Oxygen XML Editor plugin 13.2.0

Contents

Chapter 1

Introduction

Topics:

Welcome to the User Manual of Oxygen XML Editor plugin   13.2.0 .

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

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

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

Key Features and Benefits of Oxygen XML Editor plugin

Multiplatform availability: Windows, Mac OS X, Linux, Solaris	Non blocking operations, you can perform validation and transformation operations in background
Visual WYSIWYG XML editing mode based on W3C CSS stylesheets.	Visual DITA Map editor
Closely integration of the DITA Open Toolkit for generating DITA output	Support for latest versions of document frameworks: DocBook and TEI.
Support for XML, XML Schema, Relax NG , Schematron, DTD, NVDL schemas, XSLT, XSL:FO, WSDL, XQuery, HTML, CSS	Support for XML, CSS, XSLT, XSL-FO.
Validate XML Schema schemas, Relax NG schemas, DTD's, Schematron schemas, NVDL schemas, WSDL, XQuery, HTML and CSS	Manual and automatic validation of XML documents against XML Schema schemas, Relax NG schemas, DTD's, Schematron schemas, and NVDL schemas
Multiple built-in validation engines (Xerces, libxml, MSXML 4.0, MSXML.NET) and support for custom validation engines (Saxon SA, XSV, SQC).	Multiple built-in XSLT transformers (Saxon 6.5, Saxon 9 Enterprise (schema aware), Xalan, libxslt, MSXML 3.0 / 4.0, Microsoft .NET 1.0, Microsoft .NET 2.0), support for custom JAXP transformers.
Visual schema editor with full and logical model views	Generate HTML documentation from XML Schemas
Ready to use FOP support to generate PDF or PS documents	XInclude support
Context sensitive content assistant driven by XML Schema, Relax NG, DTD, NVDL or by the edited document structure enhanced with schema annotation presenter	New XML document wizards to easily create documents specifying a schema or a DTD
XML Catalog support	Unicode support
Conversions from DTD, Relax NG schema or a set of documents to XML Schema, DTD or Relax NG schema	Syntax coloring for XML, DTD, Relax NG compact syntax, Java, C++, C, PHP, Perl, etc
Easy error tracking - locate the error source by clicking on it	Easy configuration for external FO Processors
Apply XSLT and FOP transformations	XPath search, evaluation and debugging support
Preview transformation results as XHTML or XML or in your browser	Support for document templates to easily create and share documents
Import data from a database, Excel, HTML or text file	Convert database structure to XML Schema
Batch validate selected files in project	Canonicalize and sign documents
Configurable actions key bindings	Associate extensions with editors provided by the Oxygen XML Editor plugin plugin.
XSLT Debugger with Backmapping support	XSLT Profiler
XQuery Debugger with Backmapping support	XQuery Profiler
Model View	Attributes View
XQuery 1.0 support	WSDL analysis and SOAP requests support
XSLT 2.0 full support	XPath 2.0 execution and debugging support
Document folding	Spell checking supporting English, German and French including locals

XSLT refactoring actions	Generate large sets of sample XML instances from XML Schema
Pretty-printing of XML files	Drag&drop support
Outline view in sync with a non well-formed document	

Chapter 2

Installation

Topics:

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 Editor plugin) is installed on your computer. The current Eclipse version number is 3.7.

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

Installation Instructions

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

Eclipse Plugin

This section contains the following installation procedures:

- *Eclipse 3.3 Plugin Installation - The Update Site Method*
- *Eclipse 3.3 Plugin Installation - The Zip Archive Method*
- *Eclipse 3.4 - 3.7 Plugin Installation - The Update Site Method*

- *Eclipse 3.4 - 3.7 Plugin Installation - The Zip Archive Method*

Eclipse 3.3 Plugin Installation - The Update Site Method

Installation procedure for the Eclipse plugin in Eclipse 3.3 with the Update Site method.

1. Start Eclipse.
2. Choose the **Help** > **Software Update** > **Find and Install** menu option.
3. Select **Search for new features to install** checkbox.
4. Press **Next**.
5. In the **Update sites to visit** dialog press the button **Add Update Site** or **New Remote Site**.
6. Enter the value `http://www.oxygenxml.com/InstData/Eclipse/site.xml` into the **URL** field of the **New Update Site** dialog.
7. Press **OK**.
8. Select the **oXygen XML Editor** checkbox.
9. Press **Next**.
10. Select the new feature to install **oXygen XML Editor and XSLT debugger**.
11. Press the **Next** button in the following install pages.
12. You must accept the Eclipse restart confirmation.
13. When prompted for a license key, paste the license information received in the registration email.

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

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

Eclipse 3.3 Plugin Installation - The Zip Archive Method

Installation procedure for the Eclipse plugin in Eclipse 3.3 with the Zip Archive method.

1. *Download* the **Eclipse Plugin zip distribution** archive.
2. Unzip the downloaded zip archive in the `plugins` subfolder of the Eclipse install directory.
3. Restart Eclipse.

Eclipse should display an entry *com.oxygenxml.editor (13.2.0)* in the list available from Window - Preferences - Plug-in Development - Target Platform.

Eclipse 3.4 - 3.7 Plugin Installation - The Update Site Method

Installation procedure for the Eclipse plugin in Eclipse 3.4 - 3.7 with the Update Site method.

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

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

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

Eclipse 3.4 - 3.7 Plugin Installation - The Zip Archive Method

The steps for installing the Eclipse plugin in Eclipse 3.4 - 3.7 with the Zip Archive method.

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

Eclipse should display an entry *com.oxygenxml.editor (13.2.0)* in the list available from Window - Preferences - Plug-in Development - Target Platform.

Obtaining and Registering a License Key

Oxygen XML Editor 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.html* .

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

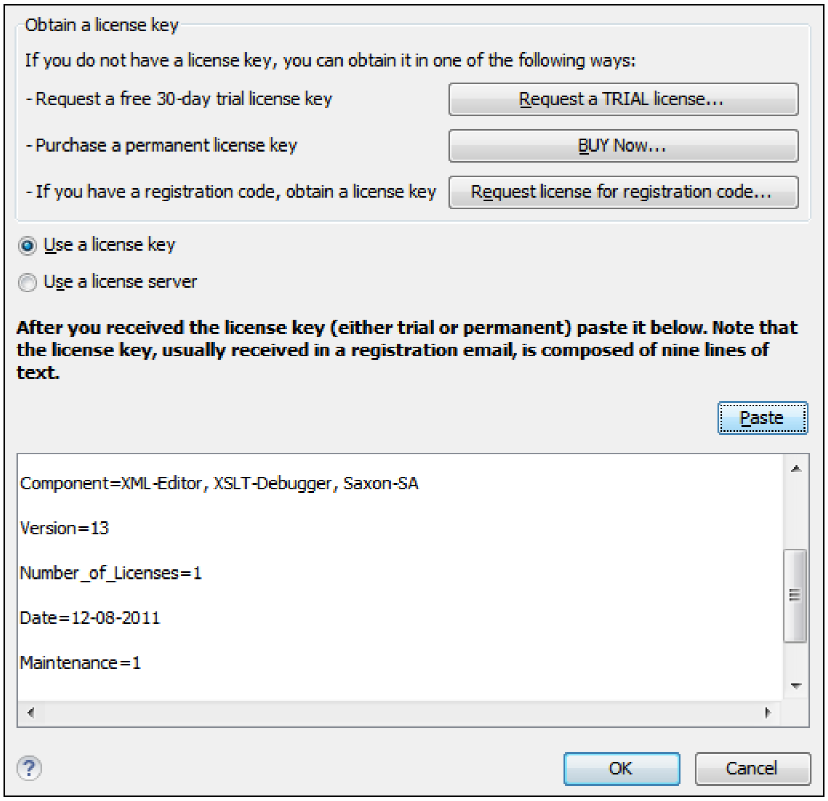
- using the application's Register dialog (the most common case) to register a named-user based or concurrent license;
- storing the license key into a file (either text or xml) that is copied into the application's install folder.

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.



Obtain a license key

If you do not have a license key, you can obtain it in one of the following ways:

- Request a free 30-day trial license key [Request a TRIAL license...]

- Purchase a permanent license key [BUY Now...]

- If you have a registration code, obtain a license key [Request license for registration code...]

● Use a license key

○ Use a license server

After you received the license key (either trial or permanent) paste it below. Note that the license key, usually received in a registration email, is composed of nine lines of text.

[Paste]

```
Component=XML-Editor, XSLT-Debugger, Saxon-SA

Version=13

Number_of_Licenses=1

Date=12-08-2011

Maintenance=1
```

[OK] [Cancel]

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 Editor plugin license server:

- if you plan to use the application on machines running in the same local network, Oxygen XML Editor plugin can manage the licenses usage by itself. Different running instances of 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 Editor plugin floating license server. A floating license server can be installed either as a Java servlet or as a standalone process.

Setting up a Floating License Server Running as a Java Servlet

Setting up the floating license servlet.

Apache Tomcat 5.5 or higher is necessary. 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 Editor plugin XML support staff at *support@oxygenxml.com* and ask for a new license key if:

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

Report Page

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

It displays in real time the following information:

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

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

Setting up a Floating License Server Running as a Standalone Process

Setting up the floating license server.

