schema repository. In this dialog the following fields can be set:
 - **XML schema file** - Location on your file system.
 - **XSR name** - Schema name.
 - **Comment** - Short comment (optional).
 - **Schema location** - Primary schema name (optional).


Decomposition means that parts of the XML documents are stored into relational tables. Which parts map to which tables and columns is specified into the schema annotations.


Schema dependencies management is done by using the **Add** and **Remove** buttons.

The actions available at  **Schema** level are the following:

-  **Refresh** - Performs a refresh of the selected node (and it's subtree).
- **Unregister** - Removes the selected schema from the XML Schema Repository.
-  **View** - Opens the selected schema in Oxygen XML Developer plugin .

Microsoft SQL Server's XML Schema Repository Level

The contextual menu of a  **XML Schema Repository** node of the tree from the **Data Source Explorer** view contains the following actions:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Register** - Opens a dialog for adding a new schema file in the DB XML repository. In this dialog you enter a collection name and the necessary schema files. XML Schema files management is done by using the **Add** and **Remove** buttons.

The actions available at  **Schema** level are the following:



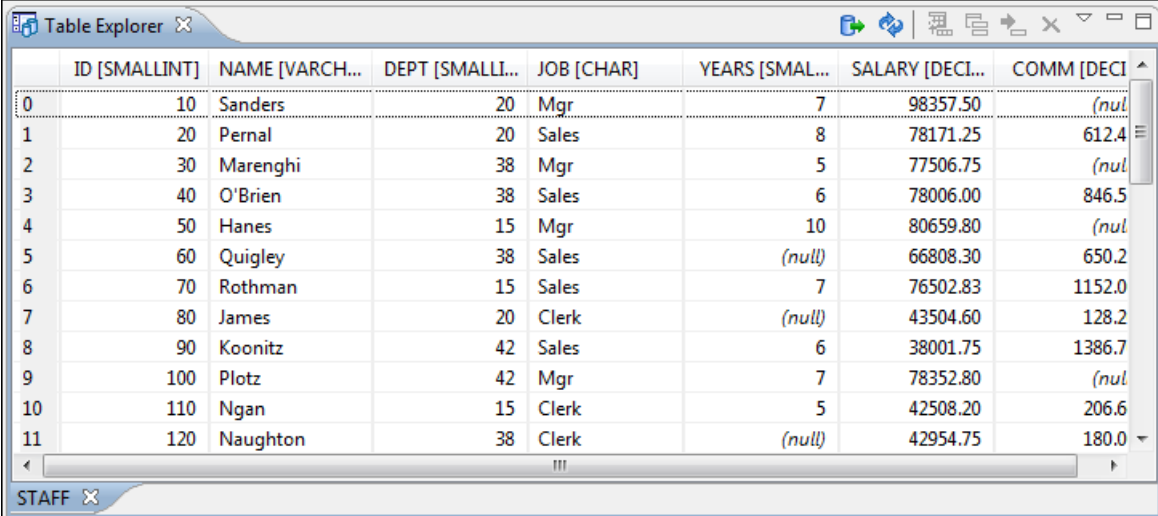
-  **Refresh** - Performs a refresh of the selected node (and it's subtree).
- **Add** - Adds a new schema to the XML Schema files.
- **Unregister** - Removes the selected schema from the XML Schema Repository.
-  **View** - Opens the selected schema in Oxygen XML Developer plugin .

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 a cell's content, double click

it and start typing. When editing is finished, Oxygen XML Developer plugin will try to update the database with the new cell content.




	ID [SMALLINT]	NAME [VARCHAR...]	DEPT [SMALL...]	JOB [CHAR]	YEARS [SMAL...]	SALARY [DECI...]	COMM [DECI...]
0	10	Sanders	20	Mgr	7	98357.50	(nul
1	20	Pernal	20	Sales	8	78171.25	612.4
2	30	Marenghi	38	Mgr	5	77506.75	(nul
3	40	O'Brien	38	Sales	6	78006.00	846.5
4	50	Hanes	15	Mgr	10	80659.80	(nul
5	60	Quigley	38	Sales	(null)	66808.30	650.2
6	70	Rothman	15	Sales	7	76502.83	1152.0
7	80	James	20	Clerk	(null)	43504.60	128.2
8	90	Koonitz	42	Sales	6	38001.75	1386.7
9	100	Plotz	42	Mgr	7	78352.80	(nul
10	110	Ngan	15	Clerk	5	42508.20	206.6
11	120	Naughton	38	Clerk	(null)	42954.75	180.0

Figure 176: The Table Explorer View

You can sort the content of a table by one of its columns by clicking on its column header.

Note the following:

- The first column is an index (does not belong to the table structure).
- Every column header contains the field name and its data type.
- The primary key columns are marked with this symbol:  .
- Multiple tables are presented in a tabbed manner

For performance issues, you can set the maximum number of cells that will be displayed in the **Table Explorer** view (the **Limit the number of cells** field from the *Data Sources* Preferences page). If a table having more cells than the value set in Oxygen XML Developer plugin 's options is displayed in the **Table Explorer** view, a warning dialog will inform you that the table is only partially shown.

You will be notified if the value you have entered in a cell is not valid (and thus it cannot be updated).

- If the content of the edited cell does not belong to the data type of the column, an Information dialog will appear, notifying you that the value you have inserted cannot be converted to the SQL type of that field. For example, in the above figure `propID` contains `LONG` values. If a character or string was inserted, you would get the error message that a String value cannot be converted to the requested SQL type (NUMBER).
- If the constraints of the database are not met (like primary key constraints for example), an Information dialog will appear, notifying you of the reason the database has not been updated. For example, if you'd try to set the primary key `propID` for the second record in the table to 10 also, you would get the following message:

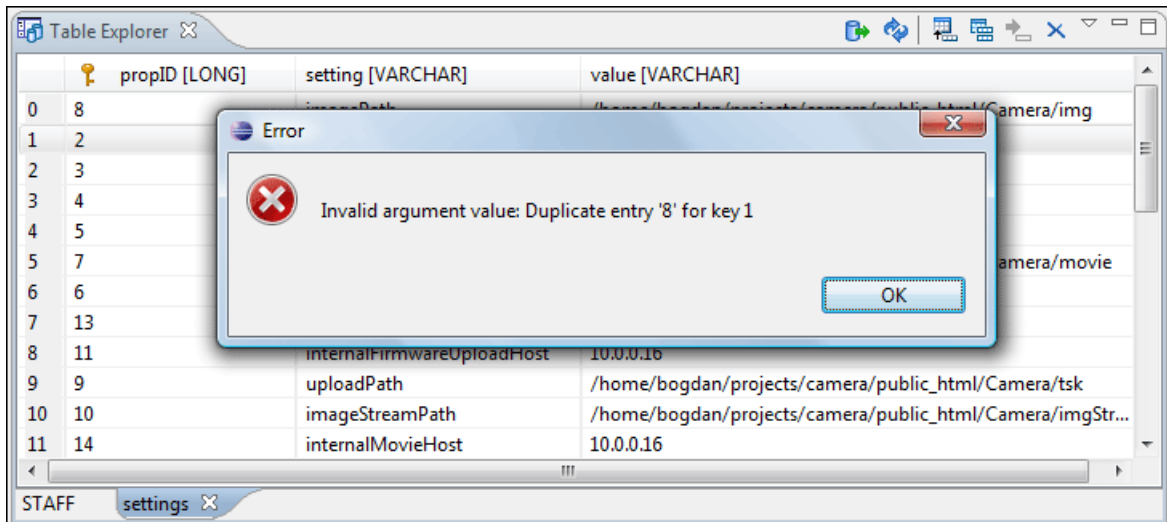


Figure 177: Duplicate entry for primary key

The usual edit actions (**Cut**, **Copy**, **Paste**, **Select All**, **Undo**, **Redo**) are available in the popup menu of the edited cell.

The contextual menu available on every cell has the following actions:

- Set NULL - Sets the content of the cell to (null). This action is disabled for columns that cannot be null.
- **Insert row** - Inserts an empty row in the table.
- **Duplicate row** - Makes a copy of the selected row and adds it in the **Table Explorer** view. You should note that the new row will not be inserted in the database table until all conflicts are solved.
- **Commit row** - Commits the selected row.
- **Delete row** - Deletes the selected row.
- **Copy** - Copies the content of the cell.
- **Paste** - Performs paste in the selected cell.

Some of the above actions are also available on the **Table Explorer** toolbar:

- **Export to XML** - Opens the **Export Criteria** dialog (a thorough description of this dialog can be found in the [Import from database](#) chapter).
- **Refresh** - Performs a refresh of the selected node's subtree.
- **Insert row** - Inserts an empty row in the table.
- **Duplicate row** - Makes a copy of the selected row and adds it in the **Table Explorer** view. You should note that the new row will not be inserted in the database table until all conflicts are solved.
- **Commit row** - Commits the selected row.
- **Delete row** - Deletes the selected row.

SQL Execution Support

Oxygen XML Developer plugin 's support for writing SQL statements includes syntax highlight, folding and drag&drop (DND) from the **Data Source Explorer** view. It also includes transformation scenarios for executing the statements and the results are displayed in the **Table Explorer** view.

Drag and Drop from Data Source Explorer View

Drag and drop (DND) from the **Data Source Explorer** view to the SQL editor allows creating SQL statements quickly by inserting the names of tables and columns in the SQL statements.

1. Configure a database connection (see the procedure specific for your database server).
2. Browse to the table you will use in your statement.

3. Drag the table or a column of the table into the editor where a SQL file is open.

DND is available both on the table and on its fields. A popup menu is displayed in the SQL editor.

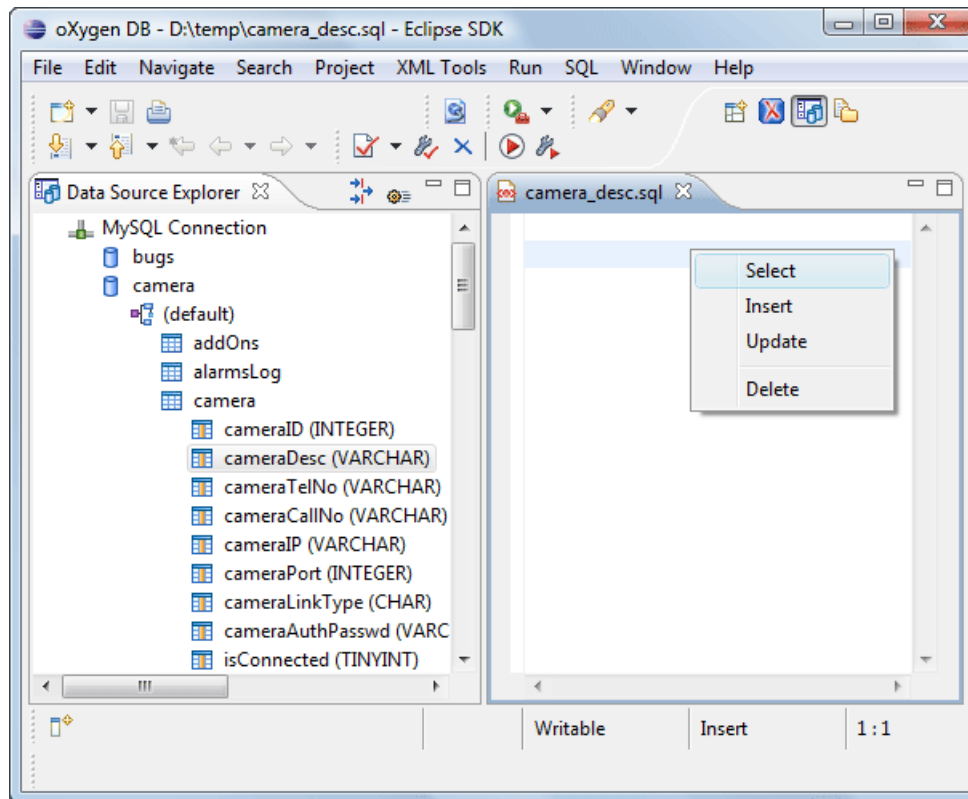


Figure 178: SQL statement editing with DND

4. Select the type of statement from the popup menu.

If you dragged a table depending on your choice, one of the following statements are inserted into the document:

- `SELECT `field1`,`field2`,... FROM `catalog`.`table`` (for this example: `SELECT `DEPT`,`DEPTNAME`,`LOCATION` FROM `test`.`department``)
- `UPDATE `catalog`.`table` SET `field1`=,`field2`=,...` (for this example: `UPDATE `test`.`department` SET `DEPT`=, `DEPTNAME`=, `LOCATION`=`)
- `INSERT INTO `catalog`.`table` (`field1`,`field2`,...) VALUES (, ,)` (for this example: `INSERT INTO `test`.`department` (`DEPT`,`DEPTNAME`,`LOCATION`) VALUES (, ,)`)
- `DELETE FROM `catalog`.`table`` (for this example: `DELETE FROM `test`.`department``)

If you dragged a column depending on your choice, one of the following statements are inserted into the document:

- `SELECT `field` FROM `catalog`.`table`` (for this example: `SELECT `DEPT` FROM `test`.`department``)
- `UPDATE `catalog`.`table` SET `field`=` (for this example: `UPDATE `test`.`department` SET `DEPT`=`)
- `INSERT INTO `catalog`.`table` (`field1) VALUES ()` (for this example: `INSERT INTO `test`.`department` (`DEPT`) VALUES ()`)
- `DELETE FROM `catalog`.`table`` (for this example: `DELETE FROM `test`.`department` WHERE `DEPT`=`)


SQL Validation

Currently, SQL validation support is offered for IBM DB2. Please note that if you choose a connection that doesn't support SQL validation you will receive a warning when trying to validate. The SQL document will be validated using the connection from the associated transformation scenario.

Executing SQL Statements

The steps for executing an SQL statement on a relational database are the following:


1. Configure a *transformation scenario* from the  **Configure Transformation Scenario** button from the **Transformation** toolbar.

A SQL transformation scenario needs a database connection. You can configure a connection from the  **Preferences** button from the scenario dialog.

The dialog that appears contains the list of existing scenarios that apply to SQL documents.

2. Set parameter values for SQL placeholders from the **Parameters** button from the scenario dialog. For example in `SELECT * FROM `test`.`department` where DEPT = ? or DEPTNAME = ?` two parameters can be configured for the place holders (?) in the transformation scenario.

When the SQL statement will be executed, the first placeholder will be replaced with the value set for the first parameter in the scenario, the second placeholder will be replaced by the second parameter value and so on.

 **Restriction:** When a stored procedure is called in an SQL statement executed on an SQL Server database mixing in-line parameter values with values specified using the **Parameters** button of the scenario dialog is not recommended. It is due to a limitation of the SQL Server driver for Java applications. An example of stored procedure call that is not recommended is: `call dbo.Test(22, ?)`.

3. Execute the SQL scenario from the **Transform now** button of the scenario dialog.

The result of a SQL transformation will be *displayed in 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 value or a CLOB one) cannot be displayed entirely in the result table.

- a) Right click on the cell containing the complex value.
- b) Select the action **Copy cell** from the popup menu.
The action will copy the value in the clipboard.
- c) Paste the value where you need it.

For example you can paste the value in an opened XQuery editor panel of Oxygen XML 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
- Software AG Tamino
- Raining Data TigerLogic
- Documentum xDb (X-Hive/DB) 10
- Oracle XML DB

Configuring Database Data Sources

This section describes the procedures for configuring the data sources for native databases.

How to Configure a Berkeley DB XML Data Source

The latest instructions on how to configure Berkeley DB XML support in Oxygen XML Developer plugin can be found on our [website](#).

Oxygen XML Developer plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The following directory definitions shall apply:

- OXY_DIR - Oxygen XML Developer plugin installation root directory. (for example on Windows C:\Program Files\Oxygen 13.2.0)
- DBXML_DIR - Berkeley DB XML database root directory. (for example on Windows C:\Program Files\Oracle\Berkeley DB XML <version>)
- DBXML_LIBRARY_DIR (usually on Mac and Unix is DBXML_DIR / lib and on Windows is DBXML_DIR / bin)

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *Berkeley DBXML* from the **Driver type** combo box.
5. Press the **Add** button to add the Berkeley DB driver files.

The driver files for the Berkeley DB database are the following:

- db.jar (check for it into DBXML_DIR / lib or DBXML_DIR / jar)
- dbxml.jar (check for it into DBXML_DIR / lib or DBXML_DIR / jar)

6. Click the **OK** button to finish the data source configuration.

How to Configure an eXist Data Source


The latest instructions on how to configure eXist support in Oxygen XML Developer plugin can be found on our [website](#).

Oxygen XML Developer plugin supports eXist database server versions 1.3, 1.4 and 1.5.

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *eXist* from the **Driver type** combo box.
5. Press the **Add** button to add the eXist driver files.

The following driver files should be added in the dialog box for setting up the eXist datasource. They are found in the installation directory of the eXist database server. Please make sure you copy the files from the installation of the eXist server where you want to connect from Oxygen.

- exist.jar
- lib/core/xmlldb.jar
- lib/core/xmlrpc-client-3.1.1.jar
- lib/core/xmlrpc-common-3.1.1.jar
- lib/core/ws-commons-util-1.0.2.jar

 **Note:** For eXist database server version 1.5, the following driver files must also be added in the dialog box for setting up the datasource:

- lib/core/slf4j-api-1.x.x.jar
- lib/core/slf4j-log4j12-1.x.x.jar

- lib/core/slf4j-simple-1.x.x.jar

The version number from the driver file names may be different for your eXist server installation.

6. Click the **OK** button to finish the data source configuration.

How to Configure a MarkLogic Data Source

The latest instructions on how to configure MarkLogic support in Oxygen XML Developer plugin can be found on our [website](#).

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *MarkLogic* from the **Driver type** combo box.
5. Press the **Add** button to add the MarkLogic driver files.


The driver files for the MarkLogic database are:

- xcc.jar
- xdbc.jar
- xdmp.jar

In the [Download links for database drivers](#) section there are listed the URLs from where to download the drivers necessary for accessing MarkLogic databases in Oxygen XML Developer plugin.

6. Click the **OK** button to finish the data source configuration.

How to Configure a Software AG Tamino Data Source

 **Note:** Support for Tamino database will be discontinued starting with version 14.

The latest instructions on how to configure Software AG Tamino support in Oxygen XML Developer plugin can be found on our [website](#).

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *Tamino* from the **Driver type** combo box.
5. Press the **Add** button to add the Tamino driver files.


The driver files for the Tamino database are the following:

- TaminoAPI4J.jar
- TaminoAPI4J-110n.jar
- TaminoJCA.jar

 **Note:** You must use the jar files from the version 4.4.1 of the Tamino database.

6. Click the **OK** button to finish the data source configuration.

How to Configure a Raining Data TigerLogic Data Source

 **Note:** Support for TigerLogic database will be discontinued starting with version 14.

The latest instructions on how to configure TigerLogic support in Oxygen XML Developer plugin can be found on our [website](#).

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

3. Enter a unique name for the data source.
4. Select *TigerLogic* from the **Driver type** combo box.
5. Press the **Add** button to add the TigerLogic driver files.

The driver files for the TigerLogic database are found in the TigerLogic JDK lib directory from the server side:

- `connector.jar`
- `jca-connector.jar`
- `tlapi.jar`
- `terror.jar`
- `utility.jar`
- `xmlparser.jar`
- `xmltypes.jar`

6. Click the **OK** button to finish the data source configuration.

How to Configure a Documentum xDb (X-Hive/DB) 10 Data Source

The latest instructions on how to configure support for Documentum xDb (X-Hive/DB) 10 versions 8 and 9 in Oxygen XML Developer plugin can be found on our [website](#).

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *XHive* from the **Driver type** combo box.
5. Press the **Add** button to add the XHive driver files.

The driver files for the Documentum xDb (X-Hive/DB) 10 database are found in the Documentum xDb (X-Hive/DB) 10 lib directory from the server installation folder:

- `antlr-runtime.jar`
- `aspectjrt.jar`
- `icu4j.jar`
- `xhive.jar`
- `google-collect.jar`

6. Click the **OK** button to finish the data source configuration.

Configuring Database Connections

This section describes the procedures for configuring the connections for native databases.

How to Configure a Berkeley DB XML Connection

Oxygen XML 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 the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Set the path to the Berkeley DB XML home directory in the **Environment home directory** field.
 - b) Select the **Verbosity** level: DEBUG, INFO, WARNING or ERROR.
 - c) Optionally, you can select the checkbox **Join existing environment**.

If checked, an attempt will be made to join an existing environment in the specified home directory and all the original environment settings will be preserved. If that fails, you should consider reconfiguring the connection with this option unchecked.

6. Click the **OK** button to finish the connection configuration.

How to Configure an eXist Connection

The steps for configuring a connection to an eXist database are the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Set the URI to the installed eXist engine in the **XML DB URI** field.
 - b) Set the user name in the **User** field.
 - c) Set the password in the **Password** field.
 - d) Enter the start collection in the **Collection** field.

eXist organizes all documents in hierarchical collections. Collections are like directories. They are used to group related documents together. This text field allows the user to set the default collection name.

6. Click the **OK** button to finish the connection configuration.

How to Configure a MarkLogic Connection

Available in the Enterprise edition only.


The steps for configuring a connection to a MarkLogic database are the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
 - a) The host name or IP address of the installed MarkLogic engine in the **XDBC Host** field.
Oxygen 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 the **Port** field.
 - c) Set the user name to access the MarkLogic engine in the **User** field.
 - d) Set the password to access the MarkLogic engine in the **Password** field.
 - e) Optionally set the URL used for browsing the MarkLogic database in the **Data Source Explorer** view in the **WebDAV URL** field.

6. Click the **OK** button to finish the connection configuration.

How to Configure a Software AG Tamino Connection

Available in the Enterprise edition only.

 **Note:** Support for Tamino database will be discontinued starting with version 14.


The steps for configuring a connection to a Tamino database are the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Connections** panel.

3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Set the URI to the installed Tamino engine in the **XML DB URI** field.
 - b) Set the user name to access the Tamino engine in the **User** field.
 - c) Set the password to access the Tamino engine in the **Password** field.
 - d) Set the name of the database to access from the Tamino engine in the **Database** field.
 - e) Check the checkbox **Show system collections** if you want to see the Tamino system collections in the **Data Source Explorer** view.
6. Click the **OK** button to finish the connection configuration.

How to Configure a Raining Data TigerLogic Connection

Available in the Enterprise edition only.


 **Note:** Support for TigerLogic database will be discontinued starting with version 14.

The steps for configuring a connection to a TigerLogic database are the following:

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Set the host name or IP address of the TigerLogic engine in the **Host** field.
 - b) Set the port number of the TigerLogic engine in the **Port** field.
 - c) Set the user name to access the TigerLogic engine in the **User** field.
 - d) Set the password to access the TigerLogic engine in the **Password** field.
 - e) Set the name of the database to access from the TigerLogic database engine in the **Database** field.
6. Click the **OK** button to finish the connection configuration.

How to Configure an Documentum xDb (X-Hive/DB) 10 Connection

The steps for configuring a connection to a Documentum xDb (X-Hive/DB) 10 database are the following.

 **Note:** The bootstrap type of X-Hive/DB connections is not supported in Oxygen XML Developer plugin. The following procedure explains the *xhive://* protocol connection type.

1. Go to menu **Preferences > Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
 - a) Set the URL property of the connection in the **URL** field.

If the property is a URL of the form *xhive://host:port*, the Documentum xDb (X-Hive/DB) 10 connection will attempt to connect to a Documentum xDb (X-Hive/DB) 10 server running behind the specified TCP/IP port.
 - b) Set the user name to access the Documentum xDb (X-Hive/DB) 10 engine in the **User** field.
 - c) Set the password to access the Documentum xDb (X-Hive/DB) 10 engine in the **Password** field.
 - d) Set the name of the database to access from the Documentum xDb (X-Hive/DB) 10 engine in the **Database** field.
 - e) Check the checkbox **Run XQuery in read / write session (with committing)** if you want to end the session with a commit, otherwise the session ends with a rollback.
6. Click the **OK** button to finish the connection configuration.

Data Source Explorer View

This view presents in a tree-like fashion the database connections configured from menu **Options > Preferences > Data Sources**. You can connect to a database simply by expanding the connection node. The database structure can be expanded up to column level. Oxygen XML Developer plugin supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.

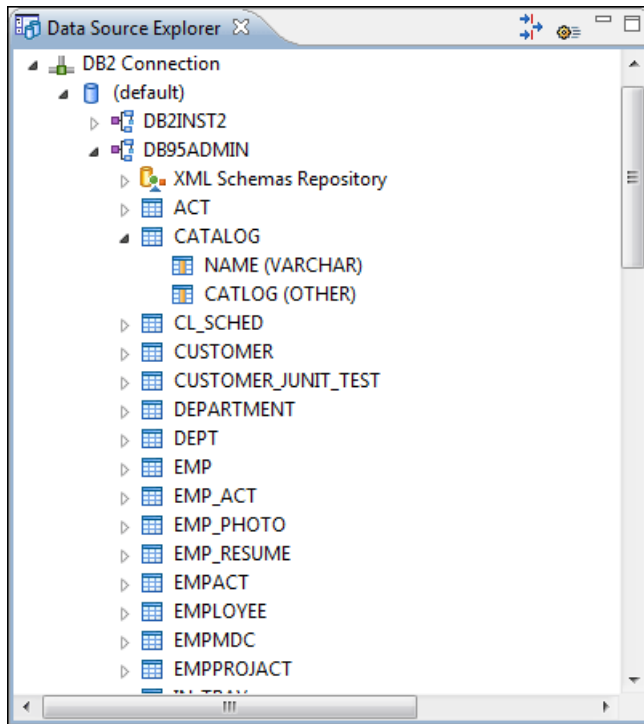















Figure 179: Data Source Explorer View

The following objects are displayed by the **Data Source Explorer** view:



-  **Connection**
-  **Catalog (Collection)**
-  **XML Schema Repository**
-  **XML Schema Component**
-  **Schema**
-  **Table**
-  **System Table**
-  **Table Column**

A  **collection** (called *catalog* in some databases) is a hierarchical container for  **resources** and further sub-collections. There are two types of resources:

-  **XML resource** - an XML document or a document fragment, selected by a previously executed XPath query.
-  **non XML resource**

 **Note:** For some connections you can add or move resources into container by dragging them from **Project view**, the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) or from another database container.

The following actions are available in the view's toolbar:

- The  **Filters** button opens the **Data Sources / Table Filters Preferences page**, allowing you to decide which table types will be displayed in the **Data Source Explorer** view.
- The  **Configure Database Sources** button opens the **Data Sources preferences page** where you can configure both data sources and connections.

Oracle XML DB Browser

Oracle XML DB is a feature of the Oracle database. It provides a high-performance, native XML storage and retrieval technology. Oxygen XML Developer plugin allows the user to browse the native Oracle XML Repository and perform various operations on the resources in the repository.

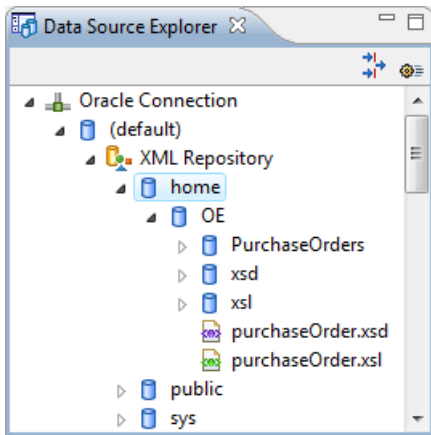






Figure 180: Browsing the Oracle XML DB Repository



The actions available at XML Repository level are the following:

-  **Refresh** - Performs a refresh of the XML Repository.
- **Add container** - Adds a new child container to the XML Repository
-  **Add resource** - Adds a new resource to the XML Repository.

The actions available at container level are the following:

-  **Refresh** - Performs a refresh of the selected container.
- **Add container** - Adds a new child container to the current one
-  **Add resource** - Adds a new resource to the folder.
- **Delete** - Deletes the current container.
- **Properties** - Shows various properties of the current container.

The actions available at resource level are the following:

-  **Refresh** - Performs a refresh of the selected resource.
-  **Open** - Opens the selected resource in the editor.
- **Rename** - Renames the current resource.
- **Move** - Moves the current resource to a new container (also available through drag and drop).
- **Delete** - Deletes the current resource.
- **Copy location** - Allows you to copy to clipboard an application specific URL for the resource which can then be used for various actions like opening or transforming the resources.
- **Properties** - Shows various properties of the current resource.

For running XQuery transformation on collections from XML Repository please see [a tutorial from Oracle](#).

PostgreSQL Connection

Oxygen XML Developer plugin allows the user to browse the structure of the PostgreSQL database in the **Data Source Explorer** view and open the tables in the **Table Explorer** view.

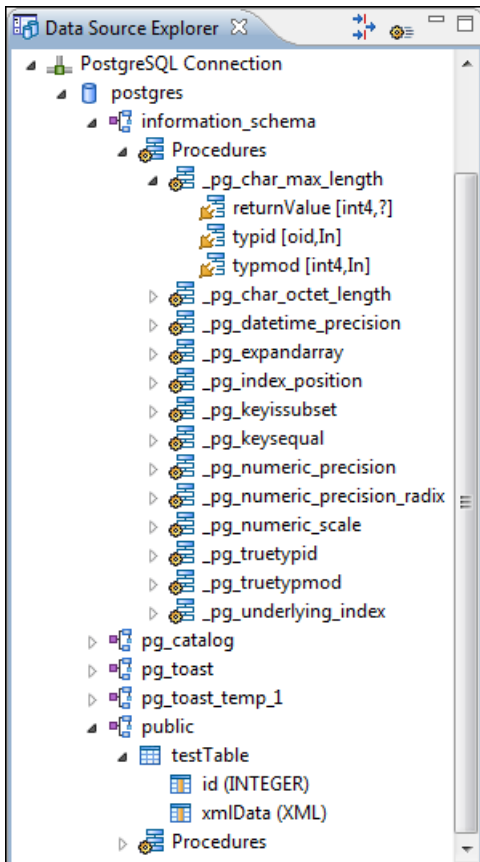






Figure 181: Browsing a PostgreSQL repository

The actions available at container level are the following:

-  **Refresh** - Performs a refresh of the selected container.

The actions available at resource level are the following:



-  **Refresh** - Performs a refresh of the selected database table.
-  **Edit** - Opens the selected database table in the **Table Explorer** view.
-  **Export to XML ...** - Exports the content of the selected database table as an XML file using [the dialog from importing data from a database](#).

Berkeley DB XML Connection

This section explains the actions that are available on a Berkeley DB XML connection.