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 Editor plugin instances. Default value: 12346.

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

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

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

```
licenseServer.bat myLicenseDir 54321
```

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

Install the License Server as a Windows Service

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

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

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

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

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

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

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

Note:

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

Common Problems

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

Windows service reports Incorrect Function when started

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

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

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

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

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

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

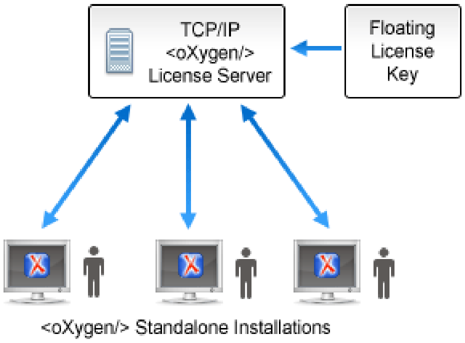


Figure 2: Floating License Server Running as a Standalone Process

1. Start the Eclipse platform.
2. Go to 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 Editor plugin . The license details are displayed in the **About** dialog opened from the **Help** menu. If the maximum number of licenses was exceeded a warning dialog pops up letting you know about the problem. The message contains information about the users who requested and successfully received the floating licenses before exceeding the maximum number of licenses.

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

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

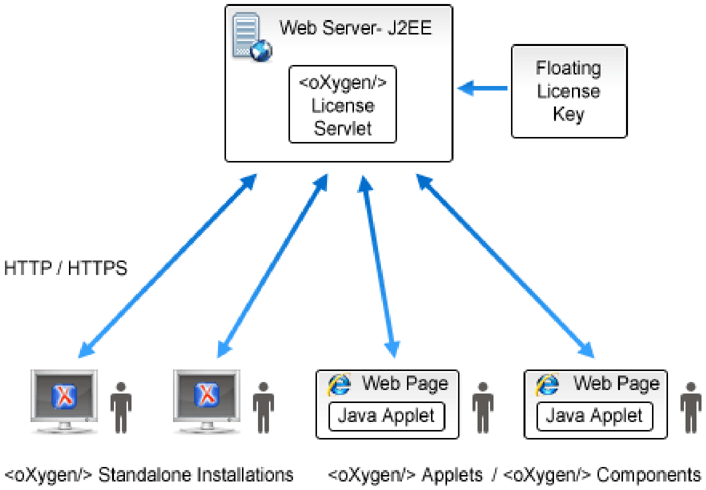


Figure 3: Floating License Server Running as a Servlet

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

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

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

Release a Floating License

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

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

License Registration with an Activation Code

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

Unregistering the License Key

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

1. Go to menu **Windows** > **Preferences** > **oXygen** > **Register**
   This displays the license registration dialog.
2. Make sure the text area for the license key is empty.
3. Make sure the option **Use a license server** is not selected.
4. Press the **OK** button of the dialog.
   This displays a confirmation dialog.
5. Select between falling back to the license key entered previously (for the case of releasing a floating license and reverting to Named User license) and removing your license key from your user account of the computer.

Upgrading the Oxygen XML Editor plugin Application

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

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

Upgrading the Eclipse Plugin

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

Checking for New Versions

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

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

Uninstalling the Application

This section contains uninstallation procedures.

Uninstalling the 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` on Windows (usually %APPDATA% has the value [user-home-dir]\Application Data) / the subfolder `.com.oxygenxml` of the user home directory on Linux / the subfolder `Library/Preferences/com.oxygenxml` of the user home folder on Mac OS X.

Performance Problems

This section contains the solutions for some common problems that may appear when running 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 Editor 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:

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 Editor plugin by going to menu **Help** > **Help Contents** > **Oxygen plugin**

Supported Types of Documents

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

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

Perspectives

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

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

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

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

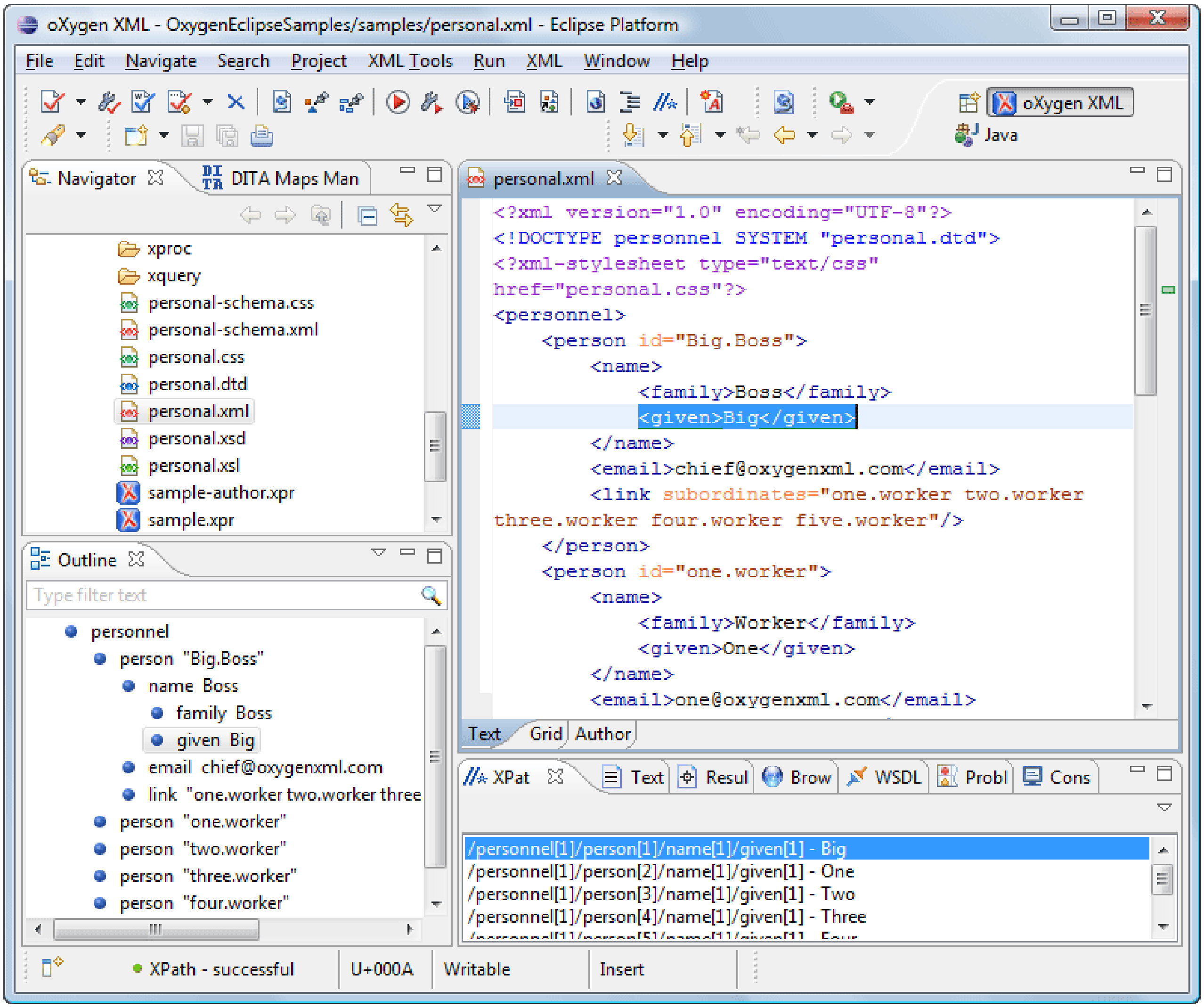
XQuery Debugger perspective
XQuery transforms can be debugged by tracing their execution step by step.

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

Oxygen XML Editor plugin XML Perspective

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

Figure 4: Oxygen XML Editor plugin XML perspective



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE personnel SYSTEM "personal.dtd">
<?xml-stylesheet type="text/css"
href="personal.css"?>
<personnel>
    <person id="Big.Boss">
        <name>
            <family>Boss</family>
            <given>Big</given>
        </name>
        <email>chief@oxygenxml.com</email>
        <link subordinates="one.worker two.worker
three.worker four.worker five.worker"/>
    </person>
    <person id="one.worker">
        <name>
            <family>Worker</family>
            <given>One</given>
        </name>
        <email>one@oxygenxml.com</email>
```