Actions Available at Connection Level

In a Berkeley DB XML repository the actions available at connection level in the **Data Source Explorer** view are the following:



-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Disconnect** - Closes the current database connection.
-  **Configure Database Sources** - Opens [the Data Sources preferences page](#) where you can configure both data sources and connections.
- **Add container** - Adds a new container in the repository with the following attributes.
 - **Name** - The name of the new container.
 - **Container type** - At creation time, every container must have a type defined for it. This container type identifies how XML documents are stored in the container. As such, the container type can only be determined at container

creation time; you cannot change it on subsequent container opens. Containers can have one of the following types specified for them:

- **Node container** - XML documents are stored as individual nodes in the container. That is, each record in the underlying database contains a single leaf node, its attributes and attribute values if any, and its text nodes, if any. Berkeley DB XML also keeps the information it needs to reassemble the document from the individual nodes stored in the underlying databases. This is the default, and preferred, container type.
- **Whole document container** - The container contains entire documents. The documents are stored without any manipulation of line breaks or whitespace.
- **Allow validation** - If checked it causes documents to be validated when they are loaded into the container. The default behavior is to not validate documents.
- **Index nodes** - If checked it causes indices for the container to return nodes rather than documents. The default is to index at the document level. This property has no meaning if the container type is whole document container.
- **Properties** - Shows a dialog containing a list of the Berkeley connection properties: version, home location, default container type, compression algorithm, etc.

Actions Available at Container Level

In a Berkeley DB XML repository the actions available at container level in the **Data Source Explorer** view are the following:

-  **Add Resource** - Adds a new XML resource to the selected container.
- **Rename** - Allows you to specify a new name for the selected container.
-  **Delete** - Removes the selected container from the database tree.
- **Edit indices** - Allows you to edit the indices for the selected container.

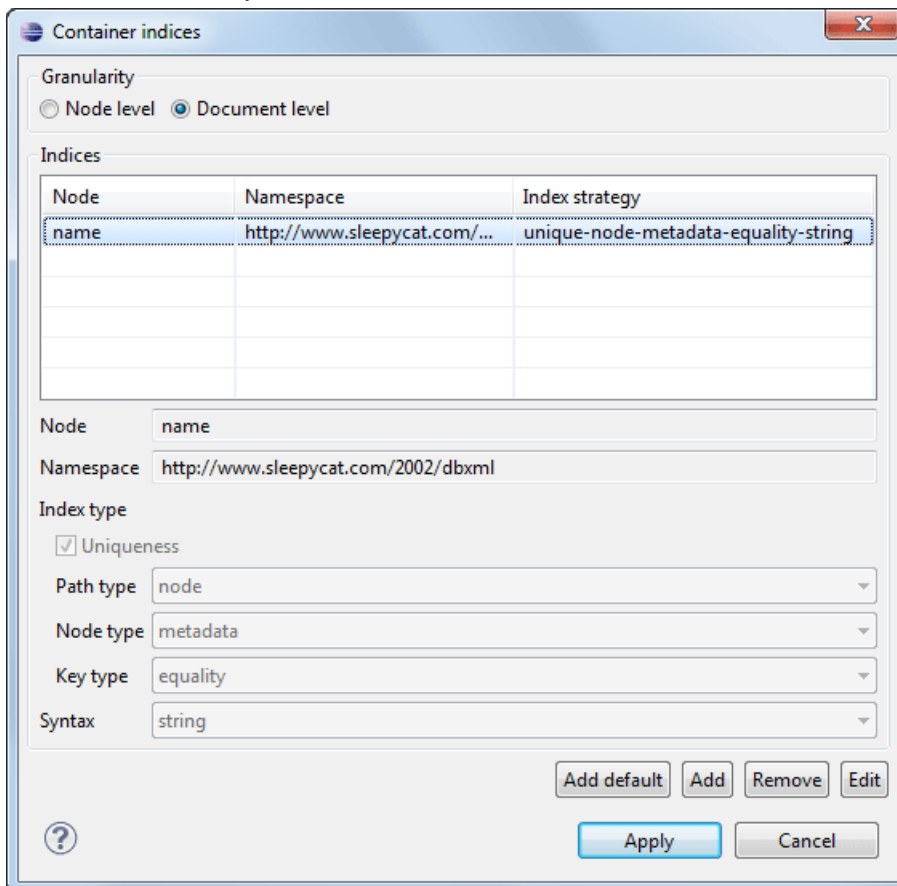






Figure 182: Container indices

The fields of the dialog are the following:

- Granularity:
 - **Document level** granularity is good for retrieving large documents.
 - **Node level** granularity is good for retrieving nodes from within documents.
- Add / Edit indices:
 - **Node** - The node name.
 - **Namespace** - The index namespace
 - Index strategy:
 - **Index type:**
 - **Uniqueness** - Indicates whether the indexed value must be unique within the container
 - **Path type:**
 - **node** - Indicates that you want to index a single node in the path
 - **edge** - Indicates that you want to index the portion of the path where two nodes meet
 - **Node type:**
 - **element** - An element node in the document content.
 - **attribute** - An attribute node in the document content.
 - **metadata** - A node found only in a document's metadata content.
 - **Key type:**
 - **equality** - Improves the performances of tests that look for nodes with a specific value
 - **presence** - Improves the performances of tests that look for the existence of a node regardless of its value
 - **substring** - Improves the performance of tests that look for a node whose value contains a given substring
 - **Syntax types** - The syntax describes what sort of data the index will contain and is mostly used to determine how indexed values are compared.
-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Properties** - Displays a dialog with a list of properties of the Berkeley container like: container type, auto indexing, page size, validate on load, compression algorithm, number of documents, etc.

Actions Available at Resource Level

In a Berkeley DB XML repository the actions available at resource level in the **Data Source Explorer** view are the following:



-  **Refresh** - Performs a refresh of the selected resource.
-  **Open** - Opens the selected resource in the editor.
- **Rename** - Allows you to change the name of the selected resource.
- **Move** - Allows you to move the selected resource in a different container in the database tree (also available through drag and drop).
-  **Delete** - Removes the selected resource from the container.
- **Copy location** - Allows you to copy to clipboard an application specific URL for the resource which can then be used for various actions like opening or transforming the resources.

eXist Connection

This section explains the actions that are available on an eXist connection.


Actions Available at Connection Level


For an eXist database the actions available at connection level in the **Data Source Explorer** view are the following:



-  **Configure Database Sources** - Opens the **Data Sources preferences page** where you can configure both data sources and connections.
- **Disconnect** - Closes the current database connection.
-  **Refresh** - Performs a refresh of the selected node's subtree.

Actions Available at Container Level

For an eXist database the actions available at container level in the **Data Source Explorer** view are the following:



- **New File** - Creates a file in the selected container
- **New Collection** - Creates a collection
- **Import Folders** - Adds recursively the content of specified folders from the local filesystem
-  **Import Files** - Adds a set of XML resources from the local filesystem
- **Cut** - Cuts the selected containers
- **Copy** - Copies the selected containers


 **Note:** You can add or move resources into container by dragging them from Project view, the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) or from another database container.


- **Paste** - Paste resources into selected container
- **Rename** - Allows you to change the name of the selected collection
-  **Delete** - Removes the selected collection
-  **Refresh** - Performs a refresh of the selected container
- **Properties** - Allows the user to view various useful properties associated with the container, like: name, creation date, owner, group, permissions.

Actions Available at Resource Level

For an eXist database the actions available at resource level in the **Data Source Explorer** view are the following:

-  **Refresh** - Performs a refresh of the selected resource.
-  **Open** - Opens the selected resource in the editor.
- **Rename** - Allows you to change the name of the selected resource.
- **Cut** - Cuts the selected resources
- **Copy** - Copies the selected resources.

 **Note:** You can add or move resources into container by dragging them from Project view, the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) or from another database container.

-  **Delete** - Removes the selected resource from the collection.
- **Copy location** - Allows you to copy to clipboard an application-specific URL for the resource which can then be used for various actions like opening or transforming the resources.
- **Properties** - Allows the user to view various useful properties associated with the resource.
- **Save As** - Allows you to save the name of the selected binary resource as a file on disk.

MarkLogic Connection



The resource management for a MarkLogic database can be done through WebDAV. For this a WebDAV URL must be configured in *the MarkLogic connection*. The actions that can be performed on MarkLogic resources through WebDAV are the same used for a WebDAV connection (see more about this in *WebDAV Connection* section).

Software AG Tamino Connection

This section explains the actions that are available on a Tamino connection.





Actions Available at Connection Level

For a Tamino database the actions available at connection level in the **Data Source Explorer** view are the following:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Disconnect** - Closes the current database connection.
-  **Configure Database Sources** - Opens the **Data Sources** [preferences page](#) where you can configure both data sources and connections.
- **Add container** - Allows you to create a new collection in the database.




Actions Available at Collection Level

For every new Tamino collection created in the **Data Source Explorer** view, you can specify if a schema is *required*, *optional* or *prohibited*. The following actions are available at collection level:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Filter ...** - An XQuery expression can be specified for filtering the nodes displayed in the selected Tamino container. It is only possible to specify one predicate. In the XQuery syntax a predicate is enclosed in square brackets. The square brackets, however, must not be specified in the dialog box displayed by this action. Only the predicate must be specified and it will be applied on the selected document type. For example: `name/surname between 'B' , 'C'`
-  **Insert XML instance** - Allows you to load a new XML document.
-  **Insert non XML instance** - Allows you to load a non XML document.
- **Modify Collection Properties** - Allows you to change the schema usage for the selected collection to optional. This action is available on collections with required and prohibited schema usage.
- **Define schema** - Allows you to add a new schema in the Schema Repository. This action is available on collections with optional and required schema usage.
-  **Delete** - Removes the selected collection. If it is a Tamino doctype then the action removes all the XML instances contained in the document type.
- **Set default** - Sets this collection as the default collection for running queries with the `input ()` function.




Actions Available at Schema Level

For a Tamino database the actions available at schema level in the **Data Source Explorer** view are the following:

-  **Refresh** - Performs a refresh of the selected schema.
-  **Open** - Opens the selected schema in the editor. There are supported schema changes that preserve the validity relative to the existent instances.
-  **Delete** - Removes the selected schema from the Schema Repository.

Actions Available at Resource Level

For a Tamino database the actions available at resource level in the **Data Source Explorer** view are the following:

-  **Refresh** - Performs a refresh of the selected resource.
-  **Open** - Opens the selected resource in the editor.
- **Rename** - Allows you to change the name of the selected resource.
-  **Delete** - Removes the selected resource.
- **Copy location** - Allows you to copy to clipboard an application specific URL for the resource which can then be used for various actions like opening or transforming the resources.
- **Properties** - Allows the user to view various useful properties associated with the resource.
- **Save As** - Allows you to save the name of the selected binary resource as a file on disk.





Validation of an XML resource stored in a Tamino database is done against the schema associated with the resource in the database.

Documentum xDb (X-Hive/DB) Connection

This section explains the actions that are available on a Documentum xDb (X-Hive/DB) 10 connection.


Actions Available at Connection Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at connection level in the **Data Source Explorer** view are the following:

-  **Refresh** - Performs a refresh of the selected node's subtree.
- **Disconnect** - Closes the current database connection.
-  **Configure Database Sources** - Opens the **Data Sources preferences page** where you can configure both data sources and connections.
- **Add library** - Allows you to add a new library.
-  **Insert XML Instance** - Allows you to add a new XML resource directly into the database root. See [Documentum xDb \(X-Hive/DB\) 10 Parser Configuration](#) for more details.
-  **Insert non XML Instance** - Allows you to add a new non XML resource directly into the database root.
- **Properties** - Displays the connection properties.




Actions Available at Catalog Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at catalog level in the **Data Source Explorer** view are the following:

-  **Refresh** - Performs a refresh of the selected catalog.
- **Add AS models** - Allows you to add a new abstract schema model to the selected catalog.
- **Set default schema** - Allows you to set a default DTD to be used for parsing. It is not possible to set a default XML Schema.
- **Clear default schema** - Allows you to clear the default DTD. The action is available only if there is a DTD set as default.
- **Properties** - Displays the catalog properties.


Actions Available at Schema Resource Level




For a Documentum xDb (X-Hive/DB) 10 database the actions available at schema resource level in the **Data Source Explorer** view are the following:

-  **Refresh** - Performs a refresh of the selected schema resource.
-  **Open** - Opens the selected schema resource in the editor.
- **Rename** - Allows you to change the name of the selected schema resource.
- **Save As** - Allows you to save the selected schema resource as a file on disk.
-  **Delete** - Removes the selected schema resource from the catalog
- **Copy location** - Allows you to copy to clipboard the URL of the selected schema resource.
- **Set default schema** - Allows you to set the selected DTD to be used as default for parsing. The action is available only for DTD.
- **Clear default schema** - Allows you to unset the selected DTD. The action is available only if the selected DTD is the current default to be used for parsing.

Actions Available at Library Level



For a Documentum xDb (X-Hive/DB) 10 database the actions available at library level in the **Data Source Explorer** view are the following:


-  **Refresh** - Performs a refresh of the selected library.
- **Add library** - Adds a new library as child of the selected library.
- **Add local catalog** - Adds a catalog to the selected library. By default, only the root-library has a catalog, and all models would be stored there.


-  **Insert XML Instance** - Allows you to add a new XML resource to the selected library. See [Documentum xDb \(X-Hive/DB\) 10 Parser Configuration](#) for more details.
-  **Insert non XML Instance** - Allows you to add a new non XML resource to the selected library.
- **Rename** - Allows you to specify a new name for the selected library.
- **Move** - Allows you to move the selected library to a different one (also available through drag and drop).
-  **Delete** - Removes the selected library.
- **Properties** - Displays the library properties.

Actions Available at Resource Level

When an XML instance document is added For a Documentum xDb (X-Hive/DB) 10 database the actions available at resource level in the **Data Source Explorer** view are the following:

-  **Refresh** - Performs a refresh of the selected resource.
-  **Open** - Opens the selected resource in the editor.
- **Rename** - Allows you to change the name of the selected resource.
- **Move** - Allows you to move the selected resource in a different library in the database tree (also available through drag and drop).

 **Note:** You can copy or move resources by dragging them from another database catalog.

- **Save As** - Allows you to save the selected binary resource as a file on disk.
-  **Delete** - Removes the selected resource from the library.
- **Copy location** - Allows you to copy to clipboard the URL of the selected resource.
- **Add AS model** - Allows you to add an XML schema to the selected XML resource.
- **Set AS model** - Allows you to set an active AS model for the selected XML resource.
- **Clear AS model** - Allows you to clear the active AS model of the selected XML resource.
- **Properties** - Displays the resource properties. Available only for XML resources.

Validation of an XML resource stored in an Documentum xDb (X-Hive/DB) 10 database is done against the schema associated with the resource in the database.

Documentum xDb (X-Hive/DB) 10 Parser Configuration for Adding XML Instances

When an XML instance document is added to a Documentum xDb (X-Hive/DB) 10 connection or library it is parsed with an internal XML parser of the database server. The following options are available for configuring this parser:

- DOM Level 3 parser configuration parameters. More about each parameter can be found here: [DOM Level 3 Configuration](#).
- Documentum xDb (X-Hive/DB) 10 specific parser parameters (for more information please consult the Documentum xDb (X-Hive/DB) 10 manual):
 - **xhive-store-schema** - If checked, the corresponding DTD's or XML schemas are stored in the catalog during validated parsing.
 - **xhive-store-schema-only-internal-subset** - Stores only the internal subset of the document (not any external subset). This options modifies the **xhive-store-schema** one (only has a function when that parameter is set to true, and when DTD's are involved). Select this option this option if you only want to store the internal subset of the document (not the external subset).
 - **xhive-ignore-catalog** - Ignores the corresponding DTD's and XML schemas in the catalog during validated parsing.
 - **xhive-psvi** - Stores **psvi** information on elements and attributes. Documents parsed with this feature turned on, give access to **psvi** information and enable support of data types by XQuery queries.
 - **xhive-sync-features** - Convenience setting. With this setting turned on, parameter settings of `XhiveDocumentIf` are synchronized with the parameter settings of `LSParser`. Note that parameter settings **xhive-psvi** and **schema-location** are always synchronized.

Troubleshooting

Cannot save the file. DTD factory class `org.apache.xerces.impl.dv.dtd.DTDDVFactoryImpl` does not extend from `DTDDVFactory`

I am able to access my XML Database in the Data Source Explorer and open files for reading but when I try to save changes to a file, back into the database, I receive the following error: "Cannot save the file. DTD factory class `org.apache.xerces.impl.dv.dtd.DTDDVFactoryImpl` does not extend from `DTDDVFactory`." How can I fix this?

Answer:

xhive.jar contains a MANIFEST.MF with a classpath:

```
Class-Path: core/antlr-runtime.jar core/aspectjrt.jar core/fastutil-shrunked.jar
            core/google-collect.jar core/icu4j.jar core/lucene-regex.jar
core/lucene.jar
            core/serializer.jar core/xalan.jar core/xercesImpl.jar
```

Because the driver was configured to use `xhive.jar` directly from the *xDB* installation (where many other jars are located), `core/xercesImpl.jar` from the *xDB* installation directory is loaded even though it is not specified in the list of jars from the data source driver configuration (it is in the classpath from `xhive.jar`'s *MANIFEST.MF*). A simple workaround for this issue is to copy **ONLY** the jar files used in the driver configuration to a separate folder and configure the data source driver to use them from there.

XQuery and Databases

XQuery is a native XML query language which is useful for querying XML views of relational data to create XML results. It provides the mechanism to efficiently and easily extract information from Native XML Databases (NXD) and relational data as well. The following database systems supported in Oxygen XML Developer plugin offer XQuery support:

- *Native XML Databases:*
 - Berkeley DB XML
 - eXist
 - MarkLogic (validation support not available)
 - Software AG Tamino
 - Raining Data TigerLogic (validation support not available)
 - Documentum xDb (X-Hive/DB) 10
- *Relational Databases:*
 - IBM DB2
 - Microsoft SQL Server (validation support not available)
 - Oracle (validation support not available)

Build Queries With Drag and Drop From Data Source Explorer View

When a query is edited in the XQuery editor the XPath expressions can be composed quickly with drag and drop actions from the **Data Source Explorer** view to the editor panel.

1. *Configure the data source* to the relational database.
2. *Configure the connection* to the relational database.
3. Browse the connection in the **Data Source Explorer** view up to the table or column that you want to insert in the query.
4. Drag the table name or the column name to the XQuery editor panel.

5. Drop the table name / column name where the XPath expression is needed.

An XPath expression that selects the dragged name will be inserted in the XQuery document at caret position.


XQuery Transformation

XQuery is designed to retrieve and interpret XML data from any source, whether it is a database or a document. Data is stored in relational databases but often it is required that data is extracted and transformed as XML when interfacing to other components and services. Also, it is an XPath-based querying language supported by most NXD vendors. To perform a query you need an XQuery transformation scenario.

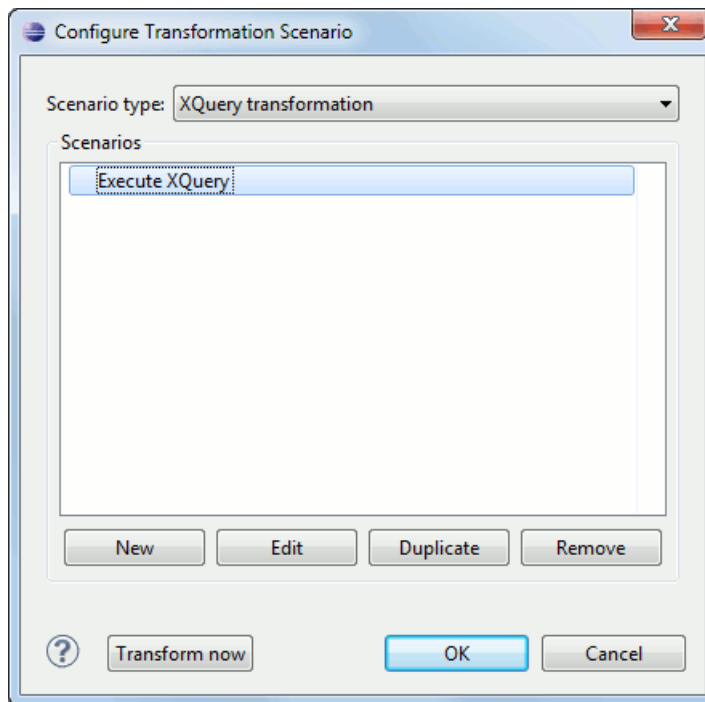
1. Configure a data source for the database.

The data source can be *relational* or *XML native*.

2. Configure an XQuery transformation scenario.

a) Click the  **Configure Transformation Scenario** toolbar button or go to menu **Document > Transformation > Configure Transformation Scenario**.

The dialog for configuring a scenario will be opened.



b) Click the **New** button of the dialog.

The dialog for editing an XQuery scenario will be opened.

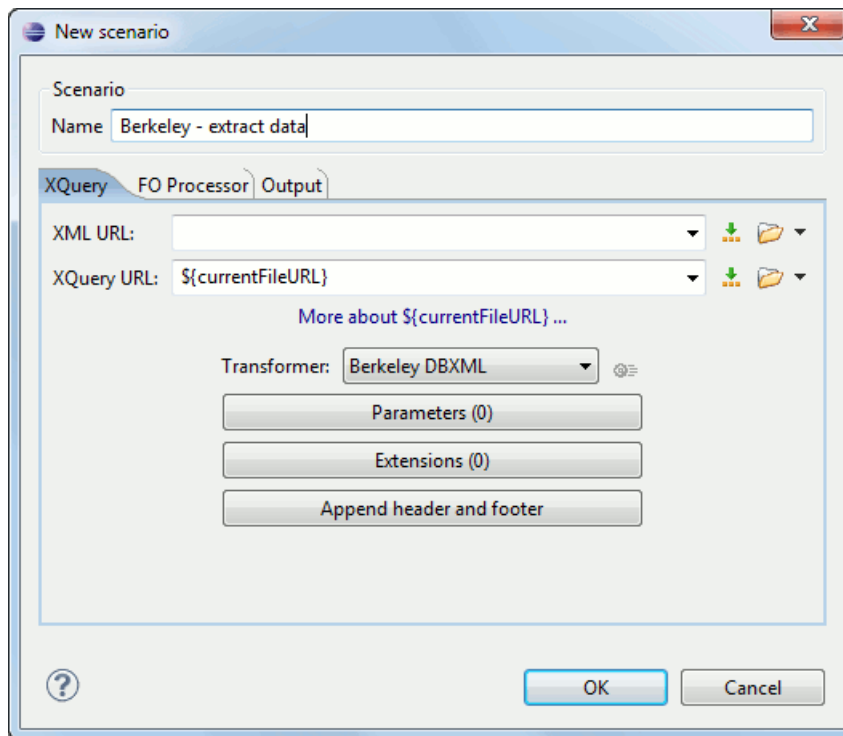



Figure 183: Edit Scenario Dialog

- c) Insert the scenario name in the dialog for editing the scenario.
- d) Choose the database connection in the **Transformer** combo box.
- e) Configure any other parameters if necessary.

For an XQuery transformation the output tab has an option called **Sequence** which allows you to execute an XQuery in lazy mode. The amount of data extracted from the database is controlled from option *Size limit on Sequence view*. If you choose **Perform FO Processing** in the **FO Processor** tab, the **Sequence** option is ignored.

- f) Click the **OK** button to finish editing the scenario.

Once the scenario is associated with the XQuery file, the query can include calls to specific XQuery functions implemented by that engine. The available functions depend on the target database engine selected in the scenario. For example for eXist and Berkeley DB XML *the content completion assistant* lists the functions supported by that database engine. This is useful for inserting in the query only calls to the supported functions (standard XQuery functions or extension ones).

 **Note:** An XQuery transformation is executed against a Berkeley DB XML server as a transaction using the query transaction support of the server.

3. Run the scenario.

To view a more complex value returned by the query that cannot be displayed entirely in the XQuery query result table at the bottom of the Oxygen XML Developer plugin window, for example an XMLTYPE value or a CLOB value, do the following actions:

- right click on that table cell
- select the action **Copy cell** from the popup menu for copying the value in the clipboard
- paste the value where you need it, for example an opened XQuery editor panel of Oxygen XML Developer plugin .

XQuery Database Debugging

This section describes the procedures for debugging XQuery transformations that are executed against MarkLogic databases and Berkeley DB XML ones.

Debugging with MarkLogic

To start a debug session against the MarkLogic engine you will first need to configure a [MarkLogic data source](#) and a [MarkLogic connection](#). Also you have to make sure that the debugging support is enabled in the MarkLogic server that will be accessed from Oxygen XML Developer plugin. On the server side debugging must be activated both in the XDBC server and in the section *Task Server* of the server control console (the switch *debug allow*) otherwise the error `DBG-TASKDEBUGALLOW` is reported by the MarkLogic server.

The MarkLogic XQuery debugger integrates seamlessly into the [XQuery Debugger perspective](#). If you already have a MarkLogic scenario configured for the XQuery file you can choose directly to [debug the scenario](#). If not, you just have to switch to the XQuery Debugger perspective, open the XQuery file in the editor and select the MarkLogic connection in the XQuery engine selector from the [debug control toolbar](#). For general information about how a debugging session is started and controlled see the [Working with the Debugger](#) section.




Peculiarities and Limitations of the MarkLogic Debugger

MarkLogic debugger has the following peculiarities and limitations:

- Debugging support is available only for MarkLogic server versions 4.0 or newer.
- For MarkLogic server versions 4.0 or newer there are three XQuery syntaxes which are supported: '0.9-ml' (inherited from MarkLogic 3.2), '1.0-ml' and '1.0'
- All the debugging steps are executed by the MarkLogic server and the results or possible errors of each step are presented by the local debugger user interface.
- All declared variables are presented as strings. The **Value** column of the **Variables** view contains the expression from the variable declaration. It can be evaluated by copying the expression with the **Copy value** action from the contextual menu of [the Variables view](#) and pasting it in [the XWatch view](#).
- No support for [Output to Source Mapping](#).
- No support for [showing the trace](#).
- [Breakpoints](#) can be set in the imported modules but they are only active if the modules are opened in the editor at the time of debugging.
- The modules can only be opened in the editor during the debugging session by stepping in repeatedly until reaching the module.
- There should not be any breakpoints set in modules from the same server which are not involved in the current debugging session.
- No support for [profiling](#) when an XQuery transformation is executed in the debugger.

Debugging Queries Which Import Modules

When debugging queries on a MarkLogic database which import modules stored in the database the recommended steps for placing a breakpoint in a module are the following:

1. Start the debugging session with the action  **Debug Scenario** from the **Transformation** toolbar or the  **XQuery Debugger** toolbar button.
2.  **Step into** repeatedly until reaching the desired module.
3. Add the module to the current [project](#) for easy access.
4. Set breakpoints in the module as needed.
5. [Continue debugging](#) the query.

When starting a new debugging session make sure that the modules which you will debug are already opened in the editor. This is necessary so that the breakpoints in modules will be considered. Also make sure there are no other opened modules which are not involved in the current debugging session.

Debugging with Berkeley DB XML

The Berkeley DB XML database added a debugging interface starting with version 2.5. The current version is 2.5.13 and it is supported in Oxygen XML Developer plugin 's XQuery Debugger. *The same restrictions and peculiarities* apply for the Berkeley debugger as for the MarkLogic one.

WebDAV Connection

This section explains how to work with a WebDAV connection in the **Data Source Explorer** view.

How to Configure a WebDAV Connection

Oxygen XML Developer plugin 's default configuration already contains a WebDAV data source called **WebDAV (S)FTP**. Based on this data source you can create a WebDAV connection for browsing and editing data from a database that provides a WebDAV interface. The connection will be available in *the Data Source Explorer view*. The steps for configuring a WebDAV connection are the following:




1. Go to menu **Preferences > Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the WebDAV data sources in the **Data Source** combo box.
5. Fill-in the connection details:
 - a) Set the URL to the WebDAV repository in the field **WebDAV URL**.
 - b) Set the user name to access the WebDAV repository in the field **User**.
 - c) Set the password to access the WebDAV repository in the field **Password**.
6. Click the **OK** button.

WebDAV Connection Actions

This section explains the actions that are available on a WebDAV connection in the **Data Source Explorer** view.




Actions Available at Connection Level

The contextual menu of a WebDAV connection in the **Data Source Explorer** view contains the following actions:

-  **Configure Database Sources** - Opens the **Data Sources preferences page** where you can configure both data sources and connections.
-  **Add Resource ...** - Allows you to add a new file on the server.
- **Add Container ...** - Allows you to create a new folder on the server.
-  **Refresh** - Performs a refresh of the connection.





Actions Available at Folder Level

The contextual menu of a folder node in a WebDAV connection in the **Data Source Explorer** view contains the following actions:

- **Add Container** - Allows you to create a new folder on the server.
-  **Add Resource** - Allows you to add a new file on the server in the current folder.
- **Rename** - Allows you to change the name of the selected folder.
- **Move** - Allows you to move the selected folder in a different location in the tree (also available through drag and drop).
-  **Delete** - Removes the selected folder.
-  **Refresh** - Performs a refresh of the selected node's subtree.

Actions Available at File Level

The contextual menu of a file node in a WebDAV connection in the **Data Source Explorer** view contains the following actions:

-  **Open** - Allows you to open the selected file in the editor.
- **Unlock** - Removes the lock from the current file in the database.
- **Rename** - Allows you to change the name of the selected file.
- **Move** - Allows you to move the selected file in a different location in the tree (also available through drag and drop).
-  **Delete** - Removes the selected file.
- **Copy Location** - Allows you to copy to clipboard an application specific URL for the resource which can then be used for various actions like opening or transforming the resources.
-  **Refresh** - Performs a refresh of the selected node.
-  **Properties** - Displays the properties of the current file in a dialog.

Chapter 13

Importing Data

Topics:

- [Introduction](#)
- [Import from Database](#)
- [Import from MS Excel Files](#)
- [Import from HTML Files](#)
- [Import from Text Files](#)

This chapter shows you how to import data stored in text format, Excel sheet or relational database tables into XML documents.

Introduction

Computer systems and databases contain data in incompatible formats and one of the most time-consuming activities has been to exchange data between these systems. Converting the data to XML can greatly reduce complexity and create data that can be read by many 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.

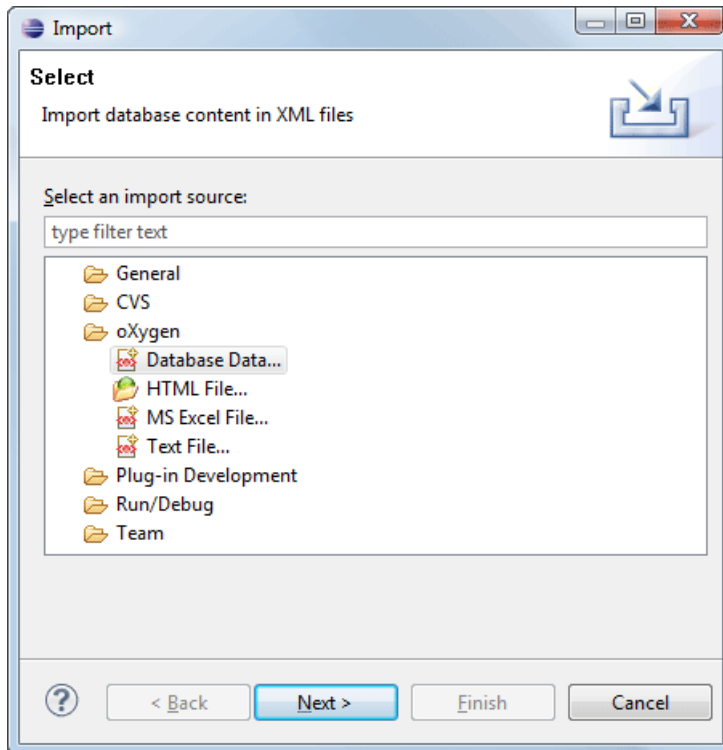


Figure 184: The Import Wizards of the Oxygen XML Developer plugin Plugin

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 will open a dialog with all the defined database connections:

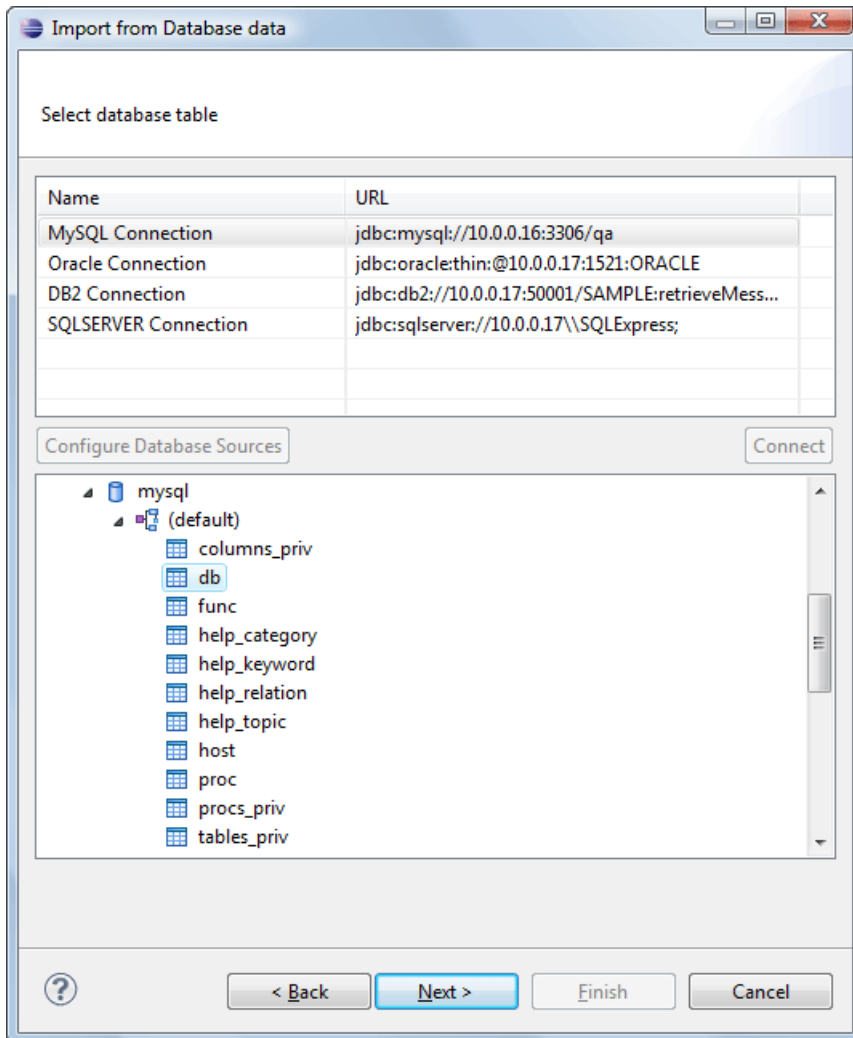


Figure 185: Import From Database Data Wizard

2. Select the connection to the database that contains the data.
Only connections configured on relational data sources can be used to import data.
3. If you want to edit, delete or add a data source or connection click on the **Configure Database Sources** button.
The **Preferences/Data Sources** option page will be 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 will open next, with a default query string in the **SQL Query** pane:

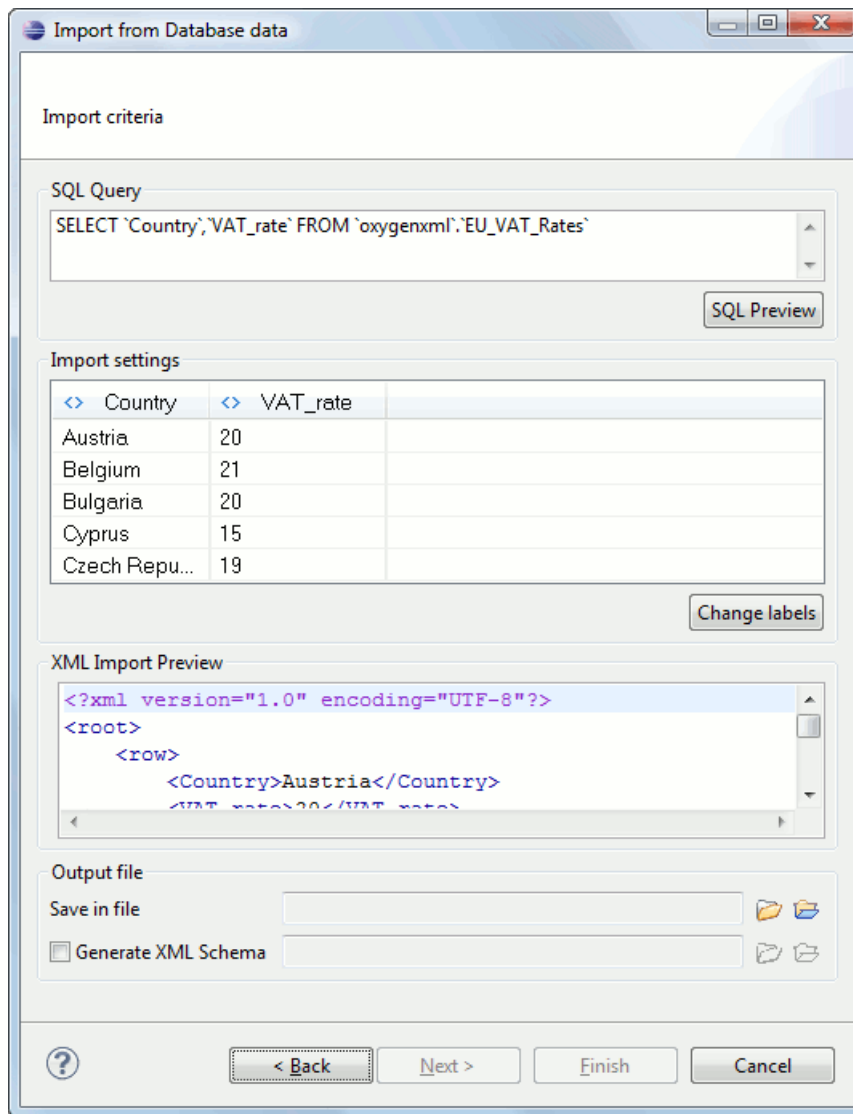


Figure 186: Import from Database Criteria Dialog

The dialog contains the following items:

- **SQL Preview** - If the **SQL Preview** button is pressed, it shows the labels that will be used in the XML document and the first 5 lines from the database into the **Import settings** panel. All data items in the input will be converted by default to element content, but this can be overridden by clicking on the individual column headers. Clicking once on a column header (ex **Heading0**) will cause the data from this column to be used as attribute values of the row elements. Click a second time and the column's data will be ignored when generating the XML file. You can cycle through these three options by continuing to click on the column header. If the data column will be converted to element content, the header will contain the <> symbols. If the data column will be converted to attribute content, the header will contain the = symbol, and if it will be skipped, the header will contain an x.
- **Change labels** - This button opens a new dialog, allowing you to edit the names of the root and row elements, change the XML name and the conversion criterion. The XML names can be edited by double-clicking on the desired item and entering the required label. The conversion criterion can also be modified by selecting from the drop-down list **ELEMENT**, **ATTRIBUTE** or **SKIPPED**.
- **Save in file** - If checked, the new XML document will be saved at the specified path.

👉 **Note:** If only **Open in editor** is checked, the newly created document will be opened in the editor, but as an unsaved file.

- **Generate XML Schema** - Allows you to specify the path of the generated XML Schema file.

7. Click the **SQL Preview** button.

The **SQL Query** string is editable. You can specify which fields should be taken into consideration.

If the query string represents a join operation of two or more tables and columns selected from different tables have the same name you should use aliases for them, like the following example. This will avoid the confusion of two columns being mapped to the same name in the result document of the importing operation.

```
select s.subcat_id,
       s.nr as s_nr,
       s.name,
       q.q_id,
       q.nr as q_nr,
       q.q_text
from faq.subcategory s,
     faq.question q
where ...
```

The input data will be displayed in a tabular form in the **Import Settings** panel. The **XML Import Preview** panel will contain an example of what the generated XML will look like.

Convert Table Structure to XML Schema

The structure of a table from a relational database can be imported in Oxygen XML Developer plugin as an XML Schema. This feature is activated by the **Generate XML Schema** checkbox from the **Import criteria** dialog used in [the procedure for importing table data](#) as an XML instance document.

Import from MS Excel Files

Oxygen XML Developer plugin can also import MS (Microsoft) Excel files into XML format documents. The required steps are:

1. Go to menu **File > Import > MS Excel File...** .
The **Select Excel Sheet** dialog will be opened.
2. Enter the URL of the Excel document in the opened dialog.
3. Choose one of the available sheets of the Excel document.
The input data is displayed in the **Import Criteria** dialog in a tabular form and the **XML Import Preview** contains an example of what the generated XML will look like. The **Import Criteria** dialog has a similar behaviour with the one shown in case of [Import from text files](#).
4. Click the **OK** button.

Import from HTML Files

HTML is one of the formats that can be imported as an XML document. The steps needed are:

1. Go to menu **File > Import > oXygen > HTML File ...**
The **Import HTML** wizard is displayed.
2. Enter the URL of the HTML document.
3. Select the type of the result XHTML document:
 - XHTML 1.0 Transitional
 - XHTML 1.0 Strict
4. Click the **OK** button.

The resulted document will be an XHTML file containing a DOCTYPE declaration referring to the XHTML DTD definition on the Web. The parsed content of the imported file will be transformed to XHTML Transitional or XHTML Strict depending on what radio button the user chose when performing the import operation.

Import from Text Files

The steps for importing a text file into an XML file are the following:

1. Go to menu **File > Import > oXygen > Text File...**
The **Select text file** dialog will be displayed.
2. Select the URL of the text file.
3. Select the encoding of the text file.
4. Click the **OK** button.

The **Import Criteria** dialog will be displayed:

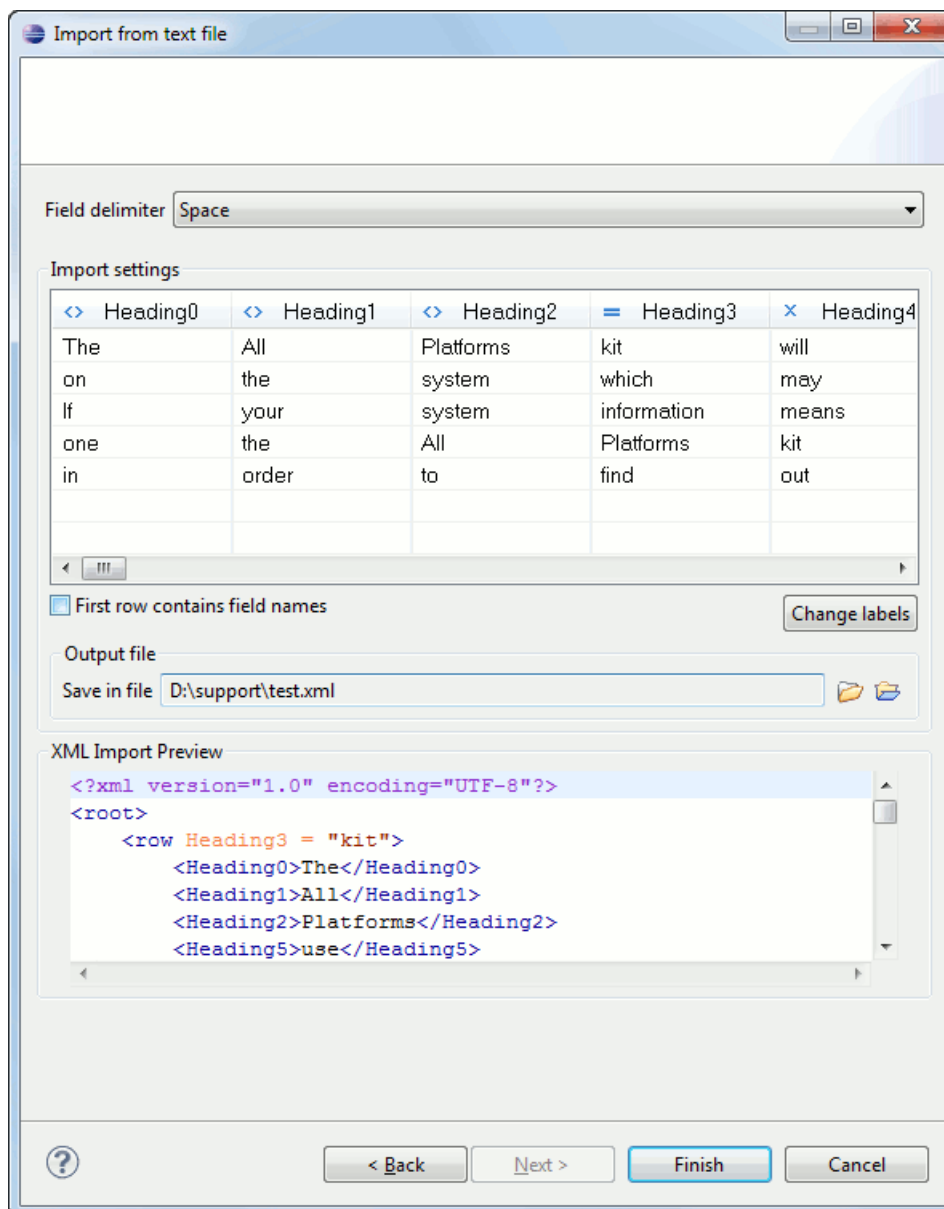


Figure 187: Import from text file

The input data is displayed in a tabular form. The **XML Import Preview** panel contains an example of what the generated XML document will look like. The names of the XML elements and the transformation of the first 5 lines from the text file are displayed in the **Import settings** section. All data items in the input will be converted by default to element content, but this can be over-ridden by clicking on the individual column headers. Clicking once on a column header will cause the data from this column to be used as attribute values of the row elements. Click the second time and the column's data will be ignored when generating the XML file. You can cycle through these three options by continuing to click on the column header. If the data column will be converted to element content, the header will contain the <> symbols. If the data column will be converted to attribute content, the header will contain the = symbol. If it will be skipped, the header will contain an *x*.

5. Select the field delimiter for the import settings:

- comma
- semicolon
- tab
- space

6. Set other optional settings of the conversion.

The dialog offers the following settings:

- **First row contains field names** - If the option is checked, you'll notice that the table has moved up. The default column headers are replaced (where such information is available) by the content of the first row. In other words, the first row is interpreted as containing the field names. The changes are also visible in the preview of the XML document. To return to default settings (where the first row is interpreted as containing data and not fields names), simply uncheck the option.
- **Change labels** - This button opens a new dialog, allowing you to edit the names of the root and row elements, change the XML name and the conversion criterion.

The XML names can be edited by double-clicking on the desired item and entering the required label. The conversion criterion can also be modified by selecting one of the drop-down list options: **ELEMENT**, **ATTRIBUTE** or **SKIPPED**.

- **Output file** - Allows you to select the output XML file.

Chapter 14

Content Management System (CMS) Integration

Topics:

- [Integration with Documentum \(CMS\)](#)

This chapter explains how Oxygen XML Developer plugin can be integrated with a content management system (CMS) so that the data stored in the CMS can be edited directly in the Oxygen XML Developer plugin editor. Only the integration with the Documentum (CMS) is explained .

Integration with Documentum (CMS)

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 or later with *Documentum Foundation Services 6.5* or later installed.



Attention:

It is recommended to use the latest 1.6.x Java version. It is possible that the Documentum (CMS) support will not work properly if you use other Java versions.

Configure Connection to Documentum Server

This section explains how to configure a connection to a Documentum server.

How to Configure a Documentum (CMS) Data Source

To configure a Documentum (CMS) data source you need the Documentum Foundation Services Software Development Kit (*DFS SDK*) corresponding to your server version. The *DFS SDK* can be found in the Documentum (CMS) server installation kit or it can be downloaded from [EMC Community Network](#).



Note: The *DFS SDK* can be found in the form of an archive named, for example, *emc-dfs-sdk-6.5.zip* for Documentum (CMS) 6.5.

1. Go to menu **Preferences > Data Sources**.
The **Preferences** dialog is opened at the **Data Sources** panel.
2. In the **Data Sources** panel click the **New** button.
3. Enter a unique name for the data source.
4. Select **Documentum (CMS)** from the driver type combo box.
5. Press the **Choose DFS SDK Folder** button.
6. Select the folder where you have unpacked the *DFS SDK* archive file.

If you have indicated the correct folder the following Java libraries (jar files) will be added to the list (some variation of the library names is possible in future versions of the *DFS SDK*):

- lib/java/emc-bpm-services-remote.jar
- lib/java/emc-ci-services-remote.jar
- lib/java/emc-collaboration-services-remote.jar
- lib/java/emc-dfs-rt-remote.jar
- lib/java/emc-dfs-services-remote.jar
- lib/java/emc-dfs-tools.jar
- lib/java/emc-search-services-remote.jar
- lib/java/ucf/client/ucf-installer.jar
- lib/java/commons/*.jar (multiple jar files)
- lib/java/jaxws/*.jar (multiple jar files)
- lib/java/utills/*.jar (multiple jar files)



Note: If for some reason the jar files are not found, you can add them manually by using the **Add Files** and **Add Recursively** buttons and navigating to the lib/java folder from the *DFS SDK*.

7. Click the **OK** button to finish the data source configuration.

How to Configure a Documentum (CMS) Connection

The steps for configuring a connection to a Documentum (CMS) server are the following:

1. Go to menu **Preferences > Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the previously configured Documentum (CMS) data sources in the **Data Source** combo box.
5. Fill-in the connection details:
 - **URL** - The URL to the Documentum (CMS) server: `http://<hostname>:<port>`
 - **User** - The user name to access the Documentum (CMS) repository.
 - **Password** - The password to access the Documentum (CMS) repository.
 - **Repository** - The name of the repository to log into.
6. Click the **OK** button to finish the configuration of the connection.

Known Issues

The following are known issues with the Documentum (CMS):

1. Please note that at the time of this implementation there is a problem in the UCF Client implementation for MAC OS X which prevents you from viewing or editing XML documents from the repository. The UCF Client is the component responsible for file transfer between the repository and the local machine. This component is deployed automatically from the server.
2. In order for the Documentum driver to work faster, you need to specify to the JVM to use a weaker random generator, instead of the very slow native implementation. This can be done by modifying in the Oxygen 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.

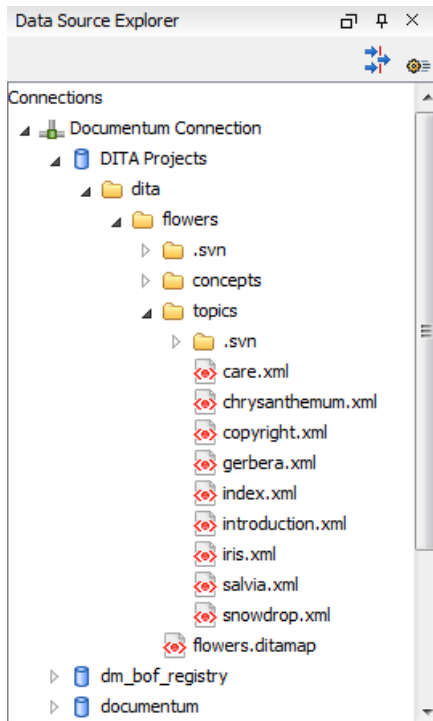




Figure 188: Browsing a Documentum repository



Actions Available on Connection

The actions available on a Documentum (CMS) connection in the **Data Source Explorer** view are the following:

-  **Configure Database Sources** - Opens the *Data Sources preferences page* where you can configure both data sources and connections.
- **New Cabinet** - Creates a new cabinet in the repository. The cabinet properties are:
 - **Type** - The type of the new cabinet (default is **dm_cabinet**).
 - **Name** - The name of the new cabinet.
 - **Title** - The title property of the cabinet.
 - **Subject** - The subject property of the cabinet.
-  **Refresh** - Refreshes the connection.

Actions Available on Cabinets / Folders

The actions available on a Documentum (CMS) cabinet in the **Data Source Explorer** view are the following:

-  **New Folder** - Creates a new folder in the current cabinet / folder. The folder properties are the following:
 - **Path** - Shows the path where the new folder will be created.
 - **Type** - The type of the new folder (default is **dm_folder**).
 - **Name** - The name of the new folder.
 - **Title** - The title property of the folder.
 - **Subject** - The subject property of the folder.
-  **New Document** - Creates a new document in the current cabinet / folder. The document properties are the following:
 - **Path** - Shows the path where the new document will be created.
 - **Name** - The name of the new document.
 - **Type** - The type of the new document (default is **dm_document**).

- **Format** - The document content type format.
- **Import** - Imports local files / folders in the selected cabinet / folder of the repository. Actions available in the import dialog:
 - **Add Files** - Shows a file browse dialog and allows you to select files to add to the list.
 - **Add Folders** - Shows a folder browse dialog that allows you to select folders to add to the list. The subfolders will be added recursively.
 - **Edit** - Shows a dialog where you can change the properties of the selected file / folder from the list.
 - **Remove** - Removes the selected files / folders from the list.
- **Rename** - Changes the name of the selected cabinet / folder.
- **Copy** - Copies the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop while holding the **(Ctrl)** key pressed.
- **Move** - Moves the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop.
- **✕ Delete** - Deletes the selected cabinet / folder from the repository. The following options are available:
 - **Folder(s)** - Allows you to delete only the selected folder or to delete recursively the folder and all subfolders and objects.
 - **Version(s)** - Allows you to specify what versions of the resources will be deleted.
 - **Virtual document(s)** - Here you can specify what happens when virtual documents are encountered. They can be either deleted either by themselves or together with their descendants.
- **🔄 Refresh** - Performs a refresh of the selected node's subtree.
- **📄 Properties** - Displays the list of properties of the selected cabinet / folder.

Actions Available on Resources

The actions available on a Documentum (CMS) resource in the **Data Source Explorer** view are the following:

- **📄 Edit** - Checks out (if not already checked out) and opens the selected 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 for viewing.
- **Open with** - Opens the selected resource in the specified editor / tool for viewing.
- **Check Out** - Checks out the selected resource from the repository. The action is not available if the resource is already checked out.
- **Check In** - Checks in the selected resource (commits changes) into the repository. The action is only available if the resource is checked out.

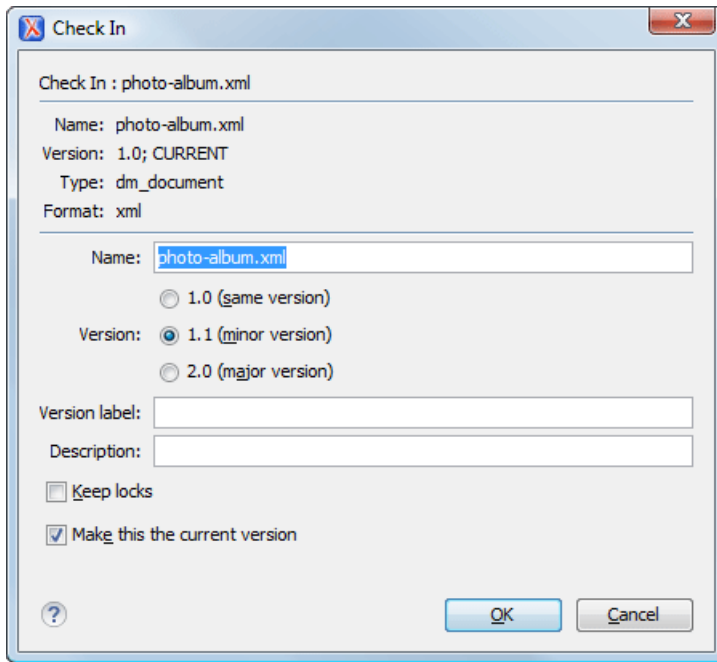


Figure 189: Check In Dialog

The properties of a resource are the following:

- **Name** - The name which the resource will have on 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** - If checked the updated resource is checked into the repository but it is also kept checked out in your name.
 - **Make this the current version** - Makes the updated resource the current version (will have the *CURRENT* version label).
- **Cancel Checkout** - Cancels the check out and loses all modifications since the check out. Action is only available if the 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 to a different location in the tree. Action is not available on virtual document descendants. This action can also be performed with drag and drop while holding the **(Ctrl)** key pressed.
 - **Move** - Moves the selected resource to 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.

Chapter 15

Composing Web Service Calls

Topics:

- [Overview](#)
- [Composing a SOAP Request](#)
- [Testing Remote WSDL Files](#)
- [The UDDI Registry Browser](#)
- [Generate WSDL Documentation](#)

This chapter covers the following topics:

- compose a SOAP request based on a WSDL file;
- send the request to a server;
- generate HTML documentation for WSDL files.

Overview

Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

The WSDL files contain information about the published services, like the name, the message types, and the bindings. Oxygen XML Developer plugin is offering a way to edit the WSDL files that is similar to editing XML, with content completion assistant and validation driven by a mix of WSDL and SOAP schemas. Oxygen XML Developer plugin supports WSDL version 1.1 and 2.0 and SOAP versions 1.1 and 1.2. That means that in the location where a SOAP extension can be inserted the content completion assistant offers elements from both SOAP 1.1 and SOAP 1.2. Validation of SOAP requests is executed first against a SOAP 1.1 schema and 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).

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

To design, compose, and test Web service calls in Oxygen XML Developer plugin follow the procedure:

1. *Create a new document* or open an existing document of type WSDL.
2. Design the Web Service descriptor in the WSDL editor.

The *content completion* is driven by a mix of the WSDL and SOAP schemas. You do not need to specify the schema location for the WSDL standard namespaces because Oxygen XML Developer plugin comes with these schemas and uses them by default to assist the user in editing Web Service descriptors.

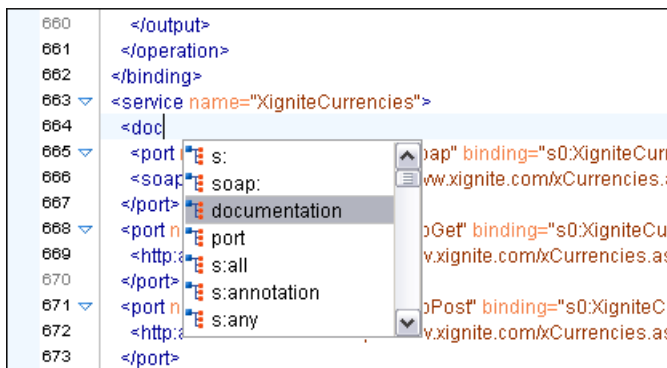


Figure 190: Content completion for WSDL documents

3. While editing the Web-Services descriptors *check their conformance* to the WSDL and SOAP schemas.

In the following example you can see how the errors are reported.

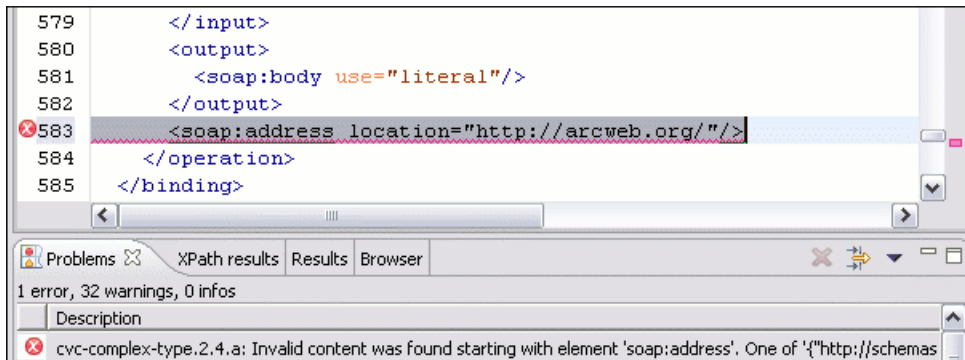


Figure 191: Validating a WSDL file

4. Check if the defined messages are accepted by the Web Services server.

Oxygen XML Developer plugin is providing two ways of testing, one for the currently edited WSDL file and other for the remote WSDL files that are published on a web server. For the currently edited WSDL file the WSDL SOAP Analyser tool can be opened by:

- pressing the toolbar button  **WSDL SOAP Analyser**
- going to the menu item **WSDL > WSDL SOAP Analyser**
- going to submenu of the **Project** view contextual menu

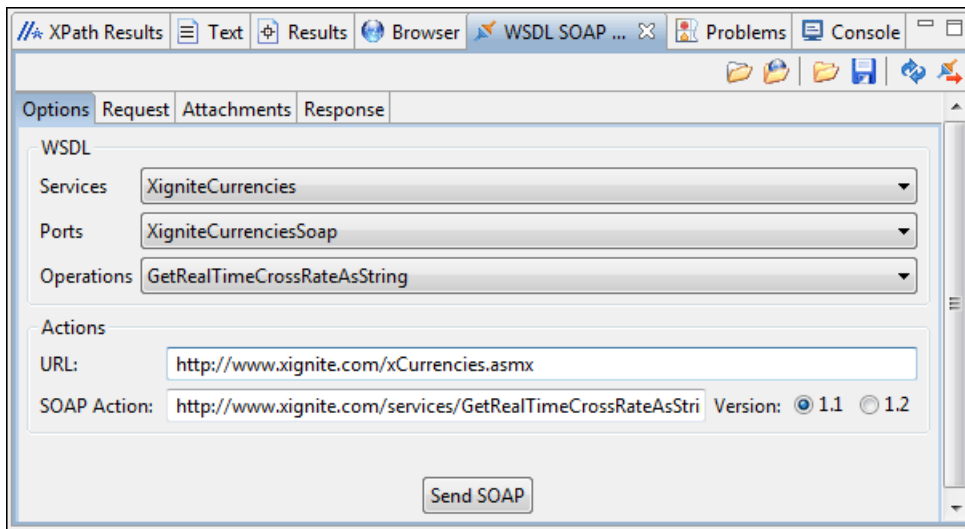


Figure 192: WSDL SOAP Analyser

This dialog contains a SOAP analyser and sender for Web Services Description Language file types. The analyser fields are:

- **Services** - The list of services defined by the WSDL file.
- **Ports** - The ports for the selected service.
- **Operations** - The list of available operations for the selected service.
- **Action URL** - Shows the script that serves the operation.
- **SOAP Action** - Identifies the action performed by the script.
- **Version** - Choose between 1.1 and 1.2. The SOAP version is selected automatically depending on the selected port.
- **Request Editor** - It allows you to compose the web service request. When an action is selected, Oxygen XML 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 is shown 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.