```
/personnel[1]/person[1]/name[1]/given[1] - Big
/personnel[1]/person[2]/name[1]/given[1] - One
/personnel[1]/person[3]/name[1]/given[1] - Two
/personnel[1]/person[4]/name[1]/given[1] - Three
/personnel[1]/person[5]/name[1]/given[1] - Four
```

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 Editor plugin Custom Menu

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

The Oxygen XML Editor plugin Toolbar Buttons

The toolbar buttons added by the Oxygen XML Editor plugin 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 Editor plugin Eclipse plugin. You know the document is associated with Oxygen XML Editor plugin from the special icon displayed in the editor's title bar which has the same graphic pattern painted with different colors for *different types of documents.*

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

The Outline View

The **Outline** view displays a general tag overview of the 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.

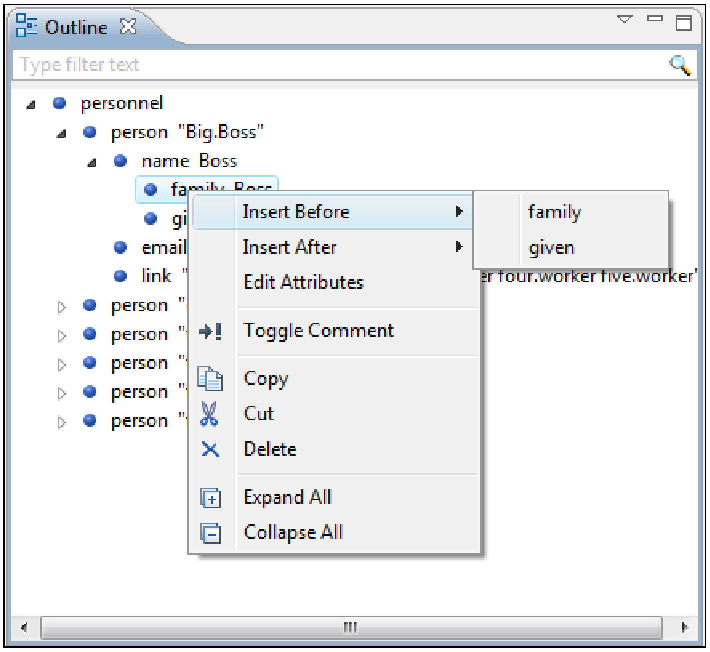
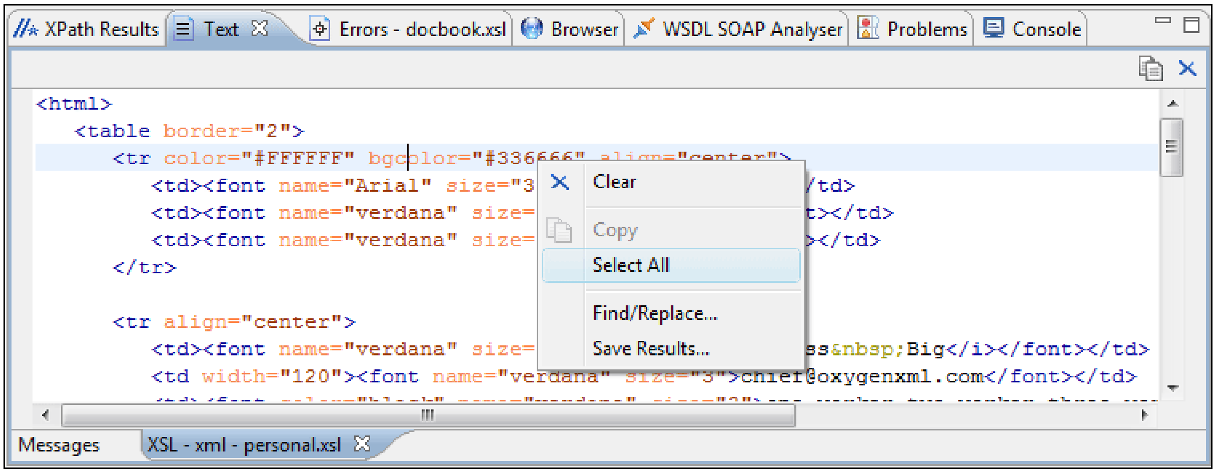


Figure 5: The Outline View

The Oxygen XML Editor plugin Text View

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

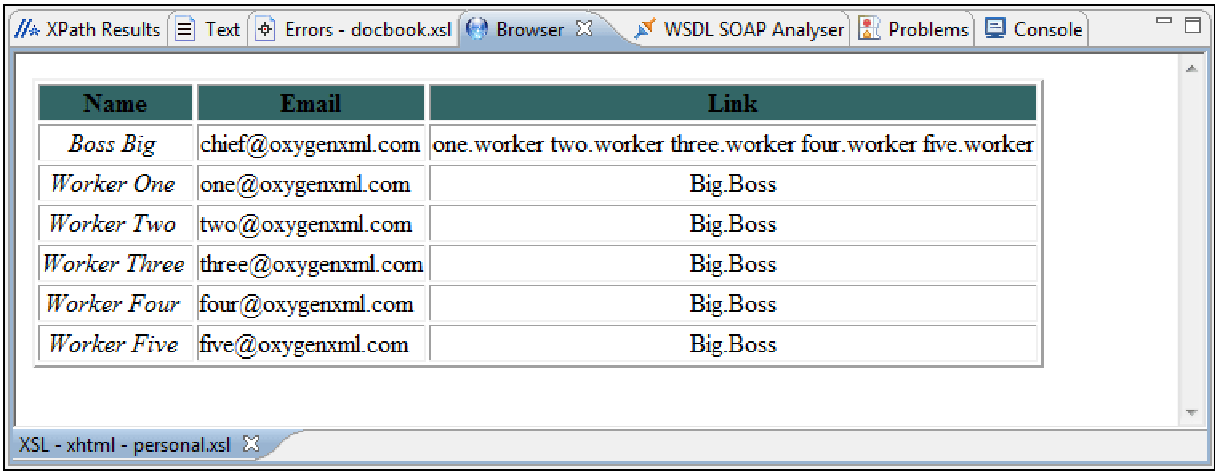


```
<html>
   <table border="2">
      <tr color="#FFFFFF" bgcolor="#336666" align="center">
         <td><font name="Arial" size="3                        /td>
         <td><font name="verdana" size=                         t></td>
         <td><font name="verdana" size=                         ></td>
      </tr>

      <tr align="center">
         <td><font name="verdana" size=                         ss&nbsp;Big</i></font></td>
         <td width="120"><font name="verdana" size="3">chief@oxygenxml.com</font></td>
```

Figure 6: The Text View

The Oxygen XML Editor plugin Browser View

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



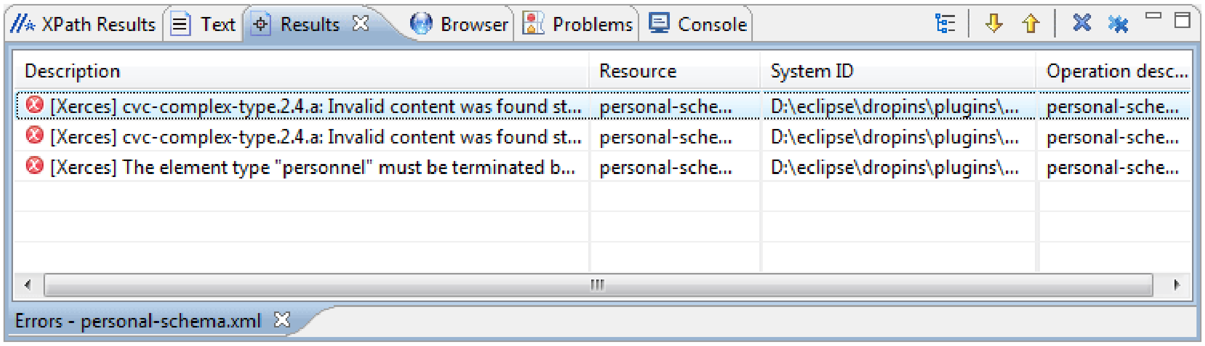
Name	Email	Link
Boss Big	chief@oxygenxml.com	one.worker two.worker three.worker four.worker five.worker
Worker One	one@oxygenxml.com	Big.Boss
Worker Two	two@oxygenxml.com	Big.Boss
Worker Three	three@oxygenxml.com	Big.Boss
Worker Four	four@oxygenxml.com	Big.Boss
Worker Five	five@oxygenxml.com	Big.Boss

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*



Description	Resource	System ID	Operation desc...
[Xerces] cvc-complex-type.2.4.a: Invalid content was found st...	personal-sche...	D:\eclipse\dropins\plugins\...	personal-sche...
[Xerces] cvc-complex-type.2.4.a: Invalid content was found st...	personal-sche...	D:\eclipse\dropins\plugins\...	personal-sche...
[Xerces] The element type "personnel" must be terminated b...	personal-sche...	D:\eclipse\dropins\plugins\...	personal-sche...

Figure 8: Errors View

The actions available on the toolbar of this view are:

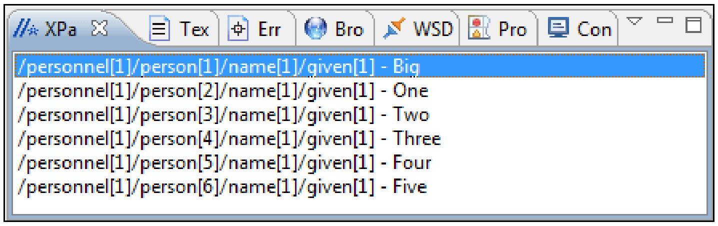
- remove actions: **Remove selected**, **Remove all** - these actions reduce the number of the messages from the view by removing them. It is useful when the view is cluttered by messages that are not important.

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

The Oxygen XML Editor plugin XPath Results View

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



/personnel[1]/person[1]/name[1]/given[1] - Big
/personnel[1]/person[2]/name[1]/given[1] - One
/personnel[1]/person[3]/name[1]/given[1] - Two
/personnel[1]/person[4]/name[1]/given[1] - Three
/personnel[1]/person[5]/name[1]/given[1] - Four
/personnel[1]/person[6]/name[1]/given[1] - Five

Figure 9: The XPath Results View

Supported Editor Types

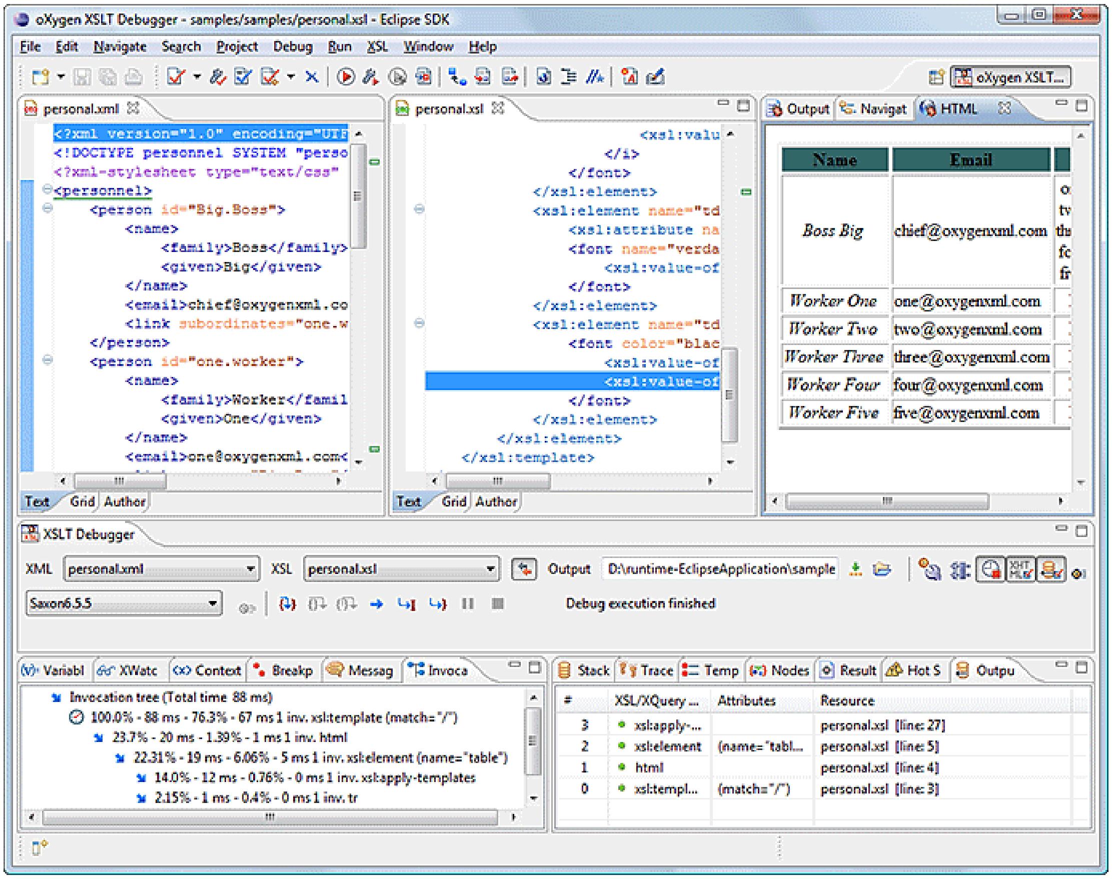
The Oxygen XML Editor plugin Eclipse plugin provides special Eclipse editors identified by the following icons:

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

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

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 other modes (*Author mode*, *Grid mode*) are available only in *the Editor perspective*.



```
<?xml version="1.0" encoding="UTF
<!DOCTYPE personnel SYSTEM "perso
<?xml-stylesheet type="text/css"
<personnel>
    <person id="Big.Boss">
        <name>
            <family>Boss</family>
            <given>Big</given>
        </name>
        <email>chief@oxygenxml.co
        <link subordinates="one.w
    </person>
    <person id="one.worker">
        <name>
            <family>Worker</famil
            <given>One</given>
        </name>
        <email>one@oxygenxml.com<
```

```
                        <xsl:valu
                    </i>
                </font>
            </xsl:element>
            <xsl:element name="td
                <xsl:attribute na
                <font name="verda
                    <xsl:value-of
                </font>
            </xsl:element>
            <xsl:element name="td
                <font color="blac
                    <xsl:value-of
                    <xsl:value-of
                </font>
            </xsl:element>
        </xsl:element>
    </xsl:template>
```

Name	Email	
Boss Big	chief@oxygenxml.com	o tv th fc fr
Worker One	one@oxygenxml.com	
Worker Two	two@oxygenxml.com	
Worker Three	three@oxygenxml.com	
Worker Four	four@oxygenxml.com	
Worker Five	five@oxygenxml.com	

XML: personal.xml XSL: personal.xsl Output: D:\runtime-EclipseApplication\sample

Saxon6.5.5

Debug execution finished

```
Invocation tree (Total time 88 ms)
    100.0% - 88 ms - 76.3% - 67 ms 1 inv. xsl:template (match="/")
        23.7% - 20 ms - 1.39% - 1 ms 1 inv. html
            22.31% - 19 ms - 6.06% - 5 ms 1 inv. xsl:element (name="table")
                14.0% - 12 ms - 0.76% - 0 ms 1 inv. xsl:apply-templates
                2.15% - 1 ms - 0.4% - 0 ms 1 inv. tr
```

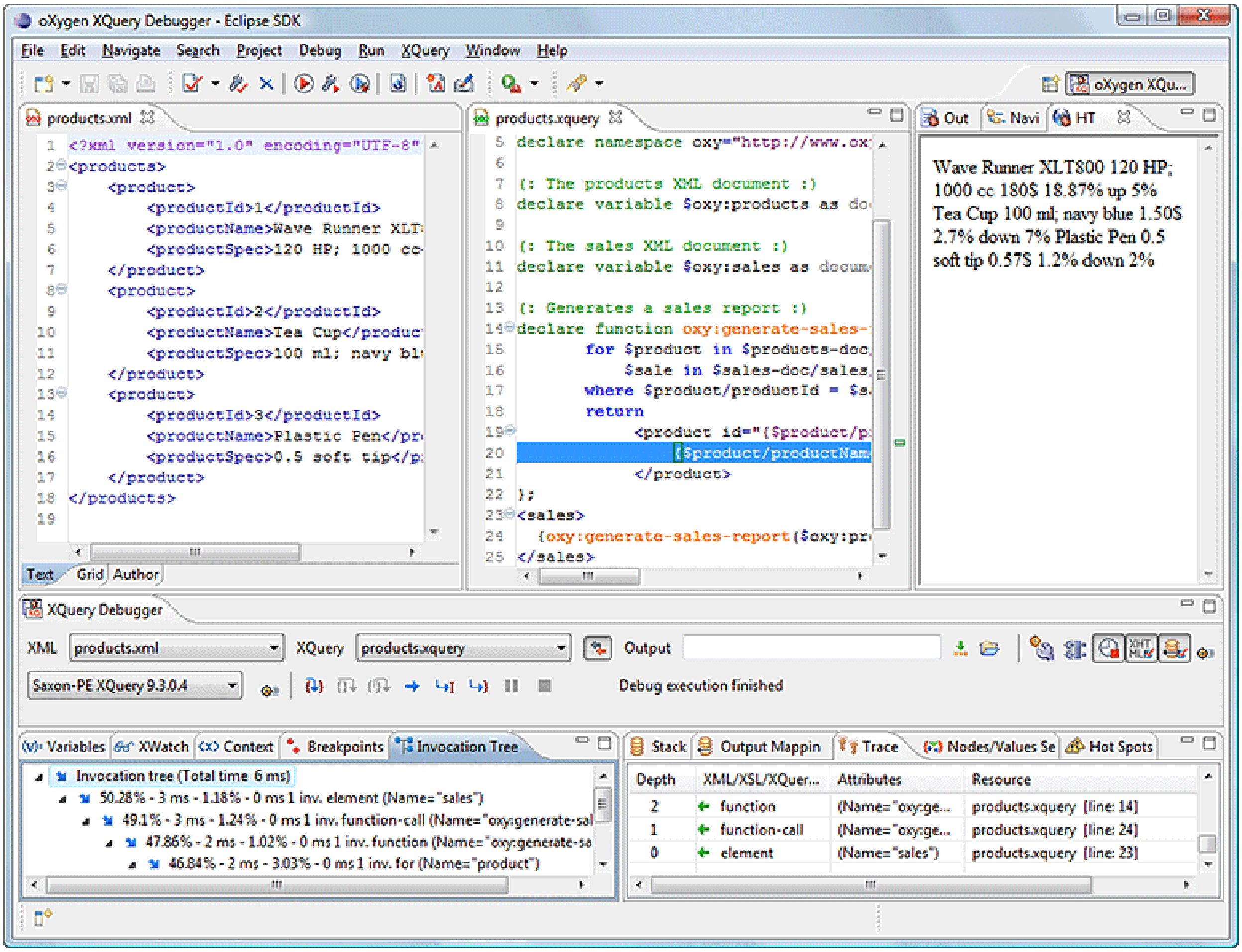
#	XSL/XQuery ...	Attributes	Resource
3	xsl:apply-...		personal.xsl [line: 27]
2	xsl:element	(name="tabl...	personal.xsl [line: 5]
1	html		personal.xsl [line: 4]
0	xsl:templ...	(match="/")	personal.xsl [line: 3]

Figure 10: Oxygen XML Editor 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:



```
<?xml version="1.0" encoding="UTF-8"
<products>
    <product>
        <productId>1</productId>
        <productName>Wave Runner XLT
        <productSpec>120 HP; 1000 cc
    </product>
    <product>
        <productId>2</productId>
        <productName>Tea Cup</produc
        <productSpec>100 ml; navy bl
    </product>
    <product>
        <productId>3</productId>
        <productName>Plastic Pen</pr
        <productSpec>0.5 soft tip</p
    </product>
</products>
```

```
declare namespace oxy="http://www.ox

(: The products XML document :)
declare variable $oxy:products as do

(: The sales XML document :)
declare variable $oxy:sales as docum

(: Generates a sales report :)
declare function oxy:generate-sales-
        for $product in $products-doc
            $sale in $sales-doc/sales
        where $product/productId = $s
        return
            <product id="{$product/p
                {$product/productNam
            </product>
};
<sales>
  {oxy:generate-sales-report($oxy:pr
</sales>
```

Wave Runner XLT800 120 HP; 1000 cc 180$ 18.87% up 5% Tea Cup 100 ml; navy blue 1.50$ 2.7% down 7% Plastic Pen 0.5 soft tip 0.57$ 1.2% down 2%

Debug execution finished

```
Invocation tree (Total time 6 ms)
  50.28% - 3 ms - 1.18% - 0 ms 1 inv. element (Name="sales")
    49.1% - 3 ms - 1.24% - 0 ms 1 inv. function-call (Name="oxy:generate-sal
      47.86% - 2 ms - 1.02% - 0 ms 1 inv. function (Name="oxy:generate-sa
        46.84% - 2 ms - 3.03% - 0 ms 1 inv. for (Name="product")
```

Depth	XML/XSL/XQuer...	Attributes	Resource
2	function	(Name="oxy:ge...	products.xquery [line: 14]
1	function-call	(Name="oxy:ge...	products.xquery [line: 24]
0	element	(Name="sales")	products.xquery [line: 23]

Figure 11: Oxygen XML Editor 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 Editor plugin Database Perspective

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

This perspective offers database specific support for:

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

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

The non-XML capabilities are:

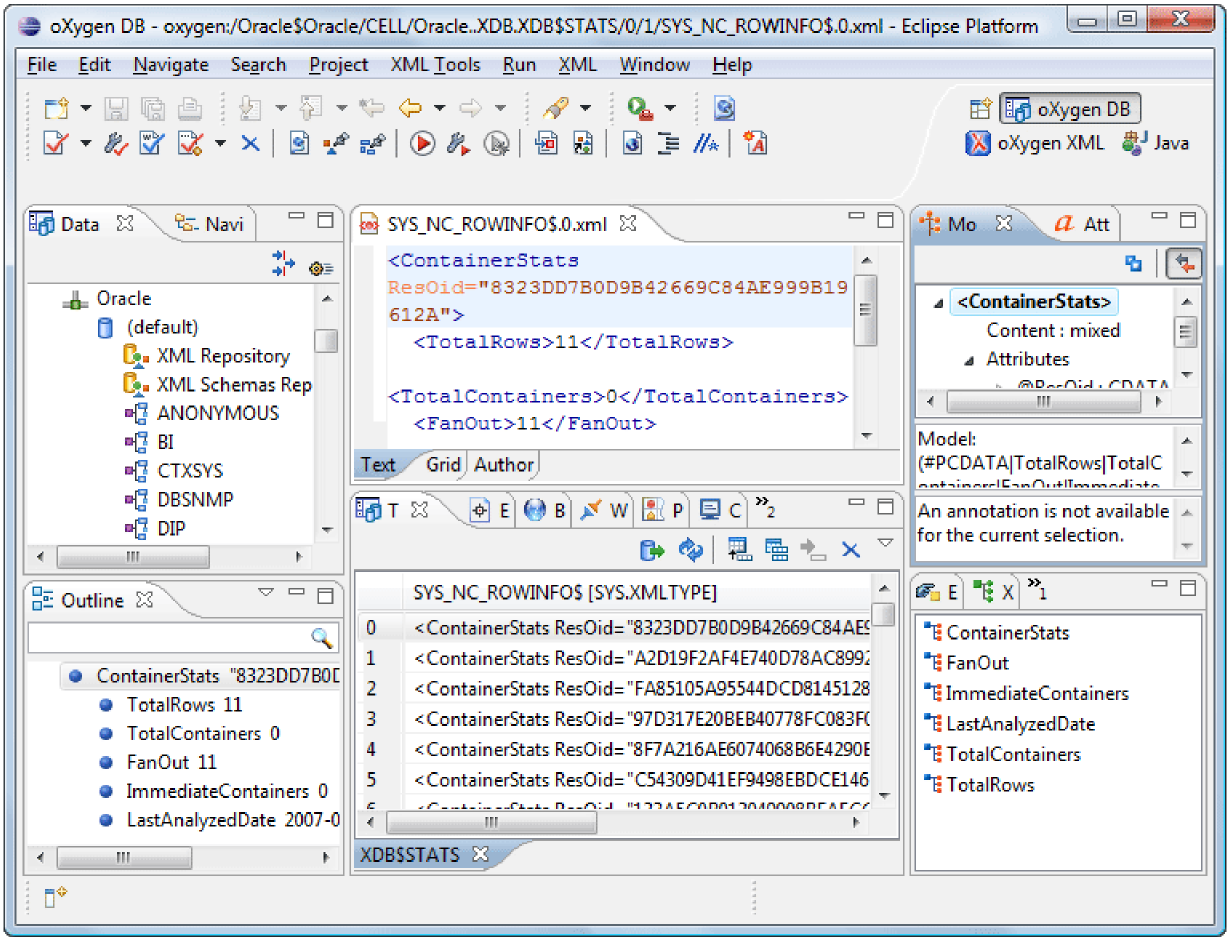
- browsing the structure of the database instance;
- opening a database table in the *Table Explorer* view;
- handling the values from **XML Type** columns as String values.

The XML capabilities are:

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

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

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



```
<ContainerStats
ResOid="8323DD7B0D9B42669C84AE999B19
612A">
  <TotalRows>11</TotalRows>