5. Save the request derived from the Web Service descriptor.

Once defined, a request derived from a Web Service descriptor can be saved with the **Save** button to a Web Service SOAP Call (WSSC) file for later reuse. In this way, you save time in configuring the URLs and parameters.

6. Open the result of a Web Service call in an editor panel.

In this way, you can save the SOAP request or process it further.

Testing Remote WSDL Files

To open and test a remote WSDL file the steps are the following:

1. Go to menu **Window > Show View > Other > oXygen > WSDL SOAP Analyser ...**
2. Press the **Choose WSDL** button and enter the URL of the remote WSDL file.


You enter the URL:

- by typing
- by browsing the local file system
- by browsing a remote file system
- by browsing [a UDDI Registry](#)

3. Press the **OK** button.

This will open the **WSDL SOAP Analyser** tool. In the **Saved SOAP Request** tab you can open directly a previously saved Web Service SOAP Call (WSSC) file thus skipping the analysis phase.

The UDDI Registry Browser

Pressing the  button in the **WSDL File Opener** dialog (menu **Tools > WSDL SOAP Analyser**) opens the **UDDI Registry Browser** dialog.

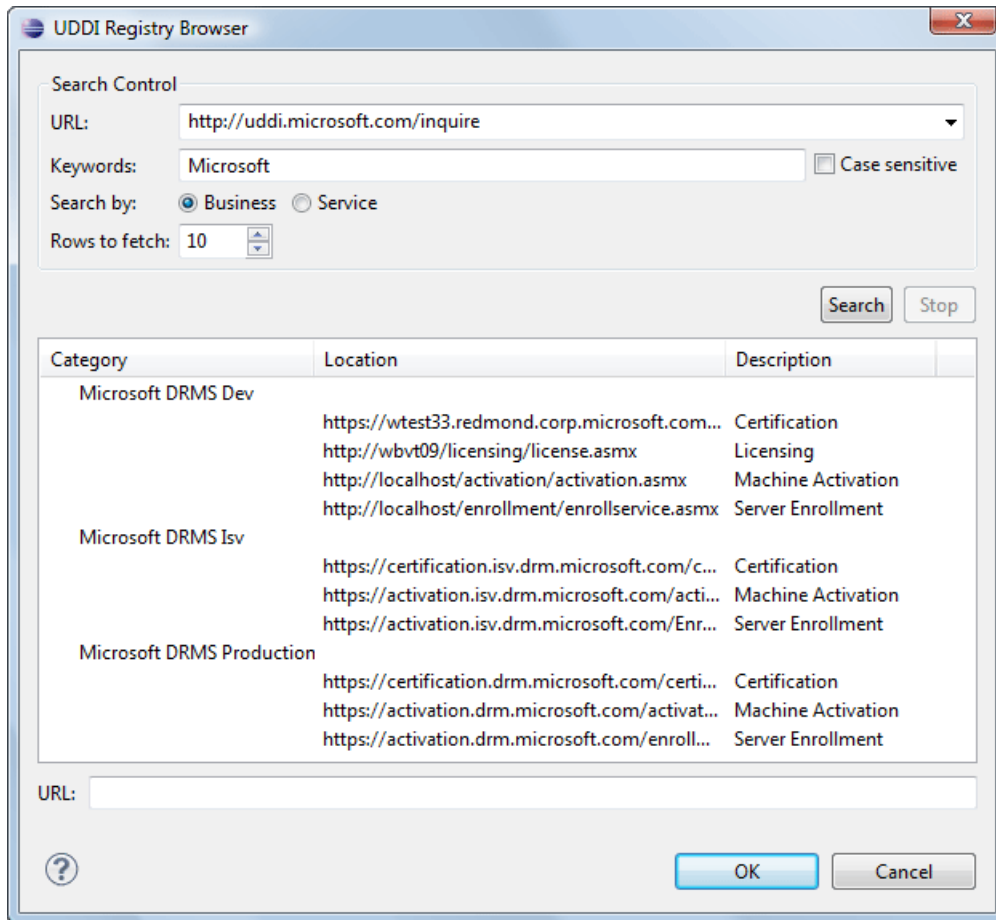


Figure 193: UDDI Registry Browser dialog

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.

Generate WSDL Documentation

To generate documentation for a WSDL document use the action from menu **XML Tools > Generate Documentation > WSDL Documentation**.

The WSDL documentation dialog can be also opened from the **Navigator** contextual menu: **Generate WSDL Documentation**.

The fields of the dialog are the following:

- **Input URL** - Type the URL of the file or click on the browse button and select it from the file system.

- **Output file (HTML)** - In this field you will have to enter the path and the filename where the documentation will be generated.
- **Open in Browser/System Application** - Select this option if you want the result to be opened in the system application associated with that file type.
 - 👉 **Note:** If you already set the **Default Internet browser** option in the **Global** preferences page, it will take precedence over the default system application settings.
- **Generate** - Press this button for generating the documentation of the WSDL file.

Chapter 16

Digital Signatures

Topics:

- [Overview](#)
- [Canonicalizing Files](#)
- [Certificates](#)
- [Signing Files](#)
- [Verifying the Signature](#)

This chapter explains how to apply and verify digital signatures on XML documents.

Overview

Digital signatures are widely used as security tokens, not just in XML. A digital signature provides a mechanism for assuring integrity of data, the authentication of its signer, and the non-repudiation of the entire signature to an external party:

- A digital signature must provide a way to verify that the data has not been modified or replaced to ensure integrity.
- The signature must provide a way to establish the identity of the data's signer for authentication.
- The signature must provide the ability for the data's integrity and authentication to be provable to a third party for non-repudiation.

A public key system is used to create the digital signature and it's also used for verification. The signature binds the signer to the document because digitally signing a document requires the originator to create a hash of the message and then encrypt that hash value with his own private key. Only the originator has that private key and he is the only one that can encrypt the hash so that it can be unencrypted using his public key. The recipient, upon receiving both the message and the encrypted hash value, can decrypt the hash value, knowing the originator's public key. The recipient must also try to generate the hash value of the message and compare the newly generated hash value with the unencrypted hash value received from the originator. If the hash values are identical, it proves that the originator created the message, because only the actual originator could encrypt the hash value correctly.

XML Signatures can be applied to any digital content (data object), including XML (see W3C Recommendation, [XML-Signature Syntax and Processing](#)). An XML Signature may be applied to the content of one or more resources:

- enveloped or enveloping signatures are applied over data within the same XML document as the signature
- detached signatures are applied over data external to the signature element; the signature is "detached" from the content it signs. This definition typically applies to separate data objects, but it also includes the instance where the signature and data object reside within the same XML document but are sibling elements.

The XML Signature is a method of associating a key with referenced data. It does not normatively specify how keys are associated with persons or institutions, nor the meaning of the data being referenced and signed.

The original data is not actually signed. Instead, the signature is applied to the output of a chain of canonicalization and transformation algorithms, which are applied to the data in a designated sequence. This system provides the flexibility to accommodate whatever "normalization" or desired preprocessing of the data that might be required or desired before subjecting it to being signed.

To canonicalize something means to put it in a standard format that everyone generally uses. Because the signature is dependent on the content it is signing, a signature produced from a not canonicalized document could possibly be different from one produced from a canonicalized document. The canonical form of an XML document is physical representation of the document produced by the method described in this specification. The term canonical XML refers to XML that is in canonical form. The XML canonicalization method is the algorithm defined by this specification that generates the canonical form of a given XML document or document subset. The term XML canonicalization refers to the process of applying the XML canonicalization method to an XML document or document subset. XML canonicalization is designed to be useful to applications that require the ability to test whether the information content of a document or document subset has been changed. This is done by comparing the canonical form of the original document before application processing with the canonical form of the document result of the application processing.

A digital signature over the canonical form of an XML document or document subset would allow the signature digest calculations to be oblivious to changes in the original document's physical representation. During signature generation, the digest is computed over the canonical form of the document. The document is then transferred to the relying party, which validates the signature by reading the document and computing a digest of the canonical form of the received document. The equivalence of the digests computed by the signing and relying parties (and hence the equivalence of the canonical forms over which they were computed) ensures that the information content of the document has not been altered since it was signed.

The following canonicalization algorithms are used in Oxygen XML 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 them and the signature will be invalid.

Exclusive Canonicalization finds out what namespaces you are actually using (the ones that are a part of the XML syntax) and just copies those. It does not look into attribute values or element content, so the namespace declarations required to process these are not copied.

This type of canonicalization is useful when you have a signed XML document that you wish to insert into other XML documents and it will insure the signature verifies correctly every time, so it is required when you need self-signed structures that support placement within different XML contexts.

Inclusive Canonicalization is useful when it is less likely that the signed data will be inserted in other XML document and it's the safer method from the security perspective because it requires no knowledge of the data that are to be secured in order to safely sign them.

The canonicalization method can specify whether or not comments should be included in the canonical form output by the XML canonicalization method. If a canonical form contains comments corresponding to the comment nodes in the input node-set, the result is called canonical XML with comments. In an uncommented canonical form comments are removed, including delimiter for comments outside document element.

These three operations: Digital Signing, Canonicalization and Verification of the signature are available from the **Tools** menu or from the Editor's **contextual menu > Source**.

Canonicalizing Files

The user can select the canonicalization algorithm to be used for his document from the following dialog displayed by the action **Canonicalize** available from the editor panel's **contextual menu > Source**.

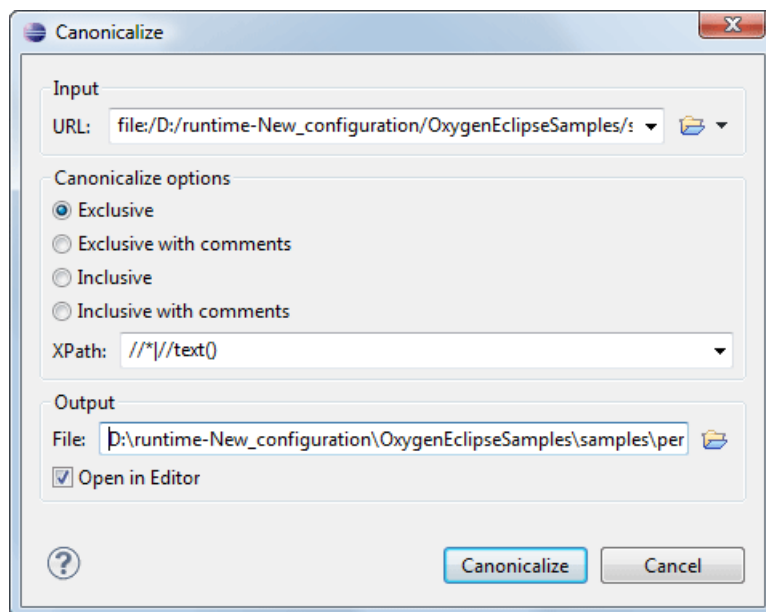


Figure 194: Canonicalization settings dialog

The fields of the dialog are the following:

- **URL** - Specifies the location of the input URL.
- **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.
- **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
- **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.

- **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.
- **XPath** - The XPath expression provides the fragments of the XML document to be signed.
- **Output** - Specifies the output file path where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in the editor.

Certificates

A certificate is a digitally signed statement from the issuer (an individual, an organization, a website or a firm), saying that the public key (and some other information) of some other entity has a particular value. When data is digitally signed, the signature can be verified to check the data integrity and authenticity. Integrity means that the data has not been modified. Authenticity means the data comes indeed from the entity that claims to have created and signed it. Certificates are kept in special repositories called keystores.

A *keystore* is an encrypted file that contains private keys and certificates. All keystore entries (key and trusted certificate entries) are accessed via unique aliases. An alias must be assigned for every new entry of either a key or certificate as a reference for that entity. No keystore can store an entity if its alias already exists in that keystore and no keystore can store trusted certificates generated with keys in its keystore.

In Oxygen XML 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 set the options for a certificate or to validate it, go to menu [Options > Preferences > Certificates](#).

Signing Files

The user can select the type of signature to be used for his document from the following dialog displayed by the action **Sign** available from the editor panel's **contextual menu > Source**

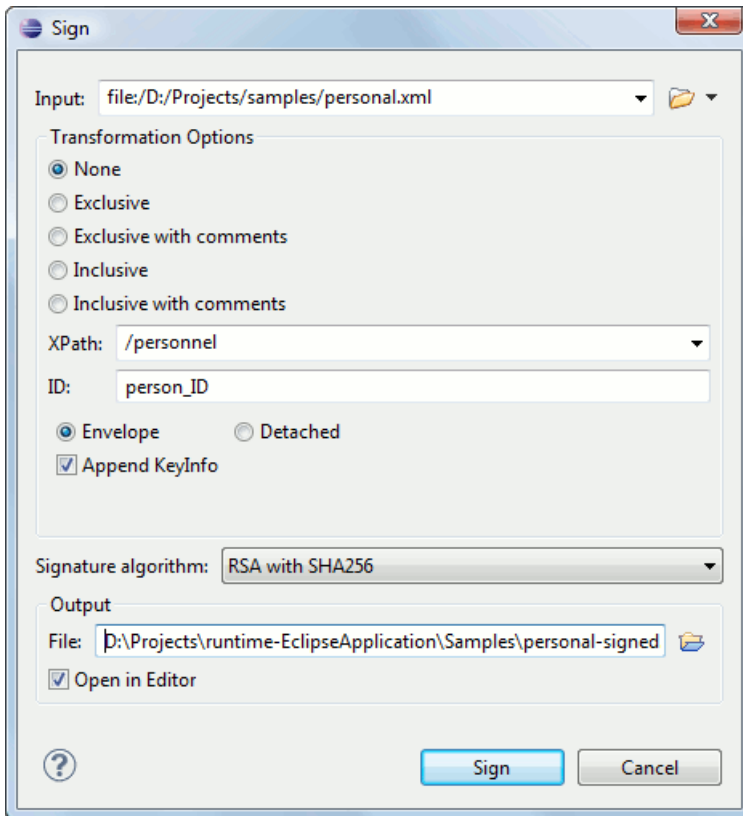


Figure 195: Signature settings dialog

The following options are available:

- **Input** - Specifies the location of the input URL.
- **None** - If selected, no canonicalization algorithm is used.
- **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.
- **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
- **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.
- **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.
- **XPath** - The XPath expression provides the fragments of the XML document to be signed.
- **ID** - Provides ID of the XML element to be signed.
- **Envelope** - If selected, the enveloping signature is used.
- **Detached** - If selected, the detached signature is used.
- **Append KeyInfo** - The element `ds:KeyInfo` will be added in the signed document only if this option is checked.
- **Signature algorithm** - Algorithm used for signing the document. The following options are available: **RSA with SHA1**, **RSA with SHA256**, **RSA with SHA384**, and **RSA with SHA512**.
- **Output** - Specifies the output file path where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in the editor.

Verifying the Signature

The user can select a file to verify its signature in the dialog opened by the action **Verify Signature** available from the editor panel's **contextual menu** > **Source**. The dialog has a field **URL** that specifies the location of the document for which to verify the signature.

If the signature is valid, a dialog displaying the name of the signer will be opened. If not, an error message will show details about the problem.

Chapter 17

Text Editor Specific Actions

Topics:

- [Finding and Replacing Text in the Current File](#)
- [Spell Checking](#)

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

Finding and Replacing Text in the Current File

This section explains how to use the find and replace features of the application.

The Find All Elements / Attributes Dialog

This dialog is opened from menu **Edit > Find All Elements...** . It assists you in defining XML elements / attributes search operations on the current document.

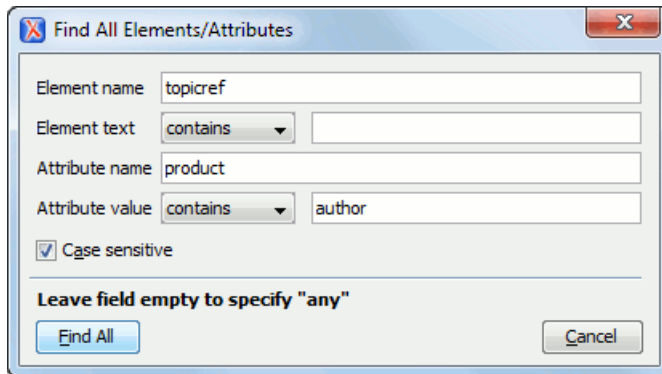


Figure 196: Find All Elements / Attributes dialog

The dialog can perform the following actions:

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

All these search criteria can be combined to fine filter your results.

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

The dialog fields are described as follows:

- **Element name** - The target element name to search for. Only the elements with this exact name are returned. For any element name just leave the field empty.
- **Element text** - The target element text to search for. The combo box beside this field allows you to specify that you are looking for an exact or partial match of the element text. For any element text, select **contains** in the combo box and leave the field empty.

If you leave the field empty but select **equals** in the combo box, only elements with no text will be found. Select **not contains** to find all elements which do not have the specified text inside.


- **Attribute name** - The name of the attribute which needs to be present in the elements. Only the elements which have an attribute with this name will be returned. For any / no attribute name just leave the field empty.
- **Attribute value** - The combo box beside this field allows you to specify that you are looking for an exact or partial match of the attribute value. For any / no attribute value select **contains** in the combo box and leave the field empty.

If you leave the field empty but select **equals** in the combo box, only elements that have at least an attribute with an empty value will be found.

Select **not contains** to find all elements which have attributes without a specified value.

- **Case sensitive** - When this option is checked, operations are case sensitive.

Spell Checking

The **Spelling** dialog enables you to check the spelling of the current document. It is opened from menu **XML > Check Spelling (Ctrl+Shift+Q)** or the toolbar button  **Check Spelling**.

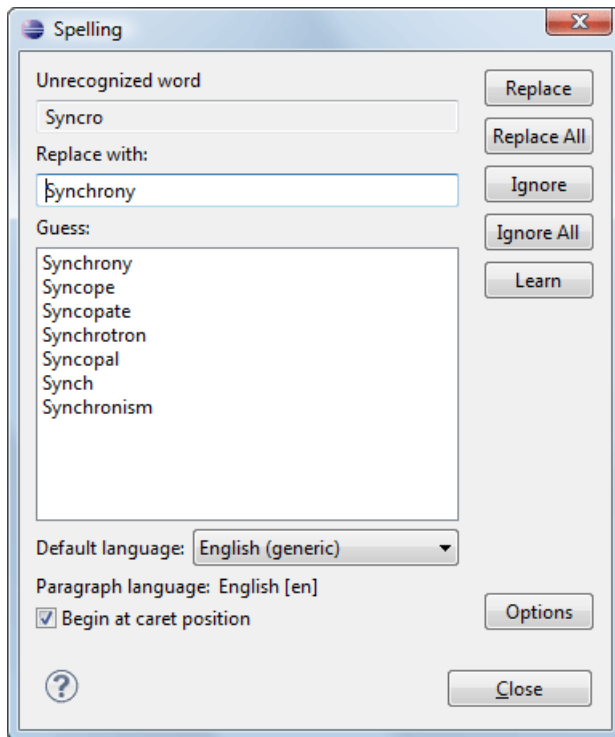


Figure 197: The Check Spelling Dialog

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 (old orthography)
- German (new orthography)
- Spanish

There is one dictionary for each language-country variant combination. If you cannot find a Hunspell dictionary that is already built for your language you can build such a dictionary. First you need a list of correct words which you want to check in your documents. You build a dictionary from this list following *these instructions*.

Adding a Dictionary for the Hunspell Checker

To add a new spelling dictionary to Oxygen XML Developer plugin or to replace an existing one you should follow these steps:

1. *Download the archive* with the files of your language dictionary.

A dictionary has two files with the same name and different extensions: a file with `.dic` extension and a file with `.aff` extension.

2. Copy the `.aff` and `.dic` files to the `spell` subfolder of the Oxygen XML Developer plugin preferences folder only if it is a new dictionary (it is not available as built-in dictionary in Oxygen XML Developer plugin).

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

- `C:\Documents and Settings\[LOGIN-USER-NAME]\Application Data` on Windows XP
- `C:\Users\[LOGIN-USER-NAME]\AppData\Roaming` on Windows Vista
- `[USER-HOME-FOLDER]/Library/Preferences` on Mac OS X
- `[USER-HOME-FOLDER]` on Linux

3. Copy the `.aff` and `.dic` files into the folder `[Oxygen-install-folder]/dicts` only if it is an existing dictionary.
4. Restart the application after copying the dictionary files.

Dictionaries for the Java Checker

A Java checker dictionary has the form of a `.dar` file located in the directory `[Oxygen-install-folder]/dicts`. Oxygen 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-install-folder]/dicts` and restarting Oxygen XML Developer plugin. There is one dictionary for each language-country variant combination. If you cannot find a dictionary that is already built for your language you can build such a dictionary with the tool available at <http://www.xmlmind.com/spellchecker/dictbuilder.shtml>.

Learned Words

Spell checker engines rely on dictionary to decide that a word is correctly spelled. To tell the spell checker engine that an unknown word is actually correctly spelled, you need to add that word to its dictionary. There are two ways to do so:

- press the **Learn** button from the **Spelling** dialog;
- invoke the contextual menu on an unknown word, then press **Learn word**.

Learned words are stored into a persistent dictionary file. Its name is composed of the currently checked language code and the `.tdi` extension, for example `en_US.tdi`. It is located in the folder:

- folder on Windows XP
- folder on Windows Vista
- folder on Mac OS X
- folder on Linux

To delete items from the list of learned words, press **Delete learned words** from *Spell Check* preferences page.

Ignored Words

The content of some XML elements like `programlisting`, `codeblock` or `screen` should always be skipped by the spell checking process. The skipping can be done manually word by word by the user using the **Ignore** button of *the Spelling dialog* or, more conveniently, automatically by maintaining a set of known element names that should never be checked. You maintain this set of element names *in the user preferences* as a list of XPath expressions that match the elements.

Only a small subset of XPath expressions is supported, that is only the `'/'` and `'//'` separators and the `'*'` wildcard. Two examples of supported expressions are `/a/*b` and `//c/d/*`.

Automatic Spell Check

To allow Oxygen 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:

- **Delete Repeated Word** action - allows you to delete repeated words;
- a list of words suggested by the spell checking engine as possible replacements of the unknown word;
- **Learn Word** action - allows you to add the current unknown word to the persistent dictionary.

Spell Checking in Multiple Files

The **Check Spelling in Files** action allows you to check the spelling on multiple local or remote documents. This action is available from:

- contextual menu of the **Project** view;
- contextual menu of the **DITA Maps Manager** view.

The results are *displayed in a view* that allows grouping the reported errors as a tree with two levels.

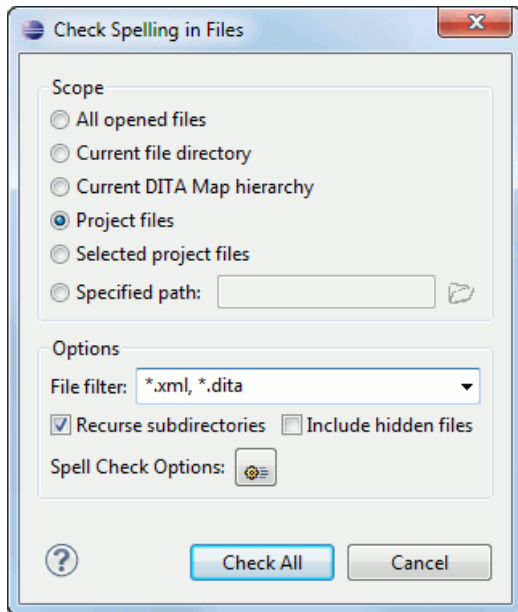


Figure 198: Check Spelling in Files Dialog

The following scopes are available:

- **All opened files** - Spell check in all opened files.
- **Directory of the current file** - All the files from the folder of the current edited file.
- **Project files** - All files from the current project.
- **Selected project files** - The selected files from the current project.
- **Specified path** - A custom path.

You can also choose a file filter, decide whether to recurse subdirectories or process hidden files.

The spell checker processor uses the options available in the [Spell Check preferences panel](#).

Chapter 18

Configuring the Application

Topics:

- [Configuring Options](#)
- [Importing / Exporting Global Options](#)
- [Preferences](#)
- [Reset Global Options](#)
- [Scenarios Management](#)
- [Editor Variables](#)

This chapter presents all the user preferences that allow you to configure the application and the editor variables that are available for customizing the user defined commands.

Configuring Options

The application is controlled by a set of options. There is an option for pretty much any of the Oxygen XML Developer plugin features. To offer you the highest degree of flexibility in customizing the application to fit the end-user's role in your organization, Oxygen XML Developer plugin comes with several distinct layers of option values, arranged here according to their priority, from low to high:

- *Default Options* - should be regarded as factory defaults or built-in values.

This predefined set of values are tuned to allow the application to behave optimally in most working environments.

- *Customized Default Options* - designed to customize the application initial option values for a group of users.

This layer allows an administrator to deploy oXygen preconfigured with a standardized set of option values.



Note: Once this layer is set, it represents the initial application state when an end-user presses the **Restore defaults** or **Reset Global Options** buttons.

- *Global Options* - allow individual users to personalize oXygen according to their specific needs.



Note: If you set a specific option in one of the layers, but it is not applied in the application, make sure that one of the higher priority layers is not overwriting it.

Customized Default Options

You can overwrite the *Default Options* values by imposing the loading of a set of options from a specific XML structure stored in one of the following file types:

- *options file* - containing all application options.

To create such a file, follow this procedure:

- set the values you want to impose as defaults in the [Preferences pages](#);
- run the action **Window > Preferences > Oxygen > Export Global Options**.

- *project file (.xpr file extension)* - containing only the options different from the *Default Options*.

To create such a file, follow this procedure:

- set the values of the default options in the Preferences dialog so that they are saved in the project file.

There are two ways of telling the application to use such an options configuration file to extract the customized default options:

- set the file path of the options configuration file as value of `com.oxygenxml.default.options` Java virtual machine parameter.

The file must be specified with an URL or a file path relative to the application installation folder. For example, add the following line in the `[Eclipse-platform-install-folder]/configuration/config.ini` file:

```
com.oxygenxml.default.options=@config.dir/./default-options.xml
```

- copy the options configuration file in the `[Eclipse-platform-install-folder]/plugins/com.oxygenxml.editor/preferences` folder.

Importing / Exporting Global Options

The following actions are available in the **Options** application menu:

- **Reset Global Options** - Restores the preferences set to *Customized Default Options layer*, or if this level is not defined, to *Default Options layer*.
- **Import Global Options** - Allows you to import a set of *Global Options* from an *options file*.
- **Export Global Options** - Allows you to export the entire set of *Global Options* in an options file.

In the Oxygen XML Developer plugin preferences page opened from menu **Window > Preferences > oXygen** you can find the import / export preferences buttons which allow you to move your global preferences in XML format from one computer to another.

Preferences

Once the application is installed you can use the **Preferences** dialog accessed from menu **Options > Preferences** to customize the application.

You can always revert modifications to their default values by pressing the **Restore Defaults** button, available in each options page.

If you don't know how to use a specific preference that is available in any **Preferences** panel or what effect it will have you can open a help page about the current panel at any time pressing the help button (?) located in the left bottom corner of the dialog or pressing the F1 key.

A restricted version of the **Preferences** dialog is available at any time in the editors of the Oxygen XML Developer plugin by right-clicking in the editor panel and selecting **Preferences**:

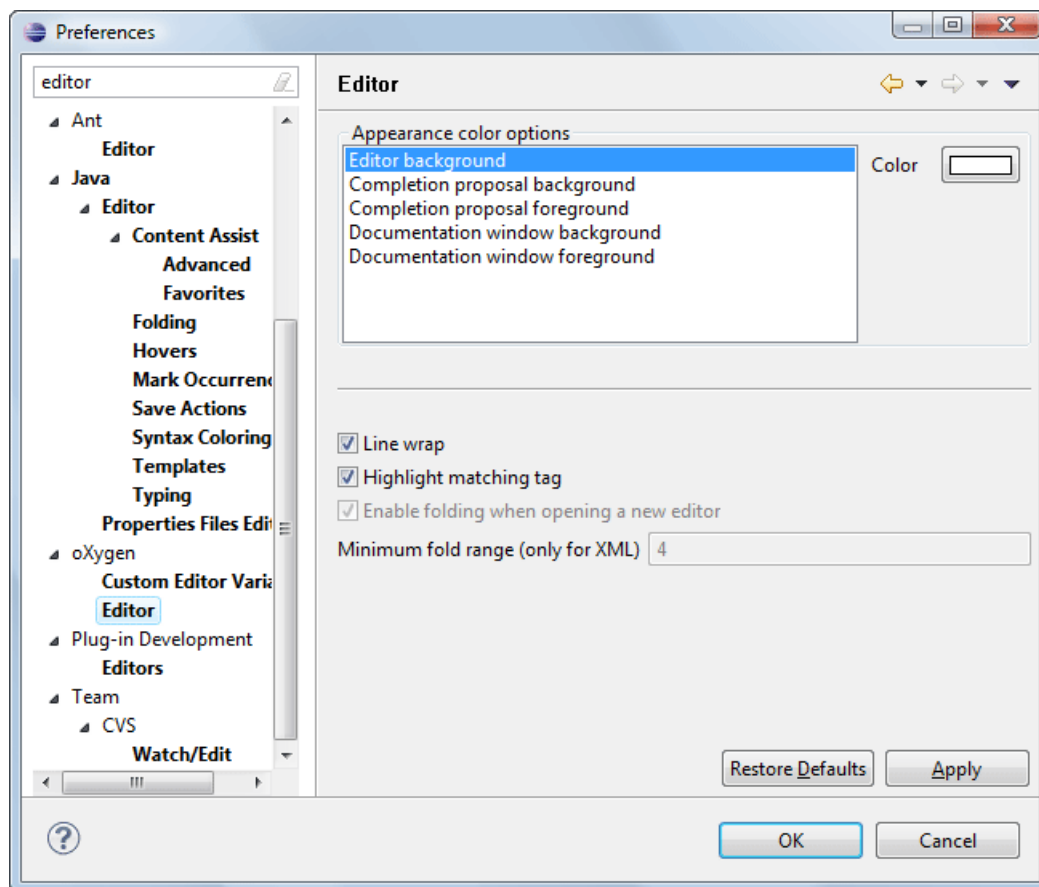


Figure 199: Eclipse Preferences dialog - restricted version

Oxygen XML Developer plugin License

The license information panel is opened from menu **Window > Preferences > oXygen**. This panel presents the data of the license key which enables the Oxygen XML Developer plugin: registration name, category and number of purchased licenses, encrypted signature of the license key. Clicking on the **Register** button opens the Oxygen XML Developer plugin **License** dialog that allows you to insert a new license key.