<TotalContainers>0</TotalContainers>
  <FanOut>11</FanOut>
```

```
   SYS_NC_ROWINFO$ [SYS.XMLTYPE]
0  <ContainerStats ResOid="8323DD7B0D9B42669C84AE9
1  <ContainerStats ResOid="A2D19F2AF4E740D78AC8992
2  <ContainerStats ResOid="FA85105A95544DCD8145128
3  <ContainerStats ResOid="97D317E20BEB40778FC083F(
4  <ContainerStats ResOid="8F7A216AE6074068B6E4290E
5  <ContainerStats ResOid="C54309D41EF9498EBDCE146
```

Model:
(#PCDATA|TotalRows|TotalC

An annotation is not available for the current selection.

Figure 12: Database perspective

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

Chapter 4

Editing Documents

Topics:

This chapter explains the editor types available in the Oxygen XML Editor 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 Editor plugin provides support for the Unicode standard enabling your XML application to be targeted across multiple platforms, languages, and countries without re-engineering. Internally, the Oxygen XML Editor plugin XML Editor uses 16bit characters covering the Unicode Character set.

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

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

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

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

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

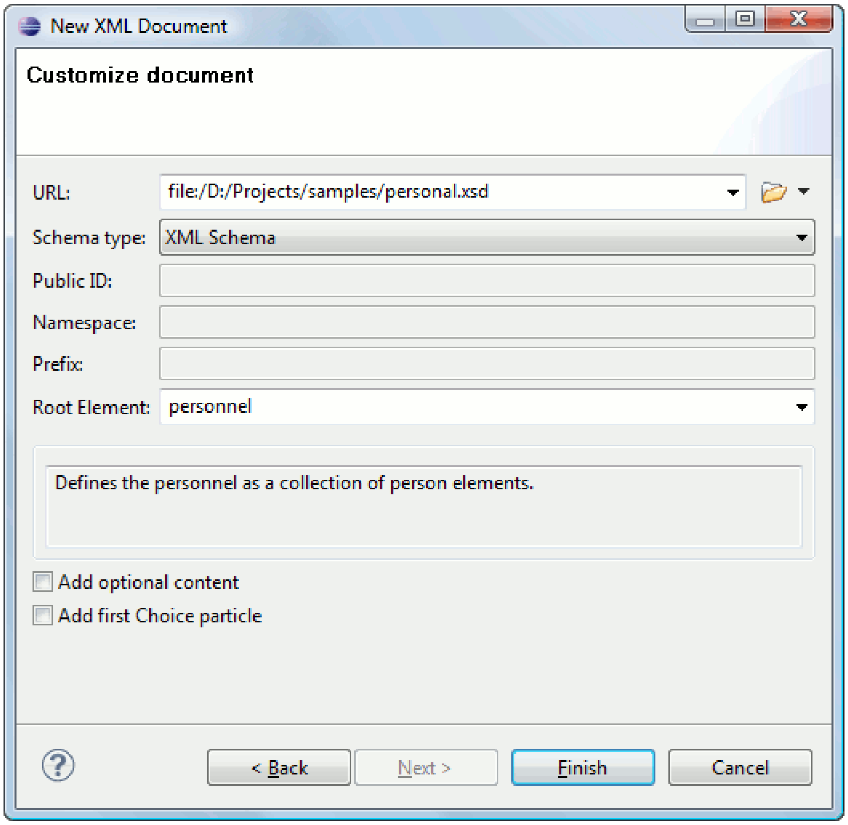
The Oxygen XML Editor plugin plugin installs a series of Eclipse wizards for easy creation of new documents. Using these wizards you let Oxygen XML complete details like the system ID or schema location of a new XML document, the minimal markup of a DocBook article or the namespace declarations of a 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:

   - 

     URL: file:/D:/Projects/samples/personal.xsd
     Schema type: XML Schema
     Public ID:
     Namespace:
     Prefix:
     Root Element: personnel

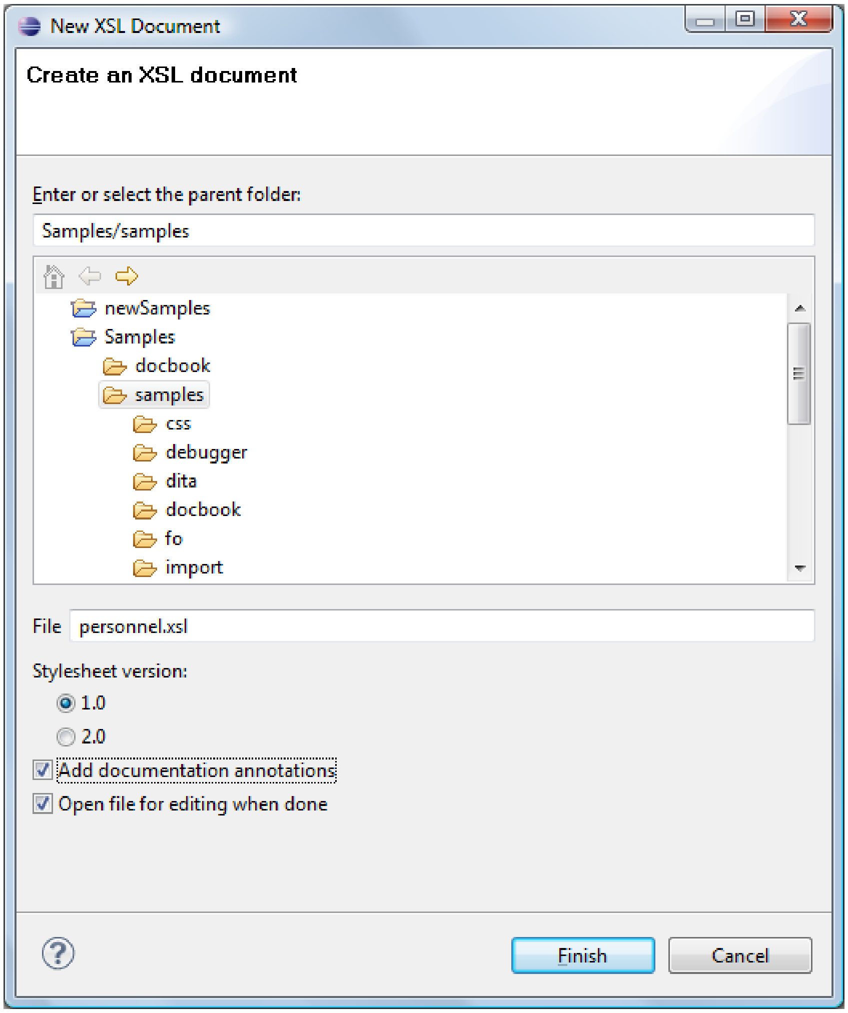
     Defines the personnel as a collection of person elements.

     □ Add optional content
     □ Add first Choice particle

     Figure 13: New XML Document Dialog

     - **Schema URL** - Path to the schema file. When a file is selected, Oxygen XML Editor 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.

- 

New XSL Document

Create an XSL document

Enter or select the parent folder:

Samples/samples

File personnel.xsl

Stylesheet version:

● 1.0

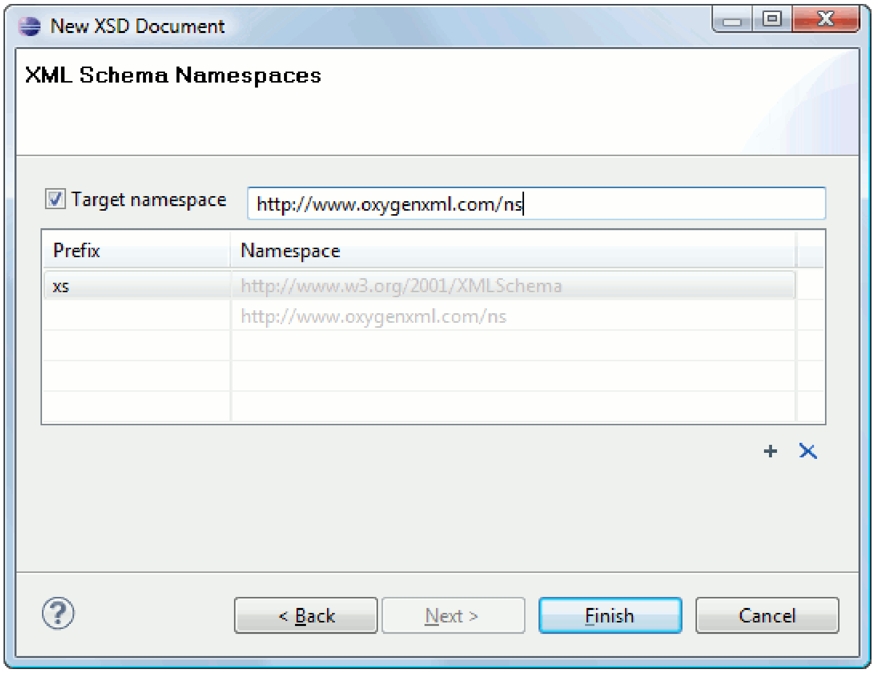
○ 2.0

☑ Add documentation annotations

☑ Open file for editing when done

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.

- 

New XSD Document

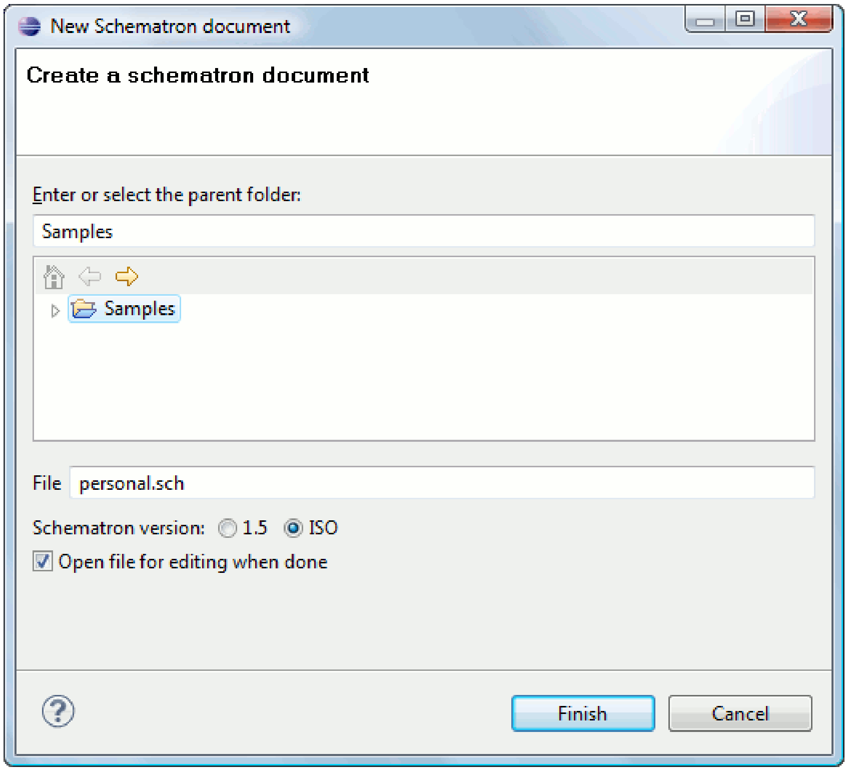
XML Schema Namespaces

☑ Target namespace: http://www.oxygenxml.com/ns

Prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema
	http://www.oxygenxml.com/ns

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.

- 

New Schematron document

Create a schematron document

Enter or select the parent folder: Samples

▷ Samples

File: personal.sch

Schematron version: ○ 1.5 ● ISO

☑ Open file for editing when done

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 Editor plugin install directory. Also in the option page **Window** > **Preferences** > **oXygen** > **Editor** > **Templates** > **Document Templates** can be specified a custom templates folder to be scanned.

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

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

Document Templates

Templates are documents 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 Editor 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 Editor 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.

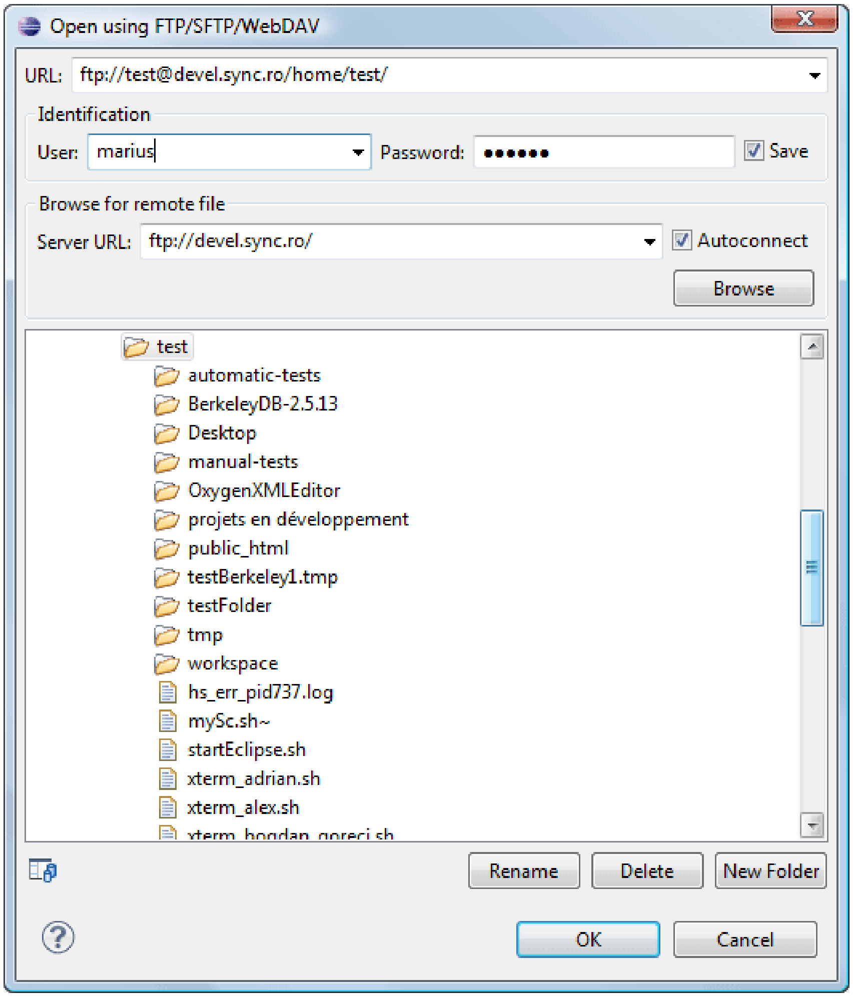
You can open one or more remote files in *the dialog **Open using FTP/SFTP***.

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

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

The Dialog Open Using FTP/SFTP

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



Open using FTP/SFTP/WebDAV

URL: ftp://test@devel.sync.ro/home/test/

Identification

User: marius Password: ●●●●●● ☑ Save

Browse for remote file

Server URL: ftp://devel.sync.ro/ ☑ Autoconnect

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 Editor plugin allows you to load and save the documents over the HTTPS protocol (if the server understands this protocol) so that any data exchange with the WebDAV server is encrypted.

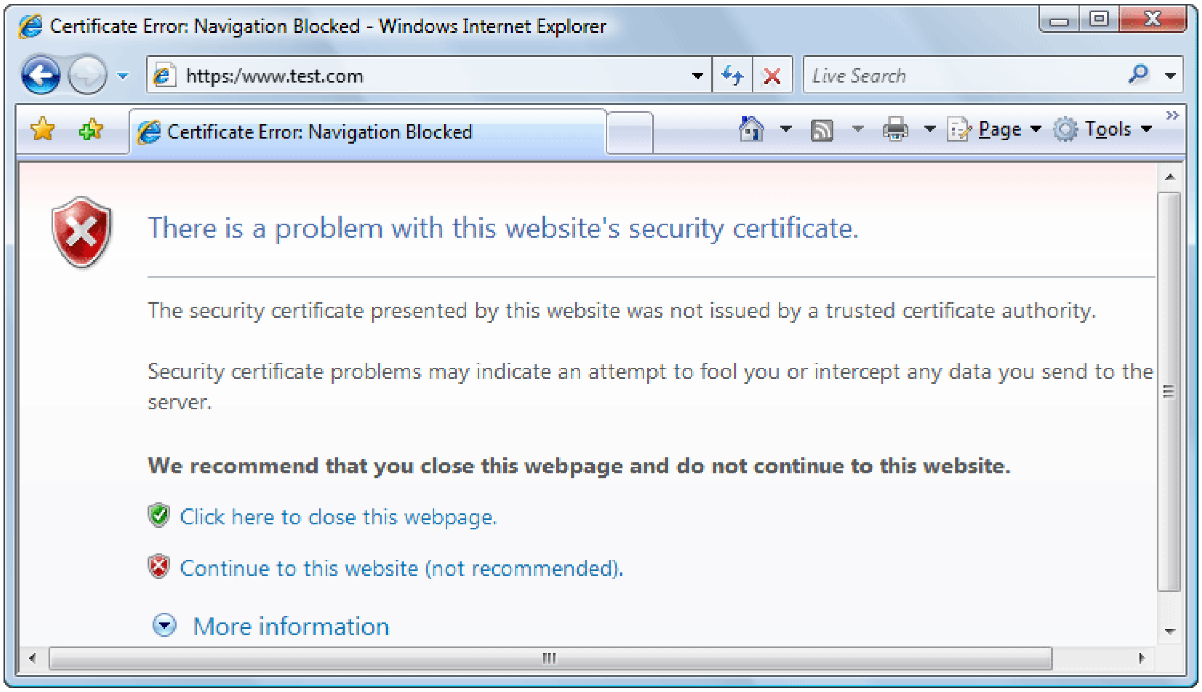
When a WebDAV repository is first accessed over HTTPS, the server hosting the repository will present a security certificate as part of the HTTPS protocol, without any user intervention. Oxygen XML Editor plugin will use this certificate to decrypt any data stream received from the server. For the authentication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

How to Add a HTTPS Server Certificate to Oxygen XML Editor 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.


Certificate Error: Navigation Blocked - Windows Internet Explorer
https://www.test.com
Live Search
Certificate Error: Navigation Blocked
Page
Tools

There is a problem with this website's security certificate.

The security certificate presented by this website was not issued by a trusted certificate authority.

Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.

We recommend that you close this webpage and do not continue to this website.

Click here to close this webpage.

Continue to this website (not recommended).

More information

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 Editor 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 Editor 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 Editor 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 Editor plugin XPR project file from the file explorer in the **Project panel**.
- **New Project** - Creates a new, empty project.

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

Creating New Project Items

A series of actions are available in the contextual menu:

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

Managing Project Content

Validate Files

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

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

Applying Transformation Scenarios

The currently selected files associated to the Oxygen XML Editor plugin 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 Editor 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.

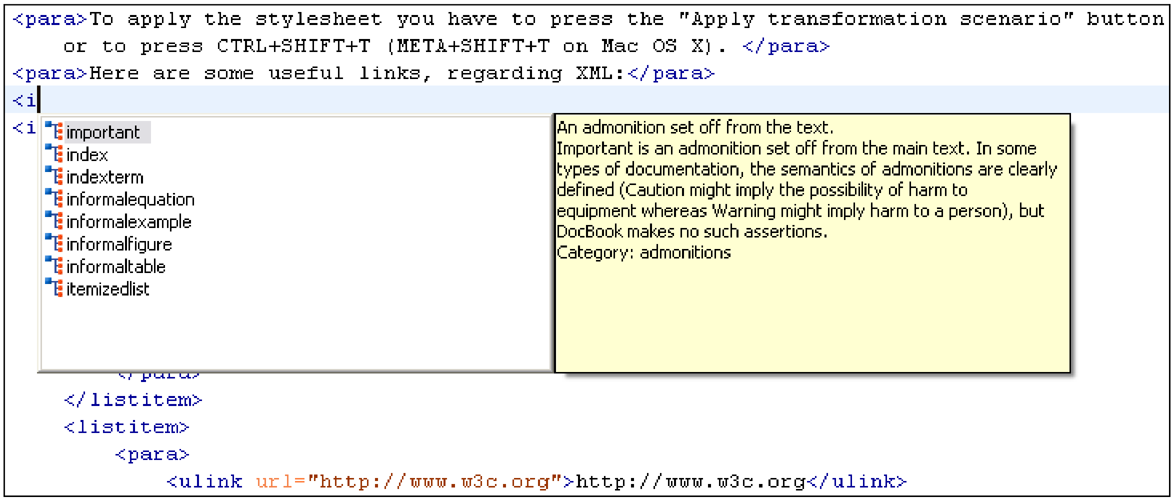
  Note: The locations are sorted by priority, from high to low.

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.