Global

The **Global** options panel is opened from menu **Window > Preferences > oXygen > Global**.

The following user options are available:

- **Show hidden files and directories** - Shows system hidden files and folders in the file browser dialog and the folder browser dialog. This setting is not available on Mac OS X.

Fonts

The **Fonts** preferences panel is opened from menu **Window > Preferences > oXygen > Fonts**.

The following options are available:


- **Text** - The family and size of the font used in text-based editors. There are two options available:
 - **Map to text font** - Uses the same font as the one set in **General / Appearance / Colors and Fonts** for the basic text editor.
 - **Customize** - Allows you to choose a specific font.
- - Allows you to specify a font to be used in Author mode.

Document Type Association


The **Document Type Association** preferences panel is opened from menu **Window > Preferences > oXygen > Document Type Association**.

Oxygen XML is highly customizable. Practically you can associate an entire class of documents (grouped logically by some common features like namespace, root element name or filename) to a bundle consisting of CSS stylesheets, validation schemas, catalog files, new files templates, transformation scenarios and even custom actions. The bundle is called *document type* and the association is called *Document Type Association*.

The following actions are available in this preferences panel:

- **Change framework directory location** - Specifies a from where Oxygen XML loads the document types.
 - **Document types table** - Presents the currently defined document type associations, ordered by priority and alphabetically. Each row of the table represents a document type association, each holding the following information:
 - **Document type** - Name of the document type.
 - **Enabled** - When checked, the corresponding document type association is enabled and analyzed when the application is trying to determine the type of an opened document.
 - **Storage** - Displays the type of location where the framework configuration file is stored. Can be one of **External** (framework configuration is saved in a file) or **Internal** (framework configuration is stored in the application's internal options).
-  **Note:** Please note that if you set the **Storage** to **Internal** and the document type association settings are already stored in a framework file, the file content is saved in application's internal options and the file is removed.
- **Priority** - Displays the framework priority. Can be one of: Lowest , Low, Normal, High, Highest. You can set a higher priority to Document Type Associations you want to be evaluated first.

When expanding a **Document Type Association** its defined rules are presented. A rule is described by:

- **Namespace** - Specifies the namespace of the root element from the association rules set (* (*any*) by default). If you want to apply the rule only when the root element is in no namespace, leave this field empty (remove the `ANY_VALUE` string).
 - **Root local name** - Specifies the local name of the root element (* (*any*) by default).
 - **File name** - Specifies the name of the file (* (*any*) by default).
 - **Public ID** - Represents the Public ID of the matched document.
 - **Java class** - Presents the name of the Java class which is used to determine if a document matches the rule.
 - **New** - Opens a dialog box that allows you to add a new association.
 - **Edit** - Opens a new dialog allowing you to edit an existing association.
-  **Note:** If you try to edit an existing association type when you have no write permissions to its store location, a dialog box will be shown, asking if you want to duplicate the document type.

- **Delete** - Deletes the selected associations.
- **Enable DTD/XML Schema processing in document type detection** - When this option is enabled, the matching process also examines the DTD/XML Schema associated with the document. For example, the fixed attributes declared in the DTD for the root element are also analyzed, if this is specified in the association rules.

This is especially useful if you are writing DITA customizations. DITA topics and maps are also matched by looking for the `DITAArchVersion` attribute of the root element. This attribute is specified as `default` in the DTD and it is detected in the root element, helping Oxygen XML 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.

Editor

The **Editor** preferences panel is opened from menu **Window > Preferences > oXygen > Editor**.

Use these options to configure the visual aspect of the text editor. The same options panel is available in [the restricted version of the Preferences dialog](#).


The following options are available:

- **Editor background** - Background color for both text editor and Diff Files editors.
- **Completion proposal background** - Background color of the content completion assistant window.
- **Completion proposal foreground** - Foreground color of the content completion assistant window.
- **Documentation window background** - Background color of the window containing documentation for the elements suggested by the content completion assistant.
- **Documentation window foreground** - Foreground color for the window containing documentation for the elements suggested by the content completion assistant.
- **Line wrap** - Enables *soft wrap* of long lines, that is automatically wrap lines in edited documents. The document content is unaltered as the application does not use newline characters to break long lines.
- **Highlight matching tag** - If you place the cursor on a start or end tag, Oxygen XML highlights the corresponding member of the pair. You can also customize the highlight color.
- **Display quick assist notification icon** - Displays the *Quick Assist* yellow bulb icon in the editor line number stripe.
- **Enable folding when opening a new editor** - Enables *folding support* when you open a new editor pane.
- **Minimum fold range (only for XML)** - If **Enable folding when opening a new editor** option is checked, you can specify the minimum number of lines in a block for which the folding support becomes active. If you modify this value, the change takes effect next time you open / reopen the editor.

Print

The **Print** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Edit modes** and allows you to customize the information printed on the header and footer of a page. These settings do not apply to the **Grid** and **Schema Diagram** modes.

You can customize the information printed in the one-row header and footer of a page. The following options are available:

- **Left, Middle** and **Right** area of the header and footer. Here you can write a pattern of the text printed in the header and footer of the page. You can use the following editor variables to write the text pattern:
 - `${currentFileURL}` - Current file as URL, that is the absolute file path of the current edited document represented as URL.
 - `${cfne}` - Current file name with extension.
 - `${cp}` - Current page number.
 - `${tp}` - Total number of pages in the document.
 - `${env(VAR_NAME)}` - Value of the `VAR_NAME` environment variable.
 - `${system(var.name)}` - Value of the `var.name` system variable.
 - `${date(pattern)}` - Current date. Follows the given pattern. Example: yyyy-MM-dd.
-  **Note:** As an example, to show the current page number against the total number of pages in the top right corner of the page, write the following pattern in the **Right** text area of the **Header** section: `${cp} from ${tp}`.
- **Color** - Text color.
- **Font** - Font type. Default is `SansSerif`.
- **Underline/Overline** - When selected, a separator line is drawn between the header/footer and the page content.

Edit modes

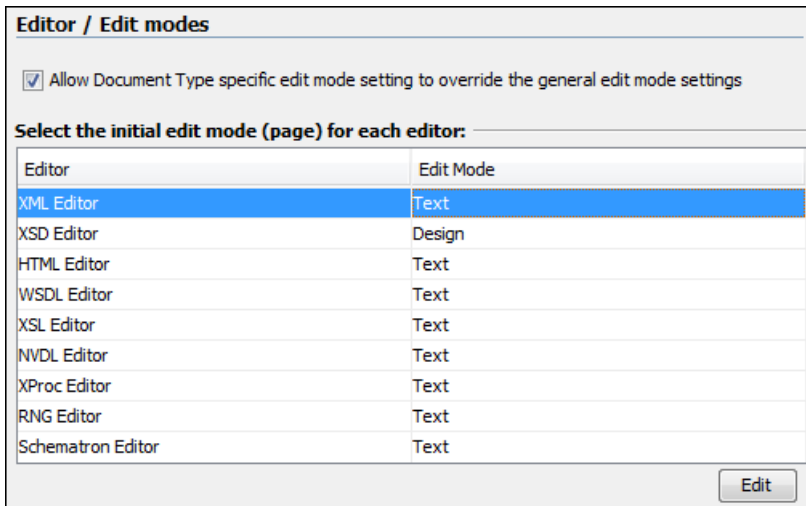
The **Edit modes** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Edit modes** and allows you to select the initial edit mode for an editor. The mode in which a file was edited in the previous session is saved and is used when the application is restarted and the file reopened.

If the checkbox **Allow Document Type specific edit mode setting to override the general mode setting** is checked, the initial edit mode setting from overrides the general edit mode setting from the table below.

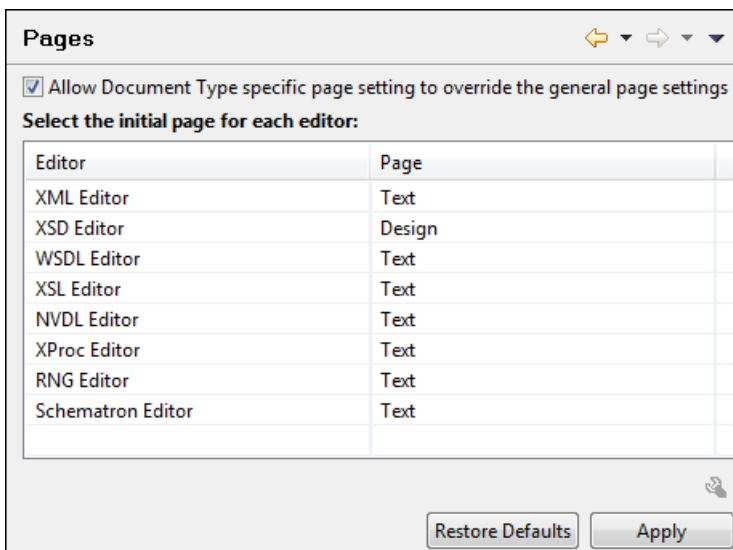
The initial edit mode of each editor type has one of the following values:

- Text
- Grid
- Design (available only for the W3C XML Schema editor)

The Oxygen XML Developer plugin Edit modes Preferences Panel





The Oxygen XML Developer plugin Edit modes Preferences Panel



Text Diagram

The **Diagram** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Edit modes > Text / Diagram**.

- Show Full Model XML Schema diagram** - If this option is enabled the *old synchronized version* of the schema diagram is available in the Text mode for XML Schemas. For editing in the schema diagram, you can use the *new schema diagram* editor mode.
 -  **Note:** When handling very large schemas, the Schema Diagram might affect the performance of your system. In such cases, disabling the schema diagram view improves the speed of navigation through the edited schema.
 -  **Note:** This option is unchecked by default.
- Enable Relax NG diagram and related views** - This option enables the Relax NG schema diagram and synchronization with the related views (**Attributes, Model, Elements, Outline**).
- Show Relax NG diagram** - Displays the Relax NG schema diagram in **Full Model View** and **Logical Model View**.
- Enable NVDL diagram and related views** - This option enables the NVDL schema diagram and synchronization with the related views (**Attributes, Model, Elements, Outline**).
- Show NVDL diagram** - Displays the NVDL schema diagram in **Full Model View** and **Logical Model View**.

- **Location relative to editor** - Sets the location of the schema diagram panel in the editing relative to the diagram **Text** editor.

Grid

The **Grid** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Edit modes > Grid**.

The following preferences are available:

- **Compact representation** - If checked, a more *compact representation* of the grid is used: a child element is displayed at the same height level with the parent element.



Note: In the *non-compact representation*, a child element is presented nested with one level in the parent container, one row lower than the parent.

- **Format and indent when passing from grid to text or on save** - The content of the document is formatted by applying the *Format and Indent* action on every switch from the **Grid** editor to the **Text** editor of the same document.
- **Default column width (characters)** - The default width in characters of a table column of the grid. A column can hold:
 - element names;
 - element text content;
 - attribute names;
 - attribute values.

If the total width of the grid structure is too large you can resize any column by dragging the column margins with the mouse pointer, but the change is not persistent. To make it persistent, set the new column width with this option.

- **Active cell color** - Background color for the active cell of the grid. There is only one active cell at a time. The keyboard input always goes to the active cell and the selection always contains it.
- **Selection color** - Background color for the selected cells of the grid except the active cell.
- **Border color** - The color used for the lines that separate the grid cells.
- **Background color** - The background color of grid cells that are not selected.
- **Foreground color** - The text color of the information displayed in the grid cells.
- **Row header colors - Background color** - The background color of row headers that are not selected.
- **Row header colors - Active cell color** - The background color of the row header cell that is currently active.
- **Row header colors - Selection color** - The background color of the header cells corresponding to the currently selected rows.
- **Column header colors - Background color** - The background color of column headers that are not selected.
- **Column header colors - Active cell color** - The background color of the column header cell that is currently active.
- **Column header colors - Selection color** - The background color of the header cells corresponding to the currently selected columns.

The column headers are painted with two color gradients, one for the upper 1/3 part of the header and the other for the lower 2/3 part. The start and end colors of the first gradient are set with the first two color buttons. The start and end colors of the second gradient are set with the last two color buttons.

Schema Design

The **Schema Design** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Edit modes > Schema Design**

The following options are available:

- **Show annotation in the diagram** - Displays the content of `xs:documentation` elements in the XML Schema **Design** view.
- **When trying to edit components from another schema** - Specifies the default behavior when you try to edit a component from an imported schema. You can choose between:
 - **Always go to its definition** - Edits the component definition in the imported file.
 - **Never go to its definition** - Edits the component definition in the current file.

- **Always ask** - You are always prompted to choose where you want to edit the component definition.

Properties

You can decide to show additional properties for XML Schema components in the XML Schema **Design** view and customize the properties displayed for each schema component. For each component properties, you can decide if you want to display them only when having a specified value or all the time.

Text Diagram

The following options are available:

- **Show Full Model XML Schema diagram** - If this option is enabled the *old synchronized version* of the schema diagram is available in the Text mode for XML Schemas. For editing in the schema diagram, you can use the *new schema diagram* editor mode.
 - 👉 **Note:** When handling very large schemas, the Schema Diagram might affect the performance of your system. In such cases, disabling the schema diagram view improves the speed of navigation through the edited schema.
 - 👉 **Note:** This option is unchecked by default.
- **Enable Relax NG diagram and related views** - This option enables the Relax NG schema diagram and synchronization with the related views (**Attributes, Model, Elements, Outline**).
- **Show Relax NG diagram** - Displays the Relax NG schema diagram in **Full Model View** and **Logical Model View**.
- **Enable NVDL diagram and related views** - This option enables the NVDL schema diagram and synchronization with the related views (**Attributes, Model, Elements, Outline**).
- **Show NVDL diagram** - Displays the NVDL schema diagram in **Full Model View** and **Logical Model View**.
- **Location relative to editor** - Sets the location of the schema diagram panel in the editing relative to the diagram **Text** editor.

Format

The **Format** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Format**.

The following options are available:

- **Detect indent on open** - The editor detects the indent settings of the open XML document. This way you can correctly format (pretty-print) files that were created with different settings, without changing your options every time you edit such a file. Besides, you can activate the option for detecting the maximum line width used by the formatting and hard wrap mechanism. These features were designed to minimize the differences created by the **Format and Indent** operation when working with a versioning system, like CVS for example.
 - 👉 **Note:** If the document contains different-size indents, the application computes a weighted average value.
- **Indent with tabs** - When checked, sets the indent size to a *tab* unit. When unchecked, the application uses space characters to form an indent. The number of space characters that form a *tab* is defined by the **Indent size** option.
- **Indent size** - Sets the number of space characters or the tab size that equals a single indent. An *indent* can be a number of spaces or a tab, selectable using the **Indent With Tabs** option. For example, if set to 4, one tab equals:
 - either 4 space characters, if the **Indent With Tabs** option is unchecked;
 - or one tab that spans 4 characters, if the **Indent With Tabs** option is checked.
- **Hard line wrap** - When enabled, Oxygen XML breaks the edited line automatically when its length exceeds the maximum set line width. This feature allows you to neatly edit a document.
- **Indent on Enter** - If enabled, Oxygen XML indents the new line introduced when pressing the Enter key.
- **Enable Smart Enter** - If you press the Enter key between a start and an end tag, Oxygen XML places the cursor in an indented position on the empty line formed between the start and end tag.
- **Detect line width on open** - Detects the line width automatically when the document is opened.

- **Format and indent the document on open** - When enabled, an XML document is formatted and indented before opening it in the editor panel. This option applies only to documents associated with the XML editor.
- **Line width - Format and Indent** - Defines the point at which the **Format and Indent** (pretty-print) function performs hard line wrapping. For example, if set to 100, after a **Format and Indent** action, the longest line will have at most 100 characters.
- **Clear undo buffer before Format and Indent** - If checked, you cannot undo anymore editing actions that preceded the **Format and Indent** operation. Only modifications performed after you have performed the operation can be undone. Check this option if you encounter out of memory problems (**OutOfMemoryError**) when performing the **Format and Indent** operation.

XML

The XML Format preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Format > XML**.

The following options are available:

- **Preserve empty lines** - The **Format and Indent** operation preserves all empty lines found in the document. Enabled by default.
- **Preserve text as it is** - The **Format and Indent** operation preserves text content as it is, without removing or adding any whitespace.
- **Preserve line breaks in attributes** - Line breaks found in attribute values are preserved. Enabled by default.



Note: When this option is enabled, **Break long lines** option is automatically disabled.

- **Break long attributes** - The **Format and Indent** operation breaks long attribute values.
- **Indent inline elements** - The *inline elements* are indented on separate lines if they are preceded by whitespaces and they follow another element start or end tag. Enabled by default. Example:

Original XML:

```
<root>
  text <parent> <child></child> </parent>
</root>
```

Indent inline elements enabled:

```
<root> text <parent>
  <child/>
</parent>
</root>
```

Indent inline elements disabled:

```
<root> text <parent> <child/> </parent> </root>
```

- **Expand empty elements** - The **Format and Indent** operation outputs empty elements with a separate closing tag, ex. `<a attr1="v1">`. When not checked, the same operation represents an empty element in a more compact form: `<a attr1="v1"/>`.
- **Sort attributes** - The **Format and Indent** operation alphabetically sorts the attributes of an element.
- **Add space before slash in empty elements** - Inserts a space character before the trailing / and > of empty elements.
- **Break line before attribute's name** - **Format and Indent** operation breaks the line before the attribute name.
- **Element spacing** Here you can control how the application handles whitespaces found in XML content.
 - **Preserve space** - List of elements for which the **Format and Indent** operation preserves the whitespaces (like blanks, tabs, and newlines). The elements can be specified by name or by XPath expressions:
 - elementName
 - //elementName

- /elementName1/elementName2/elementName3
- //xs:localName

The namespace prefixes like `xs` are treated as part of the element name without taking into account its binding to a namespace.

- **Default space** - This list contains the names of the elements for which contiguous whitespaces are merged by the **Format and Indent** operation into one blank character.
- **Mixed content** - The elements from this list are treated as mixed when applying the **Format and Indent** operation. The lines are split only when whitespaces are encountered.
- **Schema aware format and indent** - The **Format and Indent** operation takes into account the schema information regarding the *space preserve*, *mixed*, or *element only* properties of an element. Enabled by default.
- **Indent (when typing) in preserve space elements** - *Preserve space* elements (identified by the `xml:space` attribute set to `preserve` or by their presence in the **Preserve space** elements list) are normally ignored by the **Format and Indent** operation. When this option is enabled and you are editing one of these elements, its content is formatted.
- **Indent on paste - sections with number of lines less than 300** - When you paste a chunk of text that has less than 300 lines, the inserted content is indented. If you want to keep the indent style of the document you are copying content from, disable this option.

Whitespaces

This panel displays the special whitespace characters of Unicode. Any character that is checked in this panel is considered whitespace that can be normalized in an XML document. The whitespaces are normalized 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 in the group of whitespace characters that must be normalized so they are always enabled in this panel.

CSS

The CSS Format preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Format > CSS**.

The following options control the behavior of the **Format and Indent** operation:

- **Indent class content** - The *class* content is indented. Enabled by default.
- **Class body on new line** - The *class* body (including the curly brackets) is placed on a new line.
- **Add new line between classes** - An empty line is added between two classes.
- **Preserve empty lines** - The empty lines from the CSS content are preserved. Enabled by default.
- **Allow formatting embedded CSS** - The CSS content embedded in XML is formatted when the XML content is formatted. Enabled by default.

Content Completion


The *content completion feature* enables inline syntax lookup and auto completion of mark-up elements and attributes to streamline mark-up and reduce errors while editing. These settings define the operating mode of the content assistant.

The **Content Completion** preferences panel is opened from menu **Window > Preferences > oXygen > 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 matching tag** - Oxygen XML Developer plugin automatically mirrors the matching end tag when you modify or delete the name of the start tag.
- **Use content completion** - Activates the content completion assistant. Enabled by default.
- **Close the inserted element** - When you choose an entry from the content completion assistant list of proposals, Oxygen XML 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 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 inserts the optional elements specified in the DTD, XML Schema, or RELAX NG schema.
 - **Add first Choice particle** - Oxygen XML inserts the first **choice** particle specified in the DTD, XML Schema, or RELAX NG schema.
- **Case sensitive search** - The search in the content completion assistant window when you type a character is case-sensitive ('a' and 'A' are different characters).
- **Cursor position between tags** - When enabled, Oxygen XML sets the cursor automatically between start and end tag for:
 - elements with only optional attributes or no attributes at all;
 - elements with required attributes, but only when the **Insert the required attributes** option is disabled.
- **Show all entities** - Oxygen XML 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 inserts automatically the required attributes taken from the DTD or XML Schema.
- **Insert the fixed attributes** - Oxygen XML 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 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 learns the attribute values used in a document.
- **Learn on open document** - Oxygen XML automatically learns the document structure when the document is opened.
- **Learn words (Dynamic Abbreviations, available on CTRL+SPACE)** - When checked, Oxygen XML learns the typed words and makes them available in a content completion fashion by pressing **(CTRL+SPACE)**.

 **Note:** In order to be learned, the words need to be separated by space characters.

Annotations

The **Annotations** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Content Completion > Annotations**.

The following preferences can be configured for the annotations of the elements and attributes displayed by the content completion assistant:

- **Show annotations** - Oxygen XML displays the schema annotations of an element, attribute, or attribute value currently selected in the content completion assistant proposals list.
- **Show annotations as tooltip** - Oxygen XML shows the annotation of elements and attributes as a tooltip when the mouse pointer hovers over that element or attribute in the XML editor panel or in the **Elements** view (both *the Text editing mode one* and one).
- **Use DTD comments as annotation** - When enabled, Oxygen XML uses all DTD comments as annotations. If this option is disabled, Oxygen XML displays only special Oxygen XML doc : comments, or if they are missing, it displays any other comment found in the DTD.
- **Use all Relax NG annotations as documentation** - When enabled, any element outside the Relax NG namespace, that is `http://relaxng.org/ns/structure/1.0`, is considered annotation and is displayed in the annotation window next to the content completion assistant window and in the **Model** view. When disabled, only elements from the Relax NG annotations namespace, that is `http://relaxng.org/ns/compatibility/annotations/1.0` are considered annotations.

XSL

The XSL preferences panel defines what elements are suggested by the content completion assistant of the XSL editor in addition to the XSL elements. It is opened from menu **Window > Preferences > oXygen > Editor > Content Completion > XSL**.

The following options are available:

- **Automatically detect XHTML transitional or Formatting objects** - Detects if the XSL uses the XHTML or FO schemas for content completion based on the namespaces declared on the root element.

If the detection fails, choose one of the following alternatives:

- **None** - The content completion assistant offers only the XSL elements.
- **XHTML transitional** - The content completion assistant includes XHTML Transitional elements as substitutes for `xsl:element`.
- **Formating objects** - The content completion assistant includes Formating Objects elements as substitutes for `xsl:element`.
- **Custom schema** - The content completion assistant includes elements from a DTD, XML Schema, RNG schema, or NVDL schema for inserting elements from the target language of the stylesheet.

You can choose an additional schema which will be used for documenting XSL stylesheets. Either select the built-in schema or choose a custom one. Supported schemas types are: XSD, RNG, RNC, DTD, and NDVL.

XPath

The XPath preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Content Completion > XPath**.

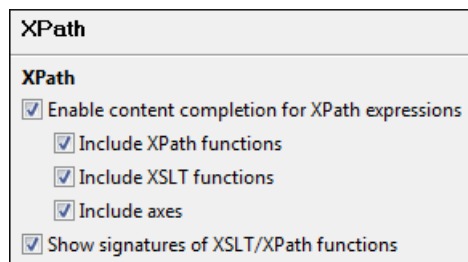


Figure 200: The Content Completion XPath Preferences Panel

The following options are available:

- **Enable content completion for XPath expressions** - Enables *the content completion assistant in XPath expressions* entered in the XSL attributes `match`, `select`, and `test` and also in the XPath toolbar. Options are available to control if the XPath functions, XSLT functions and XSLT axes are presented in the content completion proposals list when editing XPath expressions.
- **Show signatures of XSLT / XPath functions** - If checked, the editor indicates in a tooltip helper the signature of the XPath function located at the caret position. See the *XPath Tooltip Helper* section for more information.

XSD

These options 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 XSD preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Content Completion > XSD**.

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

Oxygen XML supports syntax highlight for XML, DTD, Relax NG (XML and Compact Syntax), Java, JavaScript / JSON, PHP, CSS, XQuery, C++, C, Perl, Properties, SQL, Shell and Batch documents. While Oxygen XML provides a default color configuration for highlighting the tokens, you can choose to customize it, using the Syntax Highlight options panel.

The Syntax Highlight options panel is opened from menu **Window > Preferences > oXygen > Editor > Syntax Highlight**.


Each document type has an associated set of tokens. When a document type node is expanded, the associated tokens are listed. For each token, you can customize the color and the font style. These properties are used in **Text** mode of the editor panel. The tokens for XML documents are used also in XSD, XSL, RNG documents.

 **Note:** The **Preview** area contains 4 tabs that allow you to preview XML, XSD, XSL, RNG sample files as they are rendered in Oxygen XML.

When you do not know the name of the token that you want to configure, select a token by clicking directly on that type of token in the **Preview** area.

You can edit the following color properties of the selected token:

- **Foreground color** - The **Foreground** button opens a color dialog that allows setting the color properties for the selected token with one of the following color models: Swatches, HSB, or RGB.

 **Note:** You can also open the dialog for changing the foreground color of a token by double-clicking (or pressing *Enter*) on the tree node that corresponds to that token.

- **Background color** - The **Background** button opens the same color dialog as the **Foreground** button.
- **Bold style** - This checkbox enables the bold variant of the font for the selected token. This property is not applied to a bidirectional document.
- **Italic style** - This checkbox enables the italic variant of the font for the selected token. This property is not applied to a bidirectional document.

The **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

The **Elements / Attributes by Prefix** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Syntax Highlight > Elements / Attributes by Prefix**.

One row of the table contains the association between a namespace prefix and the properties to mark start tags and end tags, or attribute names in that prefix. Note that the marking mechanism does not look at the namespace bound to that prefix. If the prefix is bound to different namespaces in different XML elements of the same file, all the tags and attribute names with that prefix are marked with the same color.

You can edit the following color properties of the selected token:

- **Foreground color** - The **Foreground** button opens a color dialog that allows setting the color properties for the selected token with one of the following color models: Swatches, HSB, or RGB.

 **Note:** You can also open the dialog for changing the foreground color of a token by double-clicking (or pressing *Enter*) on the tree node that corresponds to that token.

- **Background color** - The **Background** button opens the same color dialog as the **Foreground** button.

You can choose that only the prefix is displayed with the selected color by enabling the **Draw only the prefix with a separate color** option.

Open / Save

The **Open / Save** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Open / Save**.

The following options are available:

- **When text from BIDI Unicode range is detected** - Allows you to choose the application behavior when you try to open a file that contains BIDI Unicode characters. You can choose between **Enable bidirectional editing mode**, **Disable bidirectional editing mode** and **Prompt for each document**.
- **Safe save (only for local files)** - Option that provides an increased degree of protection in the unlikely event of a failure of the **Save** action. This mechanism creates a temporary file that holds the edited content until it is safely saved in the original file. If the **Save** action fails, the temporary file is kept in the system temporary folder, **OxygenXMLTemp** subfolder.
- **Save all files before transformation or validation** - Saves all open files before validating or transforming an XML document. This way the dependencies are resolved, for example when modifying both the XML document and its XML Schema.
- **Clear undo buffer on save** - If enabled, Oxygen XML erases its undo stack when you save a document. Only modifications made after you have saved the document can be undone. Check this option if you encounter frequent *out of memory* problems (**OutOfMemoryError**) when editing very large documents.

Templates

This panel groups the preferences that are related with code templates and document templates:

- [Code Templates](#)
- [Document Templates](#)

Code Templates

Code templates are small document fragments that can be inserted quickly at the editing position and can be reused in other editing sessions. Oxygen XML comes with a large set of ready-to use templates for XSL, XQuery, and XML Schema. You can even share your code templates with your colleagues using the template export and import functions.

To obtain the template list, use:

- The shortcut key that activates content completion assistant on request: **(Ctrl+Space)** on Windows and Linux, **(Cmd+Space)** on Mac OS X. It displays the code templates names in together with proposed element names.
- The shortcut key that activates code templates assistant on request: **(Ctrl+Shift+Space)** on Windows and Linux, **(Cmd+Shift+Space)** on Mac OS X. It displays only the code templates in the proposals list.

The **Code Templates** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Templates > Code Templates**. It contains a list with all available code templates (both built-in and custom created ones) and a code preview area. You can disable any code template by unchecking its corresponding option box.

- **New** - Defines a new code template. You can choose to set the newly defined code template for a specific type of editor or for all editor types.
- **Edit** - Edits the selected code template.
- **Duplicate** - Creates a duplicate of the currently selected code template.
- **Delete** - Deletes the currently selected code template. This action is disabled for the built-in code templates.
- **Import** - Imports a file with code templates that was created by the **Export** action.
- **Export** - Exports a file with code templates.