```
<para>To apply the stylesheet you have to press the "Apply transformation scenario" button
    or to press CTRL+SHIFT+T (META+SHIFT+T on Mac OS X). </para>
<para>Here are some useful links, regarding XML:</para>
<i
<i
        </para>
    </listitem>
    <listitem>
        <para>
            <ulink url="http://www.w3c.org">http://www.w3c.org</ulink>
```

An admonition set off from the text.
Important is an admonition set off from the main text. In some types of documentation, the semantics of admonitions are clearly defined (Caution might imply the possibility of harm to equipment whereas Warning might imply harm to a person), but DocBook makes no such assertions.
Category: admonitions

Figure 19: Content Completion Driven by DocBook DTD

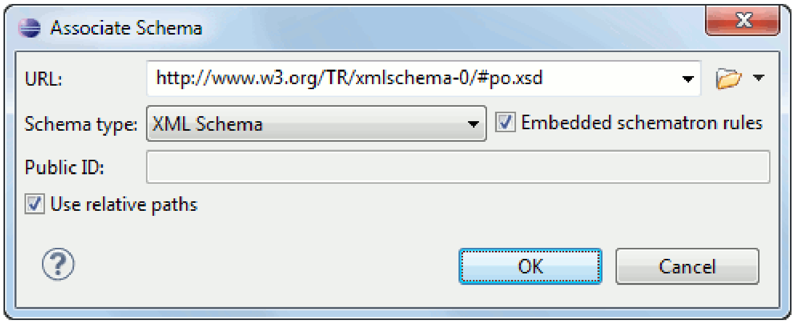
Making the Schema Association Explicit in the XML Instance Document

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

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

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

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



Associate Schema

URL: http://www.w3.org/TR/xmlschema-0/#po.xsd

Schema type: XML Schema ☑ Embedded schematron rules

Public ID:

☑ Use relative paths

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 Editor plugin is able to learn and translate the document structure to a DTD. You can choose to save the learned structure to a file in order to provide a DTD as an initialization source for *content completion* and *document validation*. This feature is also useful for producing DTD's for documents containing personal or custom element types.

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

Create a DTD from Learned Document Structure

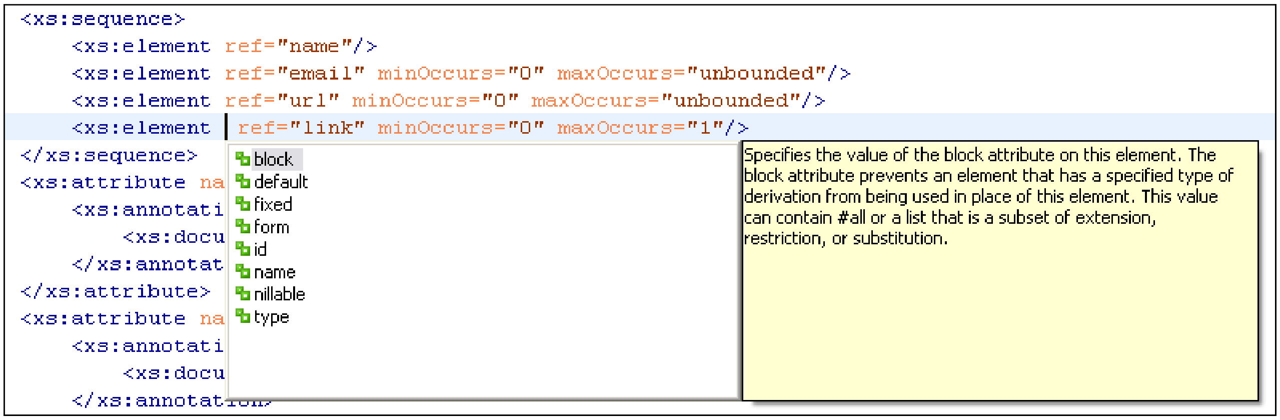
When there is no schema associated with an XML document, Oxygen XML Editor plugin can learn the document structure by parsing the document internally. This feature is enabled with *the option* ***Learn on open document*** that is available in the user preferences.

To create a DTD from the learned structure:

1. Open the XML document for which a DTD will be created.
2. Go to 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 Editor 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.