Document Templates

The list of document templates that are displayed in *the New dialog* can be extended with custom templates that are specified in the **Document Templates** preferences panel. Add the template files in a folder that is specified in this panel or in the `templates` folder of the Oxygen XML Developer plugin install directory.

The **Document Templates** preferences panel is opened from menu **Window > Preferences > oXygen > 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 looks into these location folders by using the **Up** and **Down** buttons on a selected table row.

Spell Check

The **Spell Check** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Spell Check**.

The following options are available:

- **Automatic Spell Check** - When enabled, the spell checker highlights the errors as you modify the document.
- **Select editors** - Allows you to select the file types for which the automatic spell check takes effect. File types in which automatic spell check is generally not useful, like CSS and DTD, are excluded by default.
- **Spell checking engine** - The application ships with two spell check engines, *Hunspell* and *Java spell checker*. Each engine has a specific format of spelling dictionaries. The languages of the built-in dictionaries of the selected engine are listed in the **Default language** options list.
- **Default language** - The default language list allows you to choose the language used by the spell check engine.



Note: You can [add more spelling dictionaries](#) to the spell check engines.

- **Delete learned words** - Press this button to open the list of learned words. Here you can select the items you want to remove.
- **Use "lang" and "xml:lang" attributes** - If enabled, the contents of any element with one of the `lang` or `xml:lang` attributes is checked using a dictionary suitable for the language specified in the attribute value, if this dictionary is available. When these attributes are missing, the language used is controlled by the two radio buttons: **Use the default language** or **Do not check**.
- **XML spell checking in** - These options allow you to specify if the spell checker will be enabled inside XML comments, attribute values, text, and CDATA sections.
- **Case sensitive** - When enabled, spell checking reports capitalization errors, for example a word that starts with lowercase after *etc.* or *i.e.*.
- **Ignore mixed case words** - When enabled, operations do not check words containing mixed case characters (e.g. *SpellChecker*).
- **Ignore words with digits** - When enabled, the spell checker does not check words containing digits (e.g. *b2b*).
- **Ignore Duplicates** - When enabled, the spell checker does not signal two successive identical words as an error.
- **Ignore URL** - When enabled, ignores words looking like URL or file names (e.g. *www.oxygenxml.com* or *c:\boot.ini*).
- **Check punctuation** - When enabled, punctuation checking is enabled: misplaced white space and wrong sequences, like a dot following a comma, are highlighted as errors.
- **Allow compounds words** - When enabled, all words formed by concatenating two legal words with a hyphen (hyphenated compounds) are accepted. If the language allows it, two words concatenated without hyphen (closed compounds) are also accepted.
- **Allow general prefixes** - When enabled, a word formed by concatenating a registered prefix and a legal word is accepted. For example if *mini-* is a registered prefix, the spell check engine accepts the word *mini-computer*.
- **Allow file extensions** - When enabled, accepts any word ending with registered file extensions (e.g. *myfile.txt*, *index.html*, etc.).
- **Ignore acronyms** - When enabled, the acronyms are not reported as errors.
- **Ignore elements** - A list of XPath expressions for the elements that will be ignored by the spell check engine. The following restricted set of XPath expressions are supported:
 - '/' and '/' separators;
 - '*' wildcard.

An example of allowed XPath expression: `/a*/b`.

Document Checking

The **Document Checking** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Document Checking**. It contains preferences for configuring how a document is checked for both well-formedness errors and validation errors.

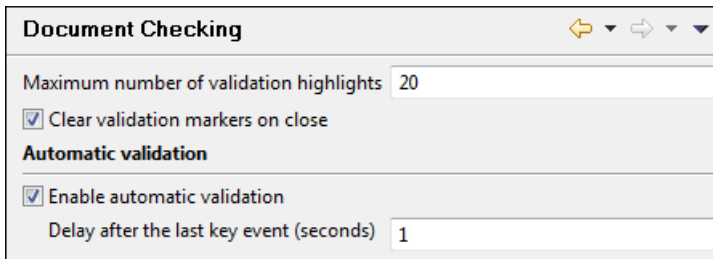


Figure 201: Document Checking Preferences Panel

The error checking preferences are the following:

- **Maximum number of validation highlights** - If validation generates more errors than the number from this option only the first errors up to this number are highlighted in editor panel and on stripe that is displayed at right side of editor panel. This option is applied both for *automatic validation* and *manual validation*.
- **Clear validation markers on close** - If this option is selected all the error markers added in the **Problems** view for that document are removed when a document edited with the Oxygen XML Developer 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

The **Mark Occurrences** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Mark Occurrences**:

The following preferences can be set:

- **XSLT files** - activates the *Highlight Component Occurrences* in XSLT files;
- **XML Schema files** - activates the *Highlight Component Occurrences* in XSD files;
- **Declaration highlight color** - color used to highlight the component declaration;
- **Reference highlight color** - color used to highlight component references.

Custom Validation Engines

The **Custom Validation Engines** preferences panel is opened from menu **Window > Preferences > oXygen > Editor > Custom Validations**.

If you want to add a new custom validation tool or edit the properties of an existing one you can use the **Custom Validator** dialog displayed by pressing the **New** button or the **Edit** button.

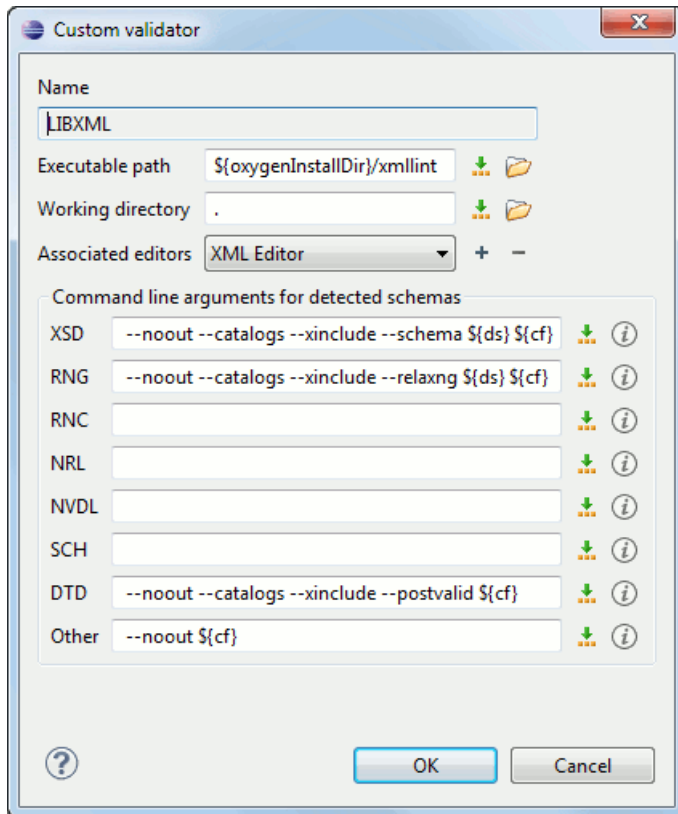


Figure 202: Edit a Custom Validator

The configurable parameters of a custom validator are the following:

- **Name** - Name of the custom validation tool displayed in the **Custom Validation Engines** toolbar.
- **Executable path** - Path to the executable file of the custom validation tool. You can insert here *editor variables* like *\${home}*, *\${pd}*, *\${oxygenInstallDir}*, etc.
- **Working directory** - The working directory of the custom validation tool.
- **Associated editors** - The editors which can perform validation with the external tool: the XML editor, the XSL editor, the XSD editor, etc.
- **Command line arguments for detected schemas** - Command line arguments used in the commands that validate the current edited file against different types of schema: W3C XML Schema, Relax NG full syntax, Relax NG compact syntax, NVDL, Schematron, DTD, etc.. The arguments can include any custom switch (like *-rng*) and the following editor variables:
 - **\${cf}** - Current file as file path, that is the absolute file path of the current edited document.
 - **\${cfu}** - The path of the current file as a URL.
 - **\${ds}** - The path of the detected schema as a local file path.
 - **\${dsu}** - The path of the detected schema as a URL.

CSS Validator

The **CSS Validator** preferences panel is opened from menu **Window > Preferences > oXygen > CSS Validator**.

The following options can be configured for Oxygen XML Developer plugin 's built-in CSS validator:

- **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 profile **CSS 3 with Oxygen extensions** includes all the CSS 3 standard properties and the CSS extensions specific for Oxygen that can be used in Author mode. That means all Oxygen specific extensions are accepted in a CSS stylesheet by *the built-in CSS validator* when this profile is selected.

- **Media type** - Selects one of the available mediums: **all, aural, braille, embossed, handheld, print, projection, screen, tty, tv, presentation, oxygen.**
- **Warning level** - Sets the minimum severity level for reported validation warnings. Can be one of: **All, Normal, Most Important, No Warnings.**
- **Ignore properties** - Here you can type comma separated patterns that match the names of CSS properties that will be ignored at validation. As wildcards you can use:
 - * to match any string;
 - ? to match any character.
- **Recognize browser CSS extensions (applies also to content completion)** - If checked, Oxygen XML 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;
 - -o- for Opera;
 - -webkit- for Safari/Webkit.

XML

This section describes the panels that contain the user preferences related with XML.

XML Catalog

The **XML Catalog** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XML Catalog.**

The **Prefer** option is used to specify if Oxygen XML Developer plugin will try to resolve first the PUBLIC or SYSTEM reference from the DOCTYPE declaration of the XML document. If PUBLIC is preferred and a PUBLIC reference is not mapped in any of the XML catalogs then a SYSTEM reference is looked up.

When using catalogs it is sometimes useful to see what catalog files are parsed, if they are valid or not, and what identifiers are resolved by the catalogs. The **Verbosity** option selects the detail level of such logging messages of the XML catalog resolver that will be displayed in the **Catalogs** view at the bottom of the window:

- **None** - No message is displayed by the catalog resolver when it tries to resolve a URI reference, a SYSTEM one or a PUBLIC one with the XML catalogs specified in this panel.
- **Unresolved entities** - Only the logging messages that track the failed attempts to resolve references are displayed.
- **All messages** - The messages of both failed attempts and successful ones are displayed.

If the **Process namespaces through URI mappings for XML Schema** option is not selected only the schema location of an XML Schema that is declared in an XML document is searched in XML catalogs. If the option is selected the schema location of an XML Schema is searched and if it is not resolved the namespace of the schema is also searched.

If the **Use default catalog** option is checked the first XML catalog which Oxygen 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 you must reopen the current edited files which use the modified catalog or run [the action Reset Cache and Validate](#) so that the XML catalog changes take full effect.

XML Parser

The **XML Parser** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XML Parser.**

The configurable options of the built-in XML parser are the following:

- **<http://apache.org/xml/features/validation/schema-full-checking>** - Sets the *schema-full-checking* feature to true, that is a validation of the parsed XML document is performed against a schema (W3C XML Schema or DTD) while the document is parsed.
- **<http://apache.org/xml/features/honour-all-schema-location>** - Sets the *honour-all-schema-location* feature to true. This means all the files that declare W3C XML Schema components from the same namespace are used to compose the validation model. If this option is not selected only the first W3C XML Schema file that is encountered in the XML Schema import tree is taken into account.
- **Ignore the DTD for validation if a schema is specified** - Forces validation against a referred schema (W3C XML Schema, Relax NG schema, Schematron schema) even if the document includes also a DTD DOCTYPE declaration. This option is useful when the DTD declaration is used only to declare DTD entities and the schema reference is used for validation against a W3C XML Schema, a Relax NG schema or a Schematron schema.
- **Enable XInclude processing** - Enables XInclude processing. If checked, the XInclude support in Oxygen 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 infocset information would be incorrect.

Unfortunately, these attributes make XInclude processing not transparent to Schema validation. One solution to this is to modify your schema to allow `xml:base` attributes to appear on elements that might be included from different base URIs.

If the addition of `xml:base` and / or `xml:lang` is undesired by your application, you can disable base URI fix-up.

- **Language fix-up** - The processor will preserve language information on a top-level included element by adding an `xml:lang` attribute if its include parent has a different [language] property. If the addition of `xml:lang` is undesired by your application, you can disable the language fix-up.
- **Check ID/IDREF** - Checks the ID/IDREF matches when the Relax NG document is validated.
- **Check feasibly valid** - Checks the Relax NG to be feasibly valid when this document is validated.
- **Schematron XPath Version** - Selects the version of XPath for the expressions that are allowed in Schematron assertion tests: 1.0 or 2.0. This option is applied both in standalone Schematron schemas and in embedded Schematron rules, both in Schematron 1.5 and in ISO Schematron.
- **Optimize (visit-no-attributes)** - If your ISO Schematron assertion tests do not contain the attributes axis you should check this option for faster ISO Schematron validation.
- **Allow foreign elements (allow-foreign)** - Enables support for `allow-foreign` on ISO Schematron. This option is used to pass non-Schematron elements to the generated stylesheet.
- **Use Saxon EE (schema aware) for xslt2 query binding** - If checked, Saxon EE will be used for `xslt2` query binding. If not checked, Saxon PE will be used instead.

Saxon EE Validation

The **Saxon EE Validation** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XML Parser > Saxon EE Validation**.

The following options are available:

- **XML Schema version** - allows you to select the version of W3C XML Schema for validation against XML Schema performed by the Saxon EE engine: XML Schema 1.0 or XML Schema 2.0.
- **XML Schema validation - Use Saxon EE as default XML Schema validation engine** - you can set Oxygen XML Developer plugin to use Saxon EE as default XML Schema validator. If enabled it is used to validate XML Schema and XML documents against an XML Schema. By default this option is turned off.

XML Instances Generator

The **XML Instances Generator** preferences panel is opened from menu **Window > Preferences > oXygen > 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** - The number of repetitions for an element that has a big value of the `maxOccurs` attribute.
- **Maximum recursivity level** - For recursive type definitions this parameter specifies the number of levels of recursive elements inserted in the parent element with the same name.
- **Choice strategy** - For choice element models specifies what choice will be generated in the XML instance:
 - **First** - the first choice is selected from the choice definition and an instance of that choice is generated in the XML instance document.
 - **Random** - a random choice is selected from the choice definition and an instance of that will be generated.
- **Generate the other options as comments** - If checked, the other options of the choice element model (the options which are not selected) will be generated inside an XML comment in the XML instance.
- **Use incremental attribute / element names as default** - If checked, the value of an element or attribute starts with the name of that element or attribute. For example, for an `a` element the generated values 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

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.

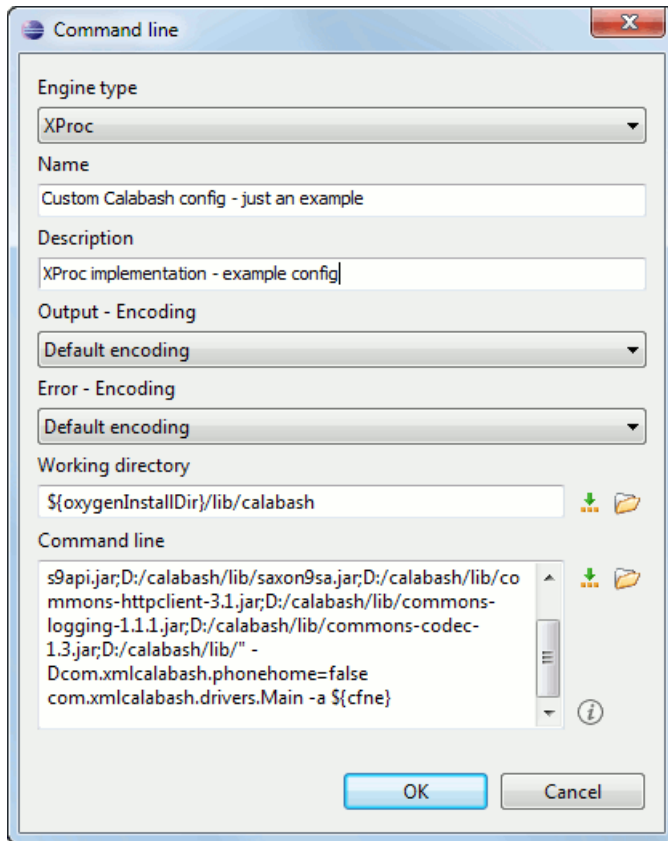


Figure 203: Creating an XProc external engine

Other parameters that can be set for an XProc external engine are the following: , and the error stream of the engine, the working directory of the command that will start the engine. The encodings will be used for reading and displaying the output of the engine. The working directory and

- a textual description that will appear as tooltip where the XProc engine will be used
- the encoding for the output stream of the XProc engine, used for reading and displaying the output messages
- the encoding for the error stream of the XProc engine, used for reading and displaying the messages from the error stream
- the working directory for resolving relative paths
- the command line that will run the XProc engine as an external process; the command line can use *built-in editor variables* and *custom editor variables* for parameterizing a file path.

XSLT/FO/XQuery

The **XSLT/FO/XQuery** preferences panel is opened from menu **Window > Preferences > oXygen > 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

The **XSLT** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XSLT**.

If you want to use an XSLT transformer implemented in Java different than the ones that ship with Oxygen XML Developer plugin namely Apache Xalan and Saxon all you have to do is to specify the name of the transformer's factory class which Oxygen XML Developer plugin will set as the value of the Java property `javax.xml.transform.TransformerFactory`. For instance, to perform an XSLT transformation with Saxon 9.3.0.5 you have to place the Saxon 9.3.0.5 jar file in the Oxygen XML Developer plugin libraries folder (the `lib` subfolder of the Oxygen XML Developer plugin installation folder), set `net.sf.saxon.TransformerFactoryImpl` as the property value and select JAXP as the XSLT processor in the transformation scenario associated to the transformed XML document.

The XSLT preferences are the following:

- **Value** - Allows the user to enter the name of the transformer factory Java class.
- **XSLT 1.0 Validate with** - Allows the user to set the XSLT engine used for validation of XSLT 1.0 documents.
- **XSLT 2.0 Validate with** - Allows the user to set the XSLT Engine used for validation of XSLT 2.0 documents.

Saxon6

The **Saxon 6** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon 6**.

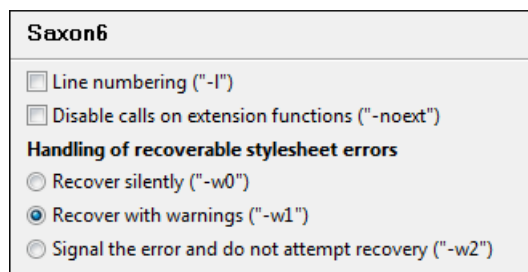


Figure 204: The Saxon 6 XSLT Preferences Panel

The built-in Saxon 6 XSLT processor can be configured with the following options:

- **Line numbering** - If checked, line numbers are maintained and reported in error messages for the XML source document.
- **Disable calls on extension functions** - If checked, external functions called is disallowed. Checking this is recommended in an environment where untrusted stylesheets may be executed. Also disables user-defined extension elements, together with the writing of multiple output files, all of which carry similar security risks.
- **Handling of recoverable stylesheet errors** - Allows the user to choose how dynamic errors will be handled. Either one of the following options can be selected:
 - **recover silently** - continue processing without reporting the error,
 - **recover with warnings** - issue a warning but continue processing,
 - **signal the error and do not attempt recovery** - issue an error and stop processing.

Saxon HE/PE/EE

The **Saxon HE/PE/EE** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon HE/PE/EE**.

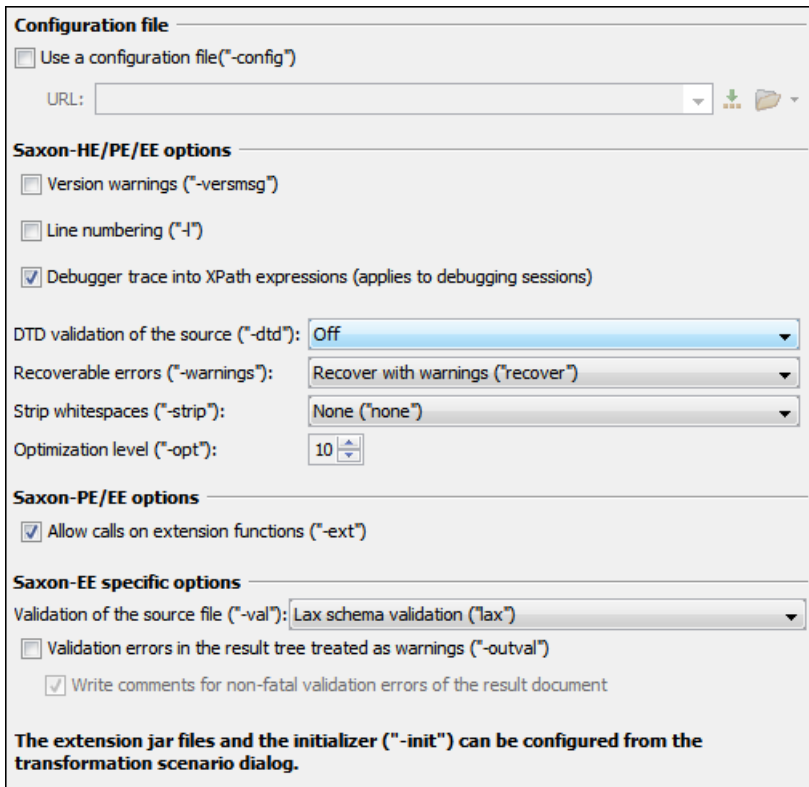


Figure 205: The Saxon HE/PE/EE XSLT preferences panel

The XSLT options which can be configured for the Saxon 9.3.0.5 transformer (all three editions: Home Edition, Professional Edition, Enterprise Edition) are the following:

- **Use a configuration file ("-config")** - Sets a Saxon 9 configuration file that will be used for XSLT transformation and validation.
- **Version warnings ("-versmsg")** - Warns you when the transformation is applied to an XSLT 1.0 stylesheet.
- **Line numbering ("-l")** - Error line number is included in the output messages.
- **Debugger trace into XPath expressions (applies to debugging sessions)** - Instructs the *XSLT Debugger* to step into XPath expressions.
- **DTD validation of the source ("-dtd")** - The following options are available:
 - **On**, requests *DTD-based* validation of the source file and of any files read using the `document()` function;
 - **Off** (default setting) suppresses DTD validation.
 - **Recover**, performs DTD validation but treats the errors as non-fatal.

Note that any external DTD is likely to be read even if not used for validation, because DTDs can contain definitions of entities.

- **Recoverable errors ("-warnings")** - Policy for handling recoverable errors in the stylesheet: Allows you to choose how dynamic errors are handled. One of the following options can be selected:
 - **Recover silently ("silent")** ;
 - **Recover with warnings ("recover")** - default setting;
 - **Signal the error and do not attempt recovery ("fatal")**.
- **Strip whitespaces ("-strip")** - Strip whitespaces feature can be one of the following three options:
 - **All ("all")** - Strips all whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes in the source document.
 - **Ignorable ("ignorable")** - Strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes

in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.

- **None ("none")** - default setting. No whitespaces are stripped before further processing. However, whitespace will still be stripped if this is specified in the stylesheet using `xsl:strip-space`.
- **Optimization level ("-opt")** - Set optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization); currently all values other than 0 result in full optimization but this is likely to change in the future. The default is full optimization; this feature allows optimization to be suppressed in cases where reducing compile time is important, or where optimization gets in the way of debugging, or causes extension functions with side-effects to behave unpredictably. (Note however, that even with no optimization, the lazy evaluation may still cause the evaluation order to be not as expected.)
- **Allow calls on extension functions ("-ext")** - If checked, the stylesheet is allowed to call external Java functions. This does not affect calls on integrated extension functions, including Saxon and EXSLT extension functions. This option is useful when loading an untrusted stylesheet, perhaps from a remote site using an `http://` URL; it ensures that the stylesheet cannot call arbitrary Java methods and thereby gain privileged access to resources on your machine.
- **Validation of the source file ("-val")** - Requests schema-based validation of the source file and of any files read using the `document()` or similar functions. Validation is available only with Saxon-EE, and this flag automatically switches on the `-sa` option. Available options:
 - **Schema validation ("strict")** - This mode requires an XML Schema and specifies that the source documents should be parsed with schema-validation enabled.
 - **Lax schema validation ("lax")** - This mode specifies if the source documents should be parsed with schema-validation enabled if an XML Schema is provided.
 - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.
- **Validation errors in the results tree treated as warnings ("-outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.
- **Write comments for non-fatal validation errors of the result document** - The validation messages are written (where possible) as a comment in the result document itself.

Saxon HE/PE/EE Advanced

The **Saxon HE/PE/EE Advanced** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XSLT > Saxon > Saxon HE/PE/EE > Advanced**.

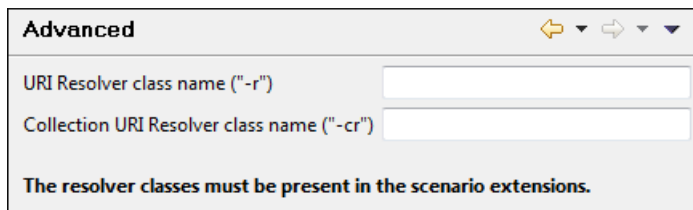


Figure 206: The Saxon HE/PE/EE XSLT Advanced Preferences Panel

There are some advanced XSLT options which can be configured for the Saxon 9.3.0.5 transformer (all three editions: Home Edition, Professional Edition, Enterprise Edition):

- **URI Resolver class name ("-r")** - Allows the user to specify a custom implementation for the URI resolver used by the XSLT Saxon 9.3.0.5 transformer (the `-r` option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from [the dialog for configuring the XSLT extension](#) for the particular transformation scenario.
- **Collection URI Resolver class name ("-cr")** - Allows the user to specify a custom implementation for the Collection URI resolver used by the XSLT Saxon 9.3.0.5 transformer (the `-cr` option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from [the dialog for configuring the XSLT extension](#) for the particular transformation scenario.

XSLTProc

The **XSLTProc** preferences panel is opened from menu **Window > Preferences > oXygen > 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

The MSXML preferences panel is opened from menu **Window > Preferences > oXygen > 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

The MSXML.NET preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XSLT > MSXML.NET**.

The options of the MSXML.NET processor are the same as *the ones available in the command line for the MSXML.NET processor*:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when MSXML.NET is used as transformer in the *XSLT transformation scenario*.
- **Validate documents during parse phase** - If checked and either the source or stylesheet document has a DTD or schema against which its content can be checked, validation is performed.
- **Do not resolve external definitions during parse phase** - By default MSXML.NET resolves external definitions such as DTD external subsets or external entity references when parsing source XML document and stylesheet document. Using this option you can disable this behaviour. Note, that it may affect also the validation process for the XML document.
- **Strip non-significant whitespaces** - If checked, strips non-significant white space from the input XML document during the load phase. Enabling this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.
- **Show time information** - If checked, the relative speed of various transformation steps can be measured:
 - the time to load, parse, and build the input document
 - the time to load, parse, and build the stylesheet document
 - the time to compile the stylesheet in preparation for the transformation
 - the time to execute the stylesheet
- **Forces ASCII output encoding** - There is a known problem with .NET 1.X XSLT processor (`System.Xml.Xsl.XslTransform` class): it doesn't support escaping of characters as XML character references when they cannot be represented in the output encoding. That means that when you output a character that cannot be represented in output encoding, it will be outputted as '?'. Usually this happens when output encoding is set to ASCII. With this option checked the output is forced to be ASCII encoded and all non-ASCII characters get escaped as XML character references (`&#nnnn;` form).
- **Allow multiple output documents** - This option allows to create multiple result documents using *the `exsl:document` extension element*.
- **Use named URI resolver class** - This option allows to specify a custom URI resolver class to resolve URI references in `xsl:import` and `xsl:include` instructions (during XSLT stylesheet loading phase) and in `document()` function (during XSL transformation phase).
- **Assembly file name for URI resolver class** - The previous option specifies partially or fully qualified URI resolver class name, e.g. `Acme.Resolvers.CacheResolver`. Such name requires additional assembly specification using this option or the next option, but fully qualified class name (which always includes an assembly specifier) is all-sufficient. See MSDN for more info about *fully qualified class names*. This option specifies a file name of the assembly, where the specified resolver class can be found.
- **Assembly GAC name for URI resolver class** - This option specifies partially or fully qualified name of the assembly in the *global assembly cache* (GAC), where the specified resolver class can be found. See MSDN for more info about *partial assembly names*. Also see the previous option.
- **List of extension object class names** - This option allows to specify *extension object* classes, whose public methods then can be used as extension functions in an XSLT stylesheet. It is a comma-separated list of namespace-qualified extension object class names. Each class name must be bound to a namespace URI using prefixes as *when providing XSLT parameters*.
- **Use specified EXSLT assembly** - MSXML.NET supports a rich library of the *EXSLT* and *EXSLT.NET* extension functions embedded or in a plugged in EXSLT.NET library. EXSLT support is enabled by default and cannot be

disabled in this version. If you want to use an external EXSLT.NET implementation instead of a built-in one use this option.

- **Credential loading source xml** - This option allows to specify user credentials to be used when loading XML source documents. The credentials should be provided in the *username:password@domain* format (all parts are optional).
- **Credential loading stylesheet** - This option allows to specify user credentials to be used when loading XSLT stylesheet documents. The credentials should be provided in the *username:password@domain* format (all parts are optional).

XQuery

The **XQuery** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XQuery**.

The generic XQuery preferences are the following:

- **XQuery validate with** - Allows you to select the processor to validate the XQuery. In case you are validating an XQuery file that has an associated validation scenario, Oxygen 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 will be used. If the processor does not support validation or if no scenario is associated, then the value from this combo box will be used as validation processor
- **Size limit of Sequence view (MB)** - When the result of an XQuery transformation is *set in the transformation scenario as sequence* the size of one chunk of the result that is fetched from the database in lazy mode in one step is set in this option. If this limit is exceeded you can extract more data from the database by a click on the **More result available** node from the **Sequence** view.
- **Format transformer output** - When checked the transformer's output is formatted and indented (pretty printed). The option is ignored if in the transformation scenario you choose **Sequence** (lazy extract data from a database).
- **Create structure indicating the type nodes** - If checked, Oxygen XML Developer plugin takes the results of a query and creates an XML document containing copies of all items in the sequence, suitably wrapped. The option is ignored if in the transformation scenario you choose **Sequence** (lazy extract data from a database).

Saxon HE/PE/EE

The **Saxon HE/PE/EE** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XQuery > Saxon HE/PE/EE**.

Figure 207: The Saxon XQuery Preferences panel

The XQuery preferences for the Saxon 9.3.0.5 are the following:

- **Use a configuration file ("-config")** - Sets a Saxon 9 configuration file that will be used for XQuery transformation and validation.
- **Handling of recoverable stylesheet errors** - Allows the user to choose how dynamic errors will be handled. Either one of the following options can be selected:
 - **recover silently** - continue processing without reporting the error,
 - **recover with warnings** - issue a warning but continue processing,
 - **signal the error and do not attempt recovery** - issue an error and stop processing.
- **Strip whitespaces** - Can have one of the following three values:
 - **All** - Strips all whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document.
 - **Ignore** - Strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
 - **None** - Strips no whitespace before further processing.
- **Optimization level** - This option allows optimization to be suppressed in cases where reducing compile time is important, or where optimization gets in the way of debugging, or causes extension functions with side-effects to behave unpredictably.

The Saxon 9.3.0.5 Professional Edition options are the following:

- **Disable calls on extension functions** - If unchecked, external functions calls is allowed. Checking this is recommended in an environment where untrusted stylesheets may be executed. Also disables user-defined extension elements, together with the writing of multiple output files, all of which carry similar security risks.

The Saxon 9.3.0.5 Enterprise Edition specific options are the following:

- **Validation of the source** - This determines whether XML source documents should be parsed with schema-validation enabled.

- **Validation errors in the results tree treated as warnings** - Available only for Saxon EE. If checked, all validation errors are treated as warnings, otherwise they are treated as fatal.
- **Enable XQuery 1.1 support** - If checked, Saxon EE runs the XQuery transformation with the XQuery 1.1 support.
- **Backup files updated by XQuery ("-backup:(on/off)")** - If checked, backup versions for any XML files updated with XQuery Update will be generated.

Saxon HE/PE/EE Advanced

The **Saxon HE/PE/EE Advanced** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XQuery > Saxon HE/PE/EE > Advanced**.

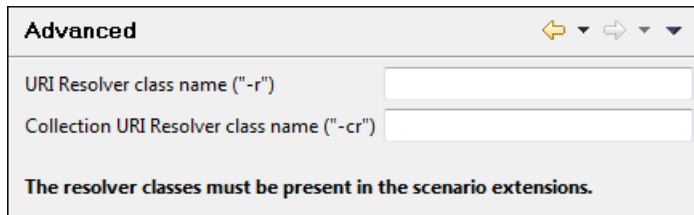


Figure 208: The Saxon HE/PE/EE XQuery Advanced Preferences Panel

The advanced XQuery options which can be configured for the Saxon 9.3.0.5 XQuery transformer (all editions: Home Edition, Professional Edition, Enterprise Edition) are the following:

- **URI Resolver class name** - Allows the user to specify a custom implementation for the URI resolver used by the XQuery Saxon 9.3.0.5 transformer (the -r option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from [the dialog for configuring the XQuery extension](#) for the particular transformation scenario.
- **Collection URI Resolver class name** - Allows the user to specify a custom implementation for the Collection URI resolver used by the XQuery Saxon 9.3.0.5 transformer (the -cr option when run from the command line). The class name must be fully specified and the corresponding jar or class extension must be configured from [the dialog for configuring the XQuery extension](#) for the particular transformation scenario.

Debugger

This section explains the settings available for the Debugger perspective. The settings are available from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > Debugger**.

The debugger preferences are the following:

- **Show xsl:result-document output** - If checked, the debugger presents the output of `xsl:result-document` instructions into the debugger output view.
- **Infinite loop detection** - Set this option to receive notifications when an infinite loop occurs during transformation.
- **Maximum depth in templates stack** - Sets how many `xsl:template` instructions can appear on the current stack. This setting is used by the infinite loop detection.
- **Debugger layout** - A horizontal layout means that the stack of XML editors takes the left half of the editing area and the stack of XSL editors takes the right one. A vertical layout means that the stack of XML editors takes the upper half of the editing area and the stack of XSL editors takes the lower one.

Annotations

The **Annotations** preferences panel is opened from menu **Window > Preferences > 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

This section explains the settings available for the XSLT Profiler. To access and modify them please go to menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > Profiler** (see [Debugger](#) on page 354).

The following profiles settings are available:

- **Show time** - Shows the total time that was spent in the node.
- **Show inherent time** - Shows the inherent time that was spent in the node. The inherent time is defined as the total time of a node minus the time of its child nodes.
- **Show invocation count** - Shows how many times the node was called in this particular call sequence.
- **Time scale** - The time scale options determine the unit of time measurement, which may be milliseconds or microseconds.
- **Hotspot** threshold - The threshold below which hot spots are ignored (milliseconds).
- **Ignore invocation less than** - The threshold below which invocations are ignored (microseconds).
- **Percentage calculation** - The percentage base determines against what time span percentages are calculated:
 - **Absolute** - Percentage values show the contribution to the total time.
 - **Relative** - Percentage values show the contribution to the calling node.

FO Processors

Besides the built-in formatting objects processor (Apache FOP) other external processors can be configured and set in transformation scenarios for processing XSL-FO documents.

Oxygen XML Developer plugin has implemented an easy way to add two of the most used commercial FO processors: RenderX XEP and Antenna House XSL Formatter. You can easily add RenderX XEP as external FO processor if the user has the XEP installed. Also, if you have the Antenna House XSL Formatter v4 or v5, Oxygen 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 just as you would for XEP.

The **FO Processors** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > FO Processors**.

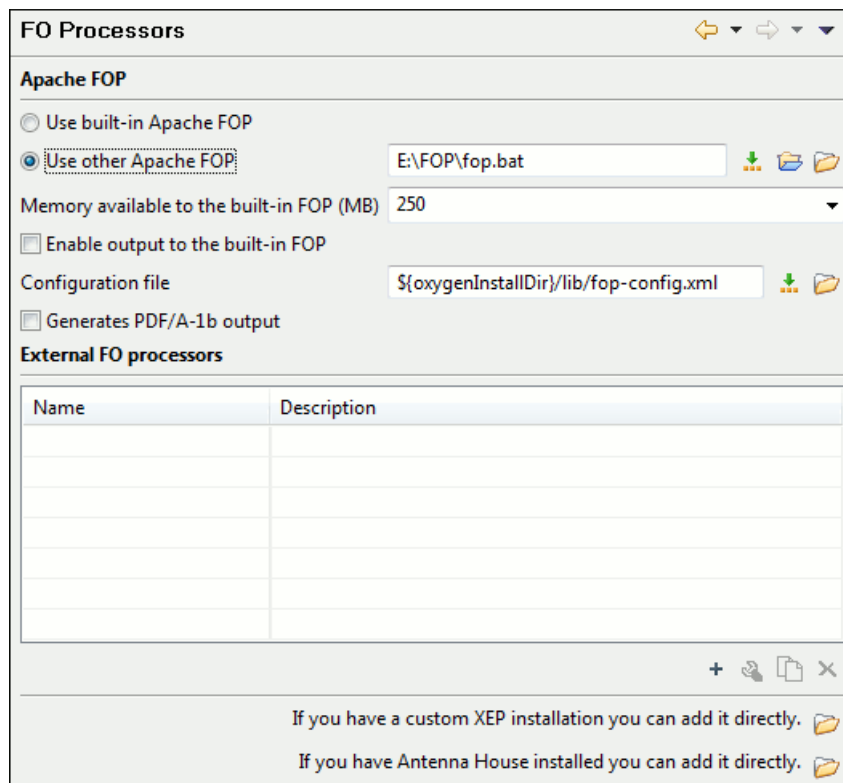


Figure 209: The FO Processors Preferences Panel

Apache FOP

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.*

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:

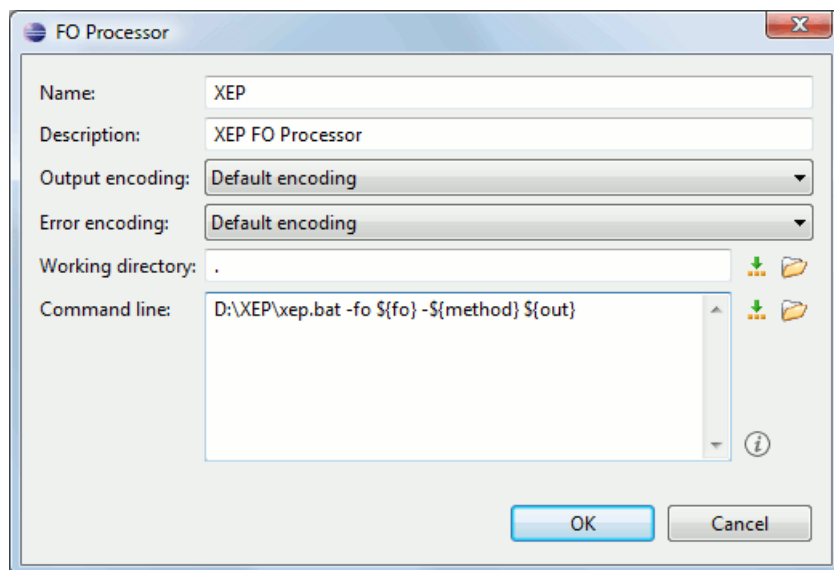


Figure 210: The External FO Processor Configuration Dialog

- **Name** - The name displayed in the list of available FOP processors on the FOP tab of the transformation scenario dialog.
- **Description** - A textual description of the FO processor displayed in the FO processors table and in tooltips of UI components where the processor is selected.
- **Output Encoding** - The encoding of the FO processor output stream displayed in a results panel at the bottom of the Oxygen XML 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.

- **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.

XPath

The XPath preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > XPath**.

The XPath options are the following:

- **Unescape XPath expression** - When checked, the entities are unescaped in the XPath expressions entered in *the XPath toolbar*. For example the expression

```
//varlistentry[starts-with(@os, '&#x73;')]
```

is equivalent with:

```
//varlistentry[starts-with(@os, 's')]
```

- **No namespace** - If checked, Oxygen XML Developer plugin will consider unprefixed element names in XPath 2.0 expressions evaluated in *the XPath console* as belonging to no namespace.
- **Use the default namespace from the root element** - If checked, Oxygen XML Developer plugin will consider unprefixed element names in XPath expressions evaluated in *the XPath console* as belonging to the default namespace declared on the root element of the queried XML document.
- **Use the namespace of the root** - If checked, Oxygen XML Developer plugin will consider unprefixed element names in XPath expressions evaluated in *the XPath console* as belonging to the same namespace as the root element of the document.
- **This namespace** - The user has the possibility to enter here the namespace of the unprefixed elements used in *the XPath console*.
- **Default prefix-namespace mappings** - Associates prefixes to namespaces. These mappings are useful when applying an XPath in the XPath console and you don't want to define these mappings in each document separately.

Custom Engines

You can configure and run XSLT and XQuery transformations with processors other than *the ones which come with the Oxygen XML Developer plugin distribution*. Such an external engine can be used in the Editor perspective and is available in the list of engines in *the dialog for editing a transformation scenario*.

The **Custom Engines** preferences panel is opened from menu **Window > Preferences > oXygen > XML > XSLT/FO/XQuery > Custom Engines**.

The following parameters can be configured for a custom engine:

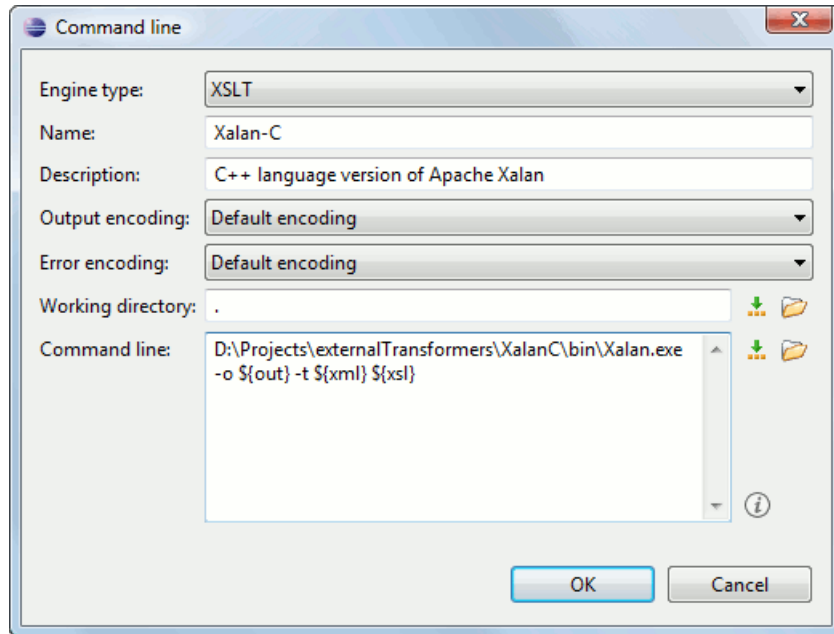


Figure 211: Parameters of a Custom Engine

- **Engine type** - Combo box allowing you to choose the transformer type. There are two options: XSLT engines and XQuery engines.
- **Name** - The name of the transformer displayed in the dialog for editing transformation scenarios
- **Description** - A textual description of the transformer.
- **Output Encoding** - The encoding of the transformer output stream.
- **Error Encoding** - The encoding of the transformer error stream.
- **Working directory** - The start directory of the transformer executable program. The following editor variables are available for making the path to the working directory independent of the location of the input files:
 - **\${homeDir}** - The user home directory in the operating system.
 - **\${cfd}** - The path to the directory of the current file.
 - **\${pd}** - The path to the directory of the current project.
 - **\${oxygenInstallDir}** - The Oxygen XML 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.

Import

The **Import** preferences panel is opened from menu **Window > Preferences > oXygen > XML > Import**. Here you can 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

Table 9: Pattern letters

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am / pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am / pm (0-11)	Number	0
h	Hour in am / pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978

Letter	Date or Time Component	Presentation	Examples
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

Pattern letters are usually repeated, as their number determines the exact presentation:

- *Text* - If the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available.
- *Number* - The number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount.
- *Year* - If the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a number.
- *Month* - If the number of pattern letters is 3 or more, the month is interpreted as text; otherwise, it is interpreted as a number.
- *General time zone* - Time zones are interpreted as text if they have names. For time zones representing a GMT offset value, the following syntax is used:
 - *GMTOffsetTimeZone* - GMT Sign Hours : Minutes
 - *Sign* - one of + or -
 - *Hours* - one or two digits
 - *Minutes* - two digits
 - *Digit* - one of 0 1 2 3 4 5 6 7 8 9

Hours must be between 0 and 23, and Minutes must be between 00 and 59. The format is locale independent and digits must be taken from the Basic Latin block of the Unicode standard.

- *RFC 822 time zone*: The RFC 822 4-digit time zone format is used:
 - *RFC822TimeZone* - Sign *TwoDigitHours* Minutes
 - *TwoDigitHours* - a number of two digits

TwoDigitHours must be between 00 and 23.

Data Sources

The **Data Sources** preferences panel is opened from menu **Window > Preferences > oXygen > Data Sources**.

Configuration of Data Sources

Here you can configure data sources and connections to relational databases as well as native XML databases. You can check the list of drivers (http://www.oxygenxml.com/database_drivers.html) available for the major database servers.

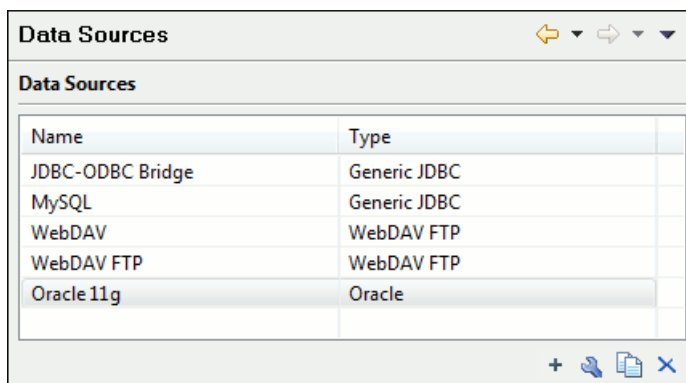


Figure 212: The Data Sources Preferences Panel

- **New** - Opens the **Data Sources Drivers** dialog that allows you to configure a new database driver.

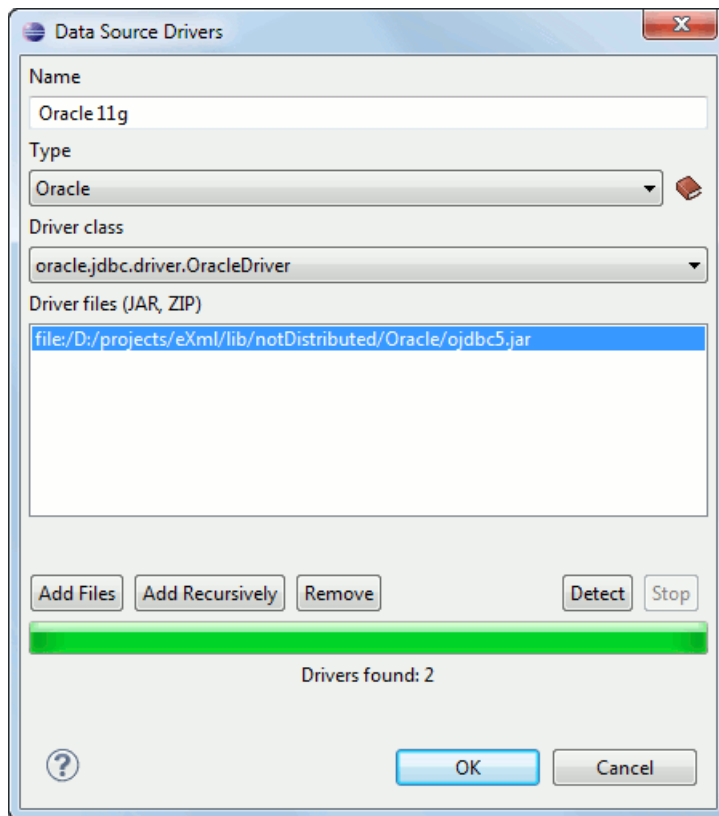


Figure 213: The Data Sources Drivers Dialog

The fields of the dialog are the following:

- **Name** - The name of the new data source driver that will be used for creating connections to the database
- **Type** - Selects the data source type from the supported driver types.
- **Help** - Opens the User Manual at [the list of the sections](#) where the configuration of supported data sources is explained and the URLs for downloading the database drivers are specified.
- **Driver Class** - Specifies the driver class for the data source driver.
- **Add** - Adds the driver class library.
- **Remove** - Removes the selected driver class library from the list.
- **Detect** - Detects driver class candidates.
- **Stop** - Stops the detection of the driver candidates.
- **Edit** - Opens the **Data Sources Drivers** dialog for editing the selected driver. See above the specifications for the **Data Sources Drivers** dialog. In order to edit a data source, there must be no connections using that data source driver.
- **Delete** - Deletes the selected driver. In order to delete a data source, there must be no connections using that data source driver.

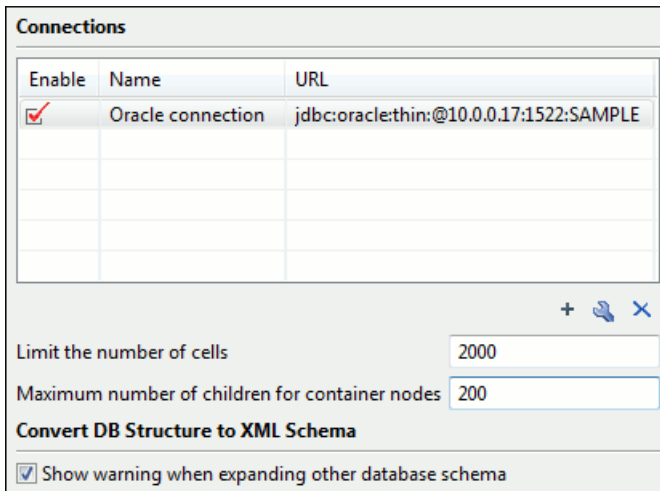


Figure 214: The Connections Preferences Panel

For performance issues, you can set the maximum number of cells that will be displayed in the **Table Explorer** view for a database table. Leave the field **Limit the number of cells** empty if you want the entire content of the table to be displayed. By default this field is set to 2,000. If a table having more cells than the value set here is displayed in the **Table Explorer** view, a warning dialog will inform you that the table is only partially shown.

In Oracle XML and Tamino databases a container can hold millions of resources. If the node corresponding to such a container in the **Data Source Explorer** view would display all the contained resources at the same time the performance of the view would be very slow. To prevent such a situation only a limited number of the contained resources is displayed as child nodes of the container node. Navigation to other contained resources from the same container is enabled by the *Up* and *Down* buttons in the **Data Source Explorer** view. This limited number is set in the option **Maximum number of children for container nodes**. The default value is 200 nodes.

The **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:

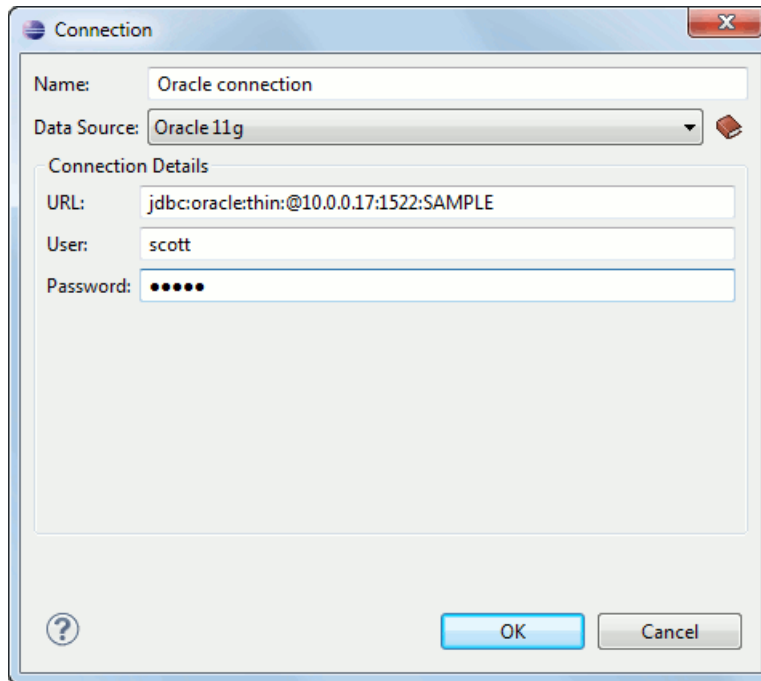


Figure 215: The Connection Dialog

- **Name** - The name of the new connection that will be used in transformation scenarios and validation scenarios.
- **Data Source** - Allows selecting a data source defined in the **Data Source Drivers** dialog.

Depending upon the selected data source, you can set some of the following parameters in the **Connection details** area:

- **URL** - The URL for connecting to the database server.
 - **User** - The user name for connecting to the database server.
 - **Password** - The password of the specified user name.
 - **Host** - The host address of the server.
 - **Port** - The port where the server accepts the connection.
 - **XML DB URI** - The database URI.
 - **Database** - The initial database name.
 - **Collection** - One of the available collections for the specified data source.
 - **Environment home directory** - Specifies the home directory (only for a Berkeley database).
 - **Verbosity** - Sets the verbosity level for output messages (only for a Berkeley database).
- **Edit** - Opens the **Connection** dialog, allowing you to edit the selected connection. See above the specifications for the **Connection** dialog.
 - **Duplicate** - Creates a duplicate of the currently selected connection.
 - **Delete** - Deletes the selected connection.

Download Links for Database Drivers

You can find below the locations where you have to go to get the drivers necessary for accessing databases in Oxygen XML Developer plugin .

- **Berkeley DB XML database** - Copy the jar files from the Berkeley database install directory to 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 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 Java and .NET XCC distributions (XCC Connectivity Packages) from [MarkLogic](#). You find the details about configuring a MarkLogic data source in [the procedure for creating a MarkLogic data source](#).
- **Microsoft SQL Server 2005 / 2008 database** - Both SQL Server 2005 and SQL Server 2008 are supported. For connecting to SQL Server 2005 you have to download the SQL Server 2005 JDBC driver file `sqljdbc.jar` from the [Microsoft website](#) and use it for [configuring an SQL Server data source](#). For connecting to SQL Server 2008 you have to download the SQL Server 2008 JDBC 1.2 driver file `sqljdbc_1.2\enu\sqljdbc.jar` from the [Microsoft website](#) and use it for [configuring an SQL Server data source](#). Please note that the SQL Server driver is compiled with a Java 1.6 compiler so you need to run Oxygen XML Developer plugin with a Java 1.6 virtual machine in order to use this driver.
- **Oracle 11g database** - Download the Oracle 11g JDBC driver called `ojdbc5.jar` from the [Oracle website](#) and use it for [configuring an Oracle data source](#).
- **PostgreSQL 8.3 database** - Download the PostgreSQL 8.3 JDBC driver called `postgresql-8.3-603.jdbc3.jar` from the [PostgreSQL website](#) and use it for [configuring a PostgreSQL data source](#).
- **RainingData TigerLogic XDMS database** - Copy the jar files from the TigerLogic JDK lib directory from the server side to the Oxygen XML Developer plugin install directory as described in [the procedure](#) for configuring a TigerLogic data source.
- **SoftwareAG Tamino database** - Copy the jar files from the `SDK\TaminoAPI4J\lib` subdirectory of the Tamino database install directory to the Oxygen XML Developer plugin install directory as described in [the procedure](#) for configuring a Tamino data source.
- **Documentum xDb (X-Hive/DB) 10 XML database** - Copy the jar files from the Documentum xDb (X-Hive/DB) 10 database install directory to the Oxygen XML Developer plugin install directory as described in [the procedure](#) for configuring a Documentum xDb (X-Hive/DB) 10 data source.

Table Filters

The **Table Filters** preferences panel is opened from menu **Window > Preferences > oXygen > Data Sources > Table Filters**.

Here you can choose which of the database table types will be displayed in the **Data Source Explorer** view.

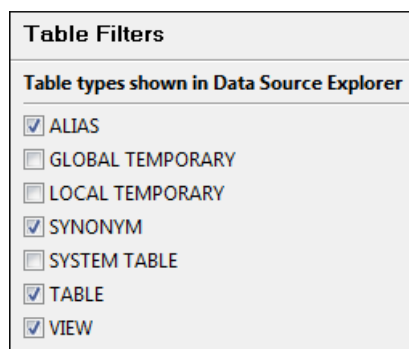


Figure 216: Table Filters Preferences Panel

Archive

The **Archive** preferences panel is opened from menu **Window > Preferences > oXygen > 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.

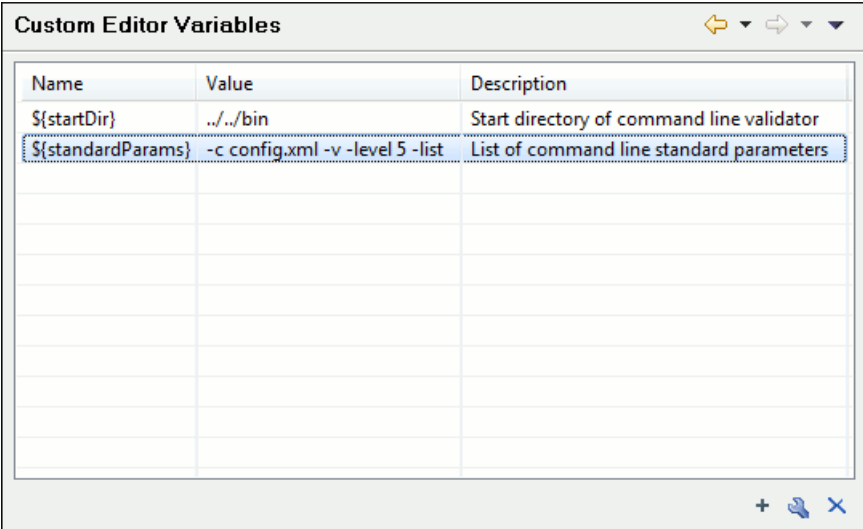
- **Show archive backup dialog** - Select this option if you want to be notified for backup when modifying in archives. The last backup option you chose will always be used as the default one.
- **Archive types** - This table contains all known archive extensions mapped to known archive formats. Each 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 an archive is modified in any way with this option turned on, UTF-8 characters are used in the names of all files in the archive.

Custom Editor Variables

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.



Name	Value	Description
<code>\${startDir}</code>	<code>../../bin</code>	Start directory of command line validator
<code>\${standardParams}</code>	<code>-c config.xml -v -level 5 -list</code>	List of command line standard parameters

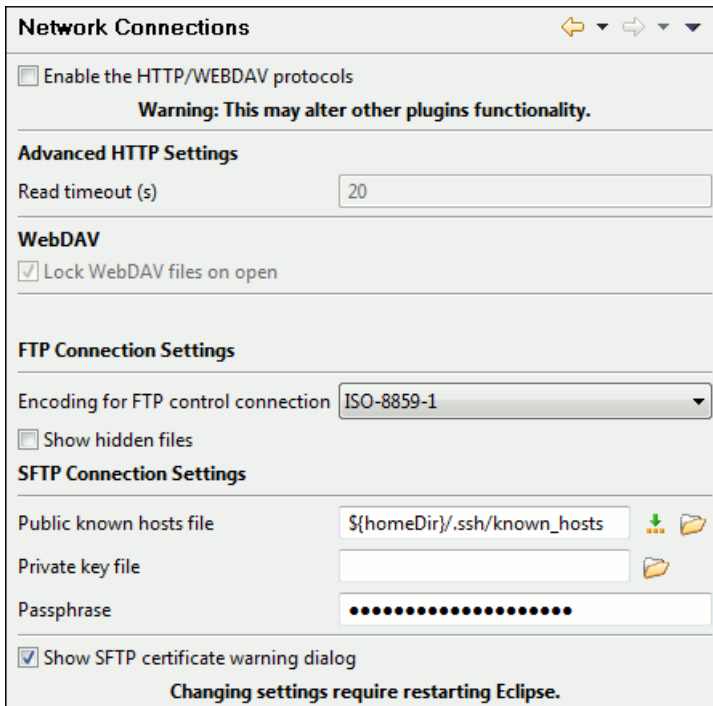
Figure 217: Custom Editor Variables

Network Connections

Some networks use proxy servers to provide Internet services to LAN clients. Clients behind the proxy may therefore, only connect to the Internet via the proxy service. If you are not sure whether your computer is required to use a proxy server to connect to the Internet or you don't know the proxy parameters, please consult your network administrator.

You can open the Network Connections panel from menu **Window > Preferences > oXygen / Network Connections**.

The Network Connections Preferences Panel



Complete the dialog as follows:

- **Enable the HTTP/WEBDAV protocols** - If checked, the HTTP(S) connections go through the proxy with the host, port, user name and password that are set in Eclipse's built-in general *Network Connections* preferences panel.
 - 👉 **Important:** This may affect other plugins functionality.
- **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.
- **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.
- **Private key file** - The path to the file containing the private key used for the private key method of authentication of the secure FTP (SFTP) protocol. The user / password method of authentication has precedence if it is used in *the Open URL dialog*.
- **Passphrase** - The passphrase used for the private key method of authentication of the secure FTP (SFTP) protocol. The user / password method of authentication has precedence if it is used in *the Open URL dialog*.
- **Show SFTP certificate warning dialog** - If checked, a warning dialog will be shown each time when the authenticity of the host cannot be established.

Certificates

In *Oxygen XML Developer plugin* there are provided two types of keystores for certificates used for digital signatures of XL documents: Java KeyStore (JKS) and Public-Key Cryptography Standards version 12 (PKCS-12). A keystore file is protected by a password. A certificate keystore is configured in *Oxygen XML Developer plugin* in the **Certificates** preferences panel which is opened from menu **Window > Preferences > oXygen > Certificates**. The parameters of a keystore are the following:

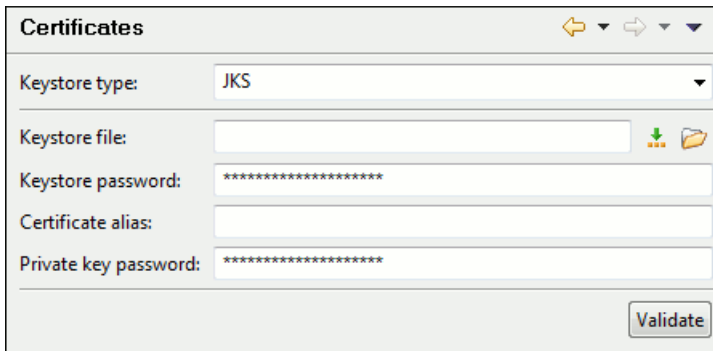


Figure 218: The Certificates Preferences Panel

- **Keystore type** - Represents the type of keystore to be used.
- **Keystore file** - Represents the location of the file to be imported.
- **Keystore password** - The password which is used to protect the privacy of the stored keys.
- **Certificate alias** - The alias to be used to store the key entry (the certificate and / or the private key) inside the keystore.
- **Private key password** - It is only necessary in case of JKS keystore. It represents the certificate's private key password.
- **Validate** - Pressing this button verifies the keystore configured with the above parameters and assures that the certificate is valid.

XML Structure Outline

The **XML Structure Outline** preferences panel is opened from menu **Window > Preferences > oXygen > XML Structure Outline** and contains the following preferences:

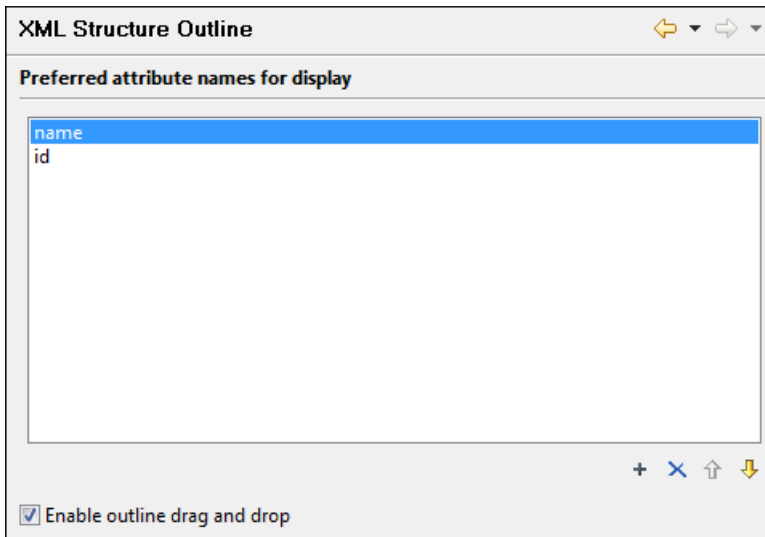


Figure 219: The XML Structure Outline Preferences Panel

- **Preferred attribute names for display** - The attribute names which should be preferred when displaying the element's attributes in the **Outline** view. If there is no preferred attribute name specified the first attribute of an element is displayed.
- **Enable outline drag and drop** - Drag and drop should be disabled for the tree displayed in the **Outline** view only of there is a possibility to accidentally change the structure of the document by such drag and drop operations.

Scenarios Management

The **Scenarios Management** preferences panel is opened from menu **Window > Preferences > oXygen > Scenarios Management** and allows sharing the global transformation scenarios with other users by exporting them to an external file that can be also imported in this preferences panel.



Figure 220: 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

The **View** preferences panel is opened from menu **Window > Preferences > oXygen > View** and contains the following preferences:

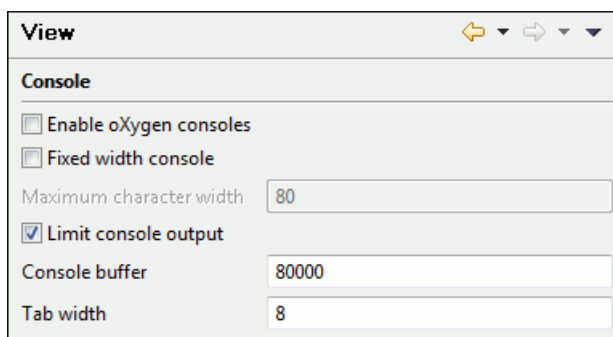


Figure 221: The View Preferences Panel

- **Fixed width console** - If checked, a line in the **Console** view will be hard wrapped after the maximum numbers of characters allowed on a line.
- **Limit console output** - If checked, the content of the **Console** view will be limited to a configurable number of characters.





- **Console buffer** - Specifies the maximum number of characters that can be written in the **Console** view.
- **Tab width** - Specifies the number of spaces used for depicting a tab character.

Reset Global Options

To reset all global preferences to their default values you have to go to menu **Window > Preferences > oXygen > Reset Global Options**. The list of transformation scenarios will be reset to the default scenarios.

Scenarios Management

You can import, export and reset the global scenarios for transformation and validation with the following actions:

- The action **Window > Preferences > oXygen / Scenarios Management >  Import Global Transformation Scenarios** loads a set of transformation scenarios from a properties file that was created with the action **Export Global Transformation Scenarios**.
- The action **Window > Preferences > oXygen / Scenarios Management >  Export Global Transformation Scenarios** stores all the global (not project-level) transformation scenarios in a properties file that can be used later by the action **Import Global Transformation Scenarios**.
- The action **Window > Preferences > oXygen / Scenarios Management >  Import Global Validation Scenarios** loads a set of validation scenarios from a properties file that was created with the action **Export Global Validation Scenarios**.
- The action **Window > Preferences > oXygen / Scenarios Management >  Export Global Validation scenarios** stores all the global (not project-level) validation scenarios in a separate properties file.

The options of **Export Global Transformation Scenarios** and **Export Global Validation Scenarios** is used to store all the scenarios in a separate file which is a properties file. In this file will also be saved the associations between document URLs and scenarios. You can load the saved scenarios using the actions **Import Global Transformation Scenarios** and **Import Global Validation Scenarios**. All the imported scenarios will have added to the name the word **import** to distinguish the existing scenarios and the imported ones.

Editor Variables

An editor variable is a shorthand notation for context-dependent information, like a file or folder path, a timestamp or a date. It is used in the definition of a command (for example the input URL of a transformation, the output file path of a transformation, the command line of an external tool) to make a command or a parameter generic and reusable with other input files. When the same command is applied to different files, the notation is expanded at the execution of the command so that the same command has different effects depending on the actual file.

The following editor variables can be used in Oxygen XML Developer plugin commands of external engines or other external tools, in transformation scenarios, validation scenarios:

- **`\${oxygenHome}** - Oxygen XML Developer plugin installation folder as URL.
- **`\${oxygenInstallDir}** - Oxygen XML Developer plugin installation folder as file path.
- **`\${frameworks}** - The path (as URL) of the `frameworks` subfolder of the Oxygen XML Developer plugin install folder.
- **`\${frameworksDir}** - The path (as file path) of the `frameworks` subfolder of the Oxygen XML Developer plugin installation folder.
- **`\${home}** - The path (as URL) of the user home folder.
- **`\${homeDir}** - The path (as file path) of the user home folder.
- **`\${pdu}** - Current project folder as URL.
- **`\${pd}** - Current project folder as file path.
- **`\${pn}** - Current project name.

- **`\${cfdu}`** - Current file folder as URL, that is the path of the current edited document up to the name of the parent folder, represented as a URL.
- **`\${cfd}`** - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- **`\${cfn}`** - Current file name without extension and without parent folder.
- **`\${cfne}`** - Current file name with extension.
- **`\${cf}`** - Current file as file path, that is the absolute file path of the current edited document.
- **`\${cfu}`** - The path of the current file as a URL.
- **`\${af}`** - The file path of the zip archive that includes the current transformed file (this variable is available only in a transformation scenario).
- **`\${afu}`** - The URL of the zip archive that includes the current transformed file (this variable is available only in a transformation scenario).
- **`\${afd}`** - The directory of the zip archive that includes the current transformed file (this variable is available only in a transformation scenario).
- **`\${afdu}`** - The URL of the directory of the zip archive that includes the current transformed file (this variable is available only in a transformation scenario).
- **`\${afn}`** - The file name (without parent directory and without file extension) of the zip archive that includes the current transformed file (this variable is available only in a transformation scenario).
- **`\${afne}`** - The file name (with file extension, for example `.zip` or `.epub`, but without parent directory) of the zip archive that includes the current transformed file (this variable is available only in a transformation scenario).
- **`\${currentFileURL}`** - Current file as URL, that is the absolute file path of the current edited document represented as URL.
- **`\${ps}`** - Path separator, that is the separator which can be used on the current platform (Windows, Mac OS X, Linux) between library files specified in the class path.
- **`\${timeStamp}`** - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- **`\${caret}`** - The position where the caret is inserted. This variable can be used in a or in a
- **`\${selection}`** - The XML content of the current selection in the editor panel. This variable can be used in a
- **`\${id}`** - Application-level unique identifier.
- **`\${uuid}`** - Universally unique identifier.
- **`\${env(VAR_NAME)}`** - Value of the `VAR_NAME` environment variable.
- **`\${ask('user-message', param-type, 'default-value' ?)}`** - To prompt for values at runtime, use the `ask('user-message', param-type, 'default-value' ?)` editor variable. The following parameters can be set:
 - `'user-message'` - the actual message to be displayed. Note the quotes that enclose the message.
 - `param-type` - optional parameter. Can have one of the following values:
 - `url` - input is considered to be an URL. Oxygen XML Developer plugin checks that the URL is valid before passing it to the transformation.
 - `password` - input characters are hidden.
 - `generic` - the input is treated as generic text that requires no special handling.
 - `'default-value'` - optional parameter. Provides a default value in the input text box.

Examples:

- ``${ask('message')}`` - Only the message displayed for the user is specified.
- ``${ask('message', generic, 'default')}`` - 'message' will be displayed for the user, the type is not specified (the default is string), the default value will be 'default'.
- ``${ask('message', password)}`` - 'message' will be displayed for the user, the characters typed will be replaced with a circle character.
- ``${ask('message', password, 'default')}`` - Same as above, default value will be 'default'.
- ``${ask('message', url)}`` - 'message' will be displayed for the user, the type of parameter will be URL.
- ``${ask('message', url, 'default')}`` - Same as above, default value will be 'default'.

- **`\${system(var.name)}`** - Value of the *var.name* system variable.
- **`\${date(pattern)}`** - Current date. Follows the given pattern. Example: yyyy-MM-dd.
- **`\${dbgXML}`** - The path to the current Debugger source selection (for tools started from the XSLT/XQuery Debugger).
- **`\${dbgXSL}`** - The path to the current Debugger stylesheet selection (for tools started from the XSLT/XQuery Debugger).
- **`\${tsf}`** - The transformation result file.
- **`\${ps}`** - The path separator which can be used on different operating systems between libraries specified in the class path.
- **`\${dsu}`** - The path of the detected schema as a URL.
- **`\${ds}`** - The path of the detected schema as a local file path.
- **`\${cp}`** - Current page number.
- **`\${tp}`** - Total number of pages in the document.

Custom Editor Variables

An editor variable can be created by the user and included in any user defined expression where a built-in editor variable is also allowed. For example a custom editor variable may be necessary for configuring the command line of an external tool, the working directory of a custom validator, the command line of a custom XSLT engine, a custom FO processor, etc. All the custom editor variables are listed together with the built-in editor variables, for example when editing the working folder or the command line of a , the working directory, etc.

Creating a custom editor variable is very simple: just specify the name that will be used in user defined expressions, the value that will replace the variable name at runtime and a textual description for the user of that variable.

The custom editor variables are

Chapter 19

Common Problems

Topics:

- [*XSLT Debugger Is Very Slow*](#)
- [*Syntax Highlight Not Available in Eclipse Plugin*](#)
- [*Problem Report Submitted on the Technical Support Form*](#)
- [*Signature verification failed error on open or edit a resource from Documentum*](#)

This chapter presents common problems that may appear when running the application and the solutions for these problems.

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 with *the option Open in browser* 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 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 plugin.
 - a) Go to menu **Windows > Preferences > General > Editors > File Associations**.
 - b) Add `*.ext` to the list of file types.
 - c) Select `*.ext` in the list by clicking on it.
 - d) Add Oxygen XML 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) Go to menu **Windows > Preferences > General > Content Types**.
 - b) Add `*.ext` to the **File associations** list for the **Text > XML > oXygen XML** content type.
3. Press the **OK** button in the Eclipse preferences dialog.

Now when an `*.ext` file is opened the icon of the editor and the syntax highlight should be the same as for XML files opened with the Oxygen XML Developer plugin plugin.

Problem Report Submitted on the Technical Support Form

What details should I add to my problem report that I enter on the Technical Support online form of the product website?

For problems like server connection error, unexpected delay while editing a document, a crash of the application, etc for which the usual details requested on the Technical Support online form are not enough you should generate a log file and attach it to the problem report. In case of a crash you should also attach the crash report file generated by your operating system. For generating a logging file you need to create a text file called `log4j.properties` in the install folder with the following content:

```
log4j.rootCategory= debug, R2

log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=logging.log
log4j.appender.R2.MaxFileSize=12000KB
log4j.appender.R2.MaxBackupIndex=20
log4j.appender.R2.layout=org.apache.log4j.PatternLayout
log4j.appender.R2.layout.ConversionPattern=%r %p [ %t ] %c - %m%n
```

Restart the application, reproduce the error and close the application. The log file is called `logging.log` and is located in the install folder.

Signature verification failed error on open or edit a resource from Documentum

When I try to open/edit a resource from Documentum, I receive the following error:

```
signature verification failed: certificate for All-MB.jar.checksum not signed  
by a certification authority.
```

The problem is that the certificates from the Java Runtime Environment 1.6.0_22 or later no longer validate the signatures of the UCF jars.

Edit the `eclipse.ini` file from the Eclipse directory and add the following parameter to the `-vmargs`:

`-Drequire.signed.ucf.jars=false`, for example:

```
-vmargs  
-Xms40m  
-Xmx256m  
-Drequire.signed.ucf.jars=false
```

Chapter 20

Terms

Active cell	The selected cell in which data is entered when you begin typing. Only one cell is active at a time. The active cell is bounded by a heavy border.
Block	A block is an element that takes up the entire available width. A block is delimited by line breaks before and after it.
Inline	An inline element takes up as much width as necessary, and does not force line breaks.
Inline XML element	Inline elements are elements which appear in a mixed-content context (parents with both non-whitespace text and elements).

Index

A

- Archives 247, 248, 250
 - browse 248
 - edit 250
 - file browser 248
 - modify 248

C

- Common problems 373
- Composing Web Service calls 307, 308, 310, 311
 - generate WSDL documentation 311
 - SOAP request 308, 310
 - testing remote WSDL files 310
 - UDDI Registry browser 310
- Configure the application 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 349, 350, 351, 352, 354, 355, 357, 358, 359, 360, 363, 364, 365, 366, 367, 368, 369
 - (S)FTP configuration 365
 - Archive 364
 - certificates 366
 - CSS validator 342
 - custom validation 341
 - Data Sources 360, 363, 364
 - download links for database drivers 363
 - table filters 364
 - document type association 328
 - Editor preferences 329, 330, 331, 332, 333, 334, 335, 338, 339, 340
 - code templates 339
 - colors 338
 - content completion 335
 - document checking 340
 - document templates 339
 - elements and attributes by prefix 338
 - format 333
 - format - CSS 335
 - format - XML 334
 - grid 332
 - open/save 339
 - pages 330
 - print 330
 - schema design 332
 - spell check 340
 - syntax highlight 338
 - text/diagram 331, 333
 - editor variables 369
 - fonts 328
 - global 328
 - HTTP configuration 365
 - HTTP proxy configuration 365
 - import 358, 359
 - date/time patterns 359
 - import/export global options 326
 - license 328
 - Network Connections 365

Configure the application (*continued*)

- outline 367
 - reset global options 369
 - scenarios management 368, 369
 - view 368
 - XML 343
 - XML catalog 343
 - XML Instances Generator 344
 - XML parser 343, 344
 - Saxon EE Validation 344
 - XProc 345
 - XSLT 347
 - XSLT/FO/XQuery 346
 - XSLT/FO/XQuery preferences 347, 349, 350, 351, 352, 354, 355, 357
 - custom engines 357
 - debugger 354
 - FO Processors 355
 - MSXML 350
 - MSXML.NET 351
 - profiler 354
 - Saxon6 347
 - Saxon HE/PE/EE 347, 352
 - Saxon HE/PE/EE advanced options 349, 354
 - XPath 357
 - XQuery 352
 - XSLTProc 350
 - Content Management System 299
 - copy/paste 188
 - grid editor 188
- ## D
- Databases 221, 251, 252, 271, 273, 285, 286, 288, 289
 - Native XML databases (NXD) 271, 273
 - Relational databases 252
 - WebDAV Connection 289
 - XQuery 221, 285, 286, 288
 - debugging 288
 - drag and drop from Data Source Explorer 285
 - transformation 286
 - validation 221
 - Debugging XSLT/XQuery documents 228, 230, 239, 242
 - Java extensions 242
 - layout 228, 230, 239
 - information views 230
 - multiple output documents in XSLT 2.0 239
 - XSLT/XQuery debugger 239
 - Debugging XSLT / XQuery documents 229
 - layout 229
 - Control toolbar 229
 - Digital signature 313, 315, 316, 317
 - canonicalizing files 315
 - certificates 316
 - signing files 316
 - verifying the signature 317
 - DITA MAP document type 180, 181
 - association rules 180

DITA MAP document type (*continued*)
 Author extension 180, 181
 catalogs 180
 templates 181
 transformation scenarios 180
 schema 180

DITA Topics document type 179
 association rules 179
 Author extensions 179
 catalogs 179
 templates 179
 transformation scenarios 179
 schema 179

DocBook Targetset document type 179
 association rules 179
 Author extensions 179
 templates 179
 schema 179

DocBook V4 document type 172
 association rules 172
 Author extensions 172
 catalogs 172
 templates 172
 transformation scenarios 172
 schema 172

DocBook V5 document type 175, 176
 association rules 175
 Author extensions 175, 176
 catalogs 175
 templates 176
 transformation scenarios 175
 schema 175

Documentum (CMS) Support 300, 301, 302, 303
 actions 301, 302, 303
 cabinets/folders 302
 connection 302
 resources 303
 configuration 300, 301
 connection 301
 data source 300

E

edit 35, 36, 40, 44, 169, 250
 archives 250
 close documents 44
 create new documents 36
 file properties 44
 open and close documents 36
 open current document in System Application 44
 open read-only files 169
 open remote documents (FTP/SFTP/WebDAV) 40
 save documents 40
 Unicode documents 36
 Unicode support 36

Editing CSS stylesheets 161, 162, 163
 content completion 162
 folding 163
 format and indent (pretty print) 163
 other editing actions 163
 Outline view 163
 validation 161

Editing JSON Documents 163, 164, 165, 166
 Convert XML to JSON 166
 folding 164
 Grid Mode 165
 syntax highlight 164
 Text Mode 164
 Validating JSON Documents 165

Editing NVDL schemas 135
 Component Dependencies View 135

Editing NVDL Schemas 133, 134, 135
 editor specific actions 135
 schema diagram 133, 134, 135
 actions in the diagram view 134
 full model view 134
 Outline view 135
 searching and refactoring actions 135

Editing RelaxNG Schemas 131
 Component Dependencies View 131

Editing Relax NG schemas 125, 126, 127, 128, 129, 130
 editor specific actions 129
 schema diagram 125, 126, 127, 128
 actions 128
 full model view 126
 logical model view 127
 Outline view 128
 symbols 126
 searching and refactoring actions 130

Editing Relax NG Schemas 130
 Resource Hierarchy/Dependencies View 130

Editing XML documents 44, 46, 47, 49, 53, 54, 55, 56, 65, 66, 68, 69, 71, 72, 74, 75, 76, 77, 78
 associate a schema to a document 46, 47, 49
 add schema association in XML instance 47
 learning a document structure 49
 setting a default schema 46
 converting between schema languages 72
 document navigation 65, 66

 folding 65
 outline view 66
 editor specific actions 75, 76, 77, 78
 document actions 76
 edit actions 75
 refactoring actions 76
 select actions 75
 smart editing 77
 source actions 75
 syntax highlight depending on namespace prefix 78

formatting and indenting documents (pretty print) 74
 grouping documents in XML projects 44, 68
 large documents 68
 new project 44
 project view 44

including document parts with XInclude 69
 status information 74
 streamline with content completion 49, 53, 54, 55, 56
 code templates 56
 the Attributes view 54
 the Elements view 55
 the Entities view 55
 the Model panel 53

working with XML Catalogs 71

- Editing XML Schemas 78, 107, 111, 112, 114, 117, 119, 122, 123
 - Component Dependencies View 123
 - generate documentation for XML Schema 112, 114, 117
 - as HTML 114
 - as PDF, DocBook or custom format 117
 - from command line 117
 - relational database table to XML schema 107
 - Resource Hierarchy/Dependencies View 122
 - schema instance generator 107
 - schema regular expressions builder 111
 - Searching and refactoring actions 119
- Editing XQuery documents 158, 159, 160
 - folding 159
 - generate HTML documentation 160
- Editing XSL Stylesheets 157
 - Component Dependencies View 157
- Editing XSLT stylesheets 136, 137, 138, 142, 144, 154, 155
 - content completion 137, 138, 142
 - code templates 142
 - in XPath expressions 138
 - find XSLT references and declarations 154
 - Outline View 144
 - refactoring actions 154
 - Resource Hierarchy/Dependencies View 155
 - validate 136, 137
 - custom validation 136
 - validation scenario 137
 - XSLT Input View 142
- Editing XSLT Stylesheets 146, 149, 151, 152
 - generate documentation for XSLT stylesheet 146
 - generate documentation for XSLT Stylesheets 149, 151, 152
 - as HTML 149
 - from command line 152
 - in custom format 151
 - XSLT stylesheet documentation 146

F

- find/replace 320
 - Find All Elements dialog 320

G

- Getting started 23, 24, 25, 29, 30, 31
 - help 24
 - perspectives 24, 25, 29, 30, 31
 - database 31
 - editor 25
 - XQuery debugger 30
 - XSLT debugger 29
 - supported types of documents 24
- grid editor 185, 186, 187, 188, 190
 - add nodes 187
 - bidirectional text 190
 - clear column content 187
 - copy/paste 188
 - drag and drop 188
 - duplicate nodes 187
 - inserting table column 187
 - insert table row 187
 - layouts (grid vs. tree) 186

- grid editor (*continued*)
 - navigation 186, 187
 - collapse all 187
 - collapse children 187
 - collapse others 187
 - expand all 187
 - expand children 187
 - refresh layout 188
 - sort table column 187
 - start editing a cell value 188
 - stop editing a cell value 188

I

- Importing data 292, 295, 296
 - from database 292, 295
 - convert table structure to XML Schema 295
 - table content as XML document 292
 - from HTML files 295
 - from MS Excel 295
 - from text files 296
- Installation 14
 - environment 14
 - requirements 14

L

- License 14, 16, 17, 18, 20, 21
 - floating (concurrent) license 16
 - floating license server 17
 - floating license servlet 16
 - license server installed as Windows service 18
 - register a license key 14
 - registration code 21
 - release floating license 20

N

- Native XML databases (NXD) 263, 270, 271, 272, 273, 274, 275, 276
 - database connections configuration 273, 274, 275
 - Berkeley DB XML 273
 - Documentum xDb (X-Hive/DB) 275
 - eXist 274
 - Tamino 274
 - TigerLogic 275
 - data sources configuration 270, 271, 272, 273
 - Berkeley DB XML 271
 - Documentum xDb (X-Hive/DB) 273
 - eXist 271
 - MarkLogic 272
 - Tamino 272
 - resource management 263, 276
 - Data Source Explorer view 263, 276

O

- OutOfMemory 21, 201, 333, 339, 355
- Out Of Memory 21, 201, 333, 339, 355
- OutOfMemoryError 21, 201, 333, 339, 355

P

- Performance problems 21
 - external processes 21
- Profiling 244
 - XSLT stylesheets and XQuery documents 244
- Profiling XSLT stylesheets and XQuery documents 244, 245
 - profiling information 244
 - Hotspots view 244
 - Invocation tree view 244
 - XSLT/XQuery profiler 245

Q

- Querying documents 158, 214, 217, 218, 219, 221, 222
 - running XPath expressions 214
 - XPath console 214
 - XQuery 158, 217, 218, 219, 221, 222
 - Input view 219
 - other editing actions 222
 - Outline view 158, 218
 - syntax highlight and content completion 218
 - transforming XML documents; advanced Saxon B/SA options 222
 - validation 221

R

- Relational databases 252, 253, 254, 255, 256, 258, 259, 260, 261, 262, 263, 266, 268, 270, 276, 292, 295
 - connections configuration 258, 259, 260, 261, 262
 - Generic JDBC 261
 - JDBC-ODBC connection 259
 - Microsoft SQL Server 259
 - MySQL 260
 - Oracle 11g 261
 - PostgreSQL 8.3 262
 - creating XML Schema from databases 295
 - data sources configuration 252, 253, 254, 255, 256
 - generic JDBC data source 254
 - IBM DB2 252
 - Microsoft SQL Server 253
 - MySQL 254
 - Oracle 11g 255
 - PostgreSQL 8.3 256
 - importing from databases 292
 - resource management 263, 266, 276
 - Data Source Explorer view 263, 276
 - Table Explorer view 266
 - SQL execution support 268, 270
 - drag and drop from Data Source Explorer 268
 - executing SQL statements 270
 - SQL validation 270

T

- TEI ODD document type 181, 182
 - association rules 181
 - Author extensions 182
 - catalogs 182
 - transformation scenarios 182
 - schema 182

- TEI P4 document type 182, 183
 - association rules 182
 - Author extensions 182, 183
 - catalogs 182
 - templates 182, 183
 - transformation scenarios 182
 - schema 182
- TEI P5 document type 183
 - association rules 183
 - Author extensions 183
 - templates 183
 - transformation scenarios 183
 - schema 183
- Text editor specific actions 319, 321, 323
 - check spelling 321
 - check spelling in files 323
- Transformation scenario 193, 194, 199, 201
 - batch transformation 193
 - built-in transformation scenarios 193
 - new transformation scenario 194, 199, 201
 - additional XSLT stylesheets 201
 - configure transformation scenario 194
 - create a transformation scenario 201
 - XSLT/XQuery extensions 201
 - XSLT parameters 199
- Transforming documents 191, 192, 193, 204, 205, 206, 207
 - custom XSLT processors 206
 - output formats 192
 - supported XSLT processors 205
 - Transformation scenario 193
 - Transformation Scenarios view 204
 - XSL-FO processors 207
 - XSLT processors extensions paths 206

U

- Uninstalling the plugin 21

V

- Validating XML documents 57, 58, 59, 60, 62, 64, 65
 - against a schema 58, 59, 60, 62, 64, 65
 - automatic validation 60
 - caching the schema used for validation 60
 - custom validation 60
 - marking validation errors 59
 - resolving references to remote schemas with an XML Catalog 65
 - validation actions 64
 - validation example 59
 - validation scenario 62
 - checking XML well-formedness 57

W

- WebDAV Connection 289, 290
 - actions 289, 290
 - at connection level 289
 - at file level 290
 - at folder level 289
 - configuration 289

X

- XHTML document type 181
 - association rules 181
 - Author extensions 181
 - catalogs 181
 - templates 181
 - transformation scenarios 181
 - schema 181
- XML Outline view 66, 67, 68
 - document structure change 68
 - popup menu 68
 - document tag selection 68
 - modification follow-up 68
 - outliner filters 67
 - XML document overview 67
- XML Schema Diagram Editor 79, 81, 84, 85, 86, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 102, 104, 105, 106
 - Attributes view 104
 - editing actions 97
 - edit schema namespaces 106
 - Facets view 105, 106
 - editing patterns 106
 - group schema components 94, 95
 - attributes 94
 - constraints 94
 - substitutions 95
 - navigation 95
 - Outline view 102
 - schema components 81, 84, 85, 86, 88, 89, 90, 91, 92, 93
 - xs:any 91
 - xs:anyAttribute 91
 - xs:attribute 84
 - xs:attributeGroup 88
 - xs:complexType 85
 - xs:element 81
 - xs:field 93
 - xs:group 88
 - xs:import 89
 - xs:include 89
- XML Schema Diagram Editor (*continued*)
 - schema components (*continued*)
 - xs:key 92
 - xs:keyRef 93
 - xs:notation 90
 - xs:redefine 89
 - xs:schema 81
 - xs:selector 93
 - xs:sequence, xs:choice, xs:all 90
 - xs:simpleType 86
 - xs:unique 92
 - validation 96
- XML Schema Text Editor 78, 79
 - content completion 78
 - flatten an XML Schema 79
 - XML Schema actions 79
- XQJ connection 223
 - XQJ configuration 223
- XQJ support 223
 - XQJ processor configuration 223
- XSLT/XQuery debugger 231, 232, 233, 234, 235, 236, 237, 238, 239, 240
 - debug steps 239
 - determining what XSLT/XQuery expression generated particular output 240
 - using breakpoints 240
 - inserting breakpoints 240
 - removing breakpoints 240
 - viewing processing information 231, 232, 233, 234, 235, 236, 237, 238
 - breakpoints view 232
 - context node view 231
 - messages view 233
 - node set view 237
 - output mapping stack view 234
 - stack view 234
 - templates view 236
 - trace history view 235
 - variables view 238
 - XPath watch view 231