```
<xs:sequence>
    <xs:element ref="name"/>
    <xs:element ref="email" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="url" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="link" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:attribute na
    <xs:annotati
        <xs:docu
    </xs:annotat
</xs:attribute>
<xs:attribute na
    <xs:annotati
        <xs:docu
    </xs:annotation>
```

Specifies the value of the block attribute on this element. The block attribute prevents an element that has a specified type of derivation from being used in place of this element. This value can contain #all or a list that is a subset of extension, restriction, or substitution.

Figure 21: Content Completion Assistant

Oxygen XML Editor 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:

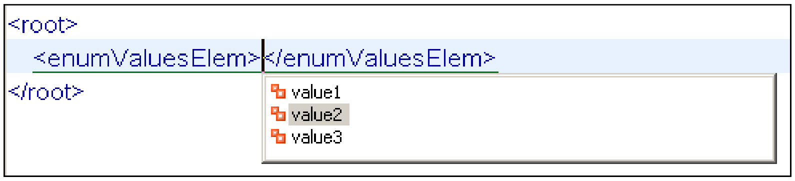
- the schema specified explicitly in the document. In this case Oxygen XML Editor plugin reads the beginning of the document and resolves the location of the DTD, XML Schema, Relax NG schema, or NVDL schema;
- the default schema rule declared in *the Document Type Association preferences panel* which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

Content Completion in Documents with Relax NG Schemas

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

```
<element name="enumValuesElem">
    <choice>
        <value>value1</value>
        <value>value2</value>
        <value>value3</value>
    </choice>
</element>
```

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



```
<root>
  <enumValuesElem></enumValuesElem>
</root>
```

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

Schema Annotations

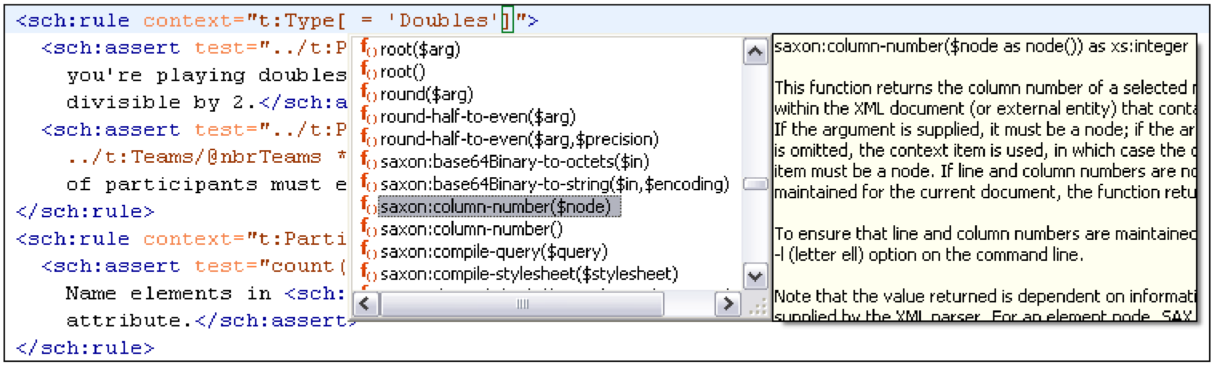
If the document's schema is an XML Schema, Relax NG (full syntax), NVDL or DTD and it contains element, attributes or attributes values annotations, these will be presented when the content completion window is displayed, only if the option *Show annotations* is enabled. Also the annotation is presented 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 Editor plugin seeks an annotation in the type definition of the element / attribute or, if no annotation is found there, in the parent type definition of that definition, etc.

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



```
<sch:rule context="t:Type[ = 'Doubles']">
  <sch:assert test="../t:P
    you're playing doubles
    divisible by 2.</sch:a
  <sch:assert test="../t:P
    ../t:Teams/@nbrTeams *
    of participants must e
</sch:rule>
<sch:rule context="t:Parti
  <sch:assert test="count(
    Name elements in <sch:
    attribute.</sch:assert>
</sch:rule>
```

root($arg)
root()
round($arg)
round-half-to-even($arg)
round-half-to-even($arg,$precision)
saxon:base64Binary-to-octets($in)
saxon:base64Binary-to-string($in,$encoding)
saxon:column-number($node)
saxon:column-number()
saxon:compile-query($query)
saxon:compile-stylesheet($stylesheet)

saxon:column-number($node as node()) as xs:integer

This function returns the column number of a selected r
within the XML document (or external entity) that conta
If the argument is supplied, it must be a node; if the ar
is omitted, the context item is used, in which case the c
item must be a node. If line and column numbers are no
maintained for the current document, the function retu

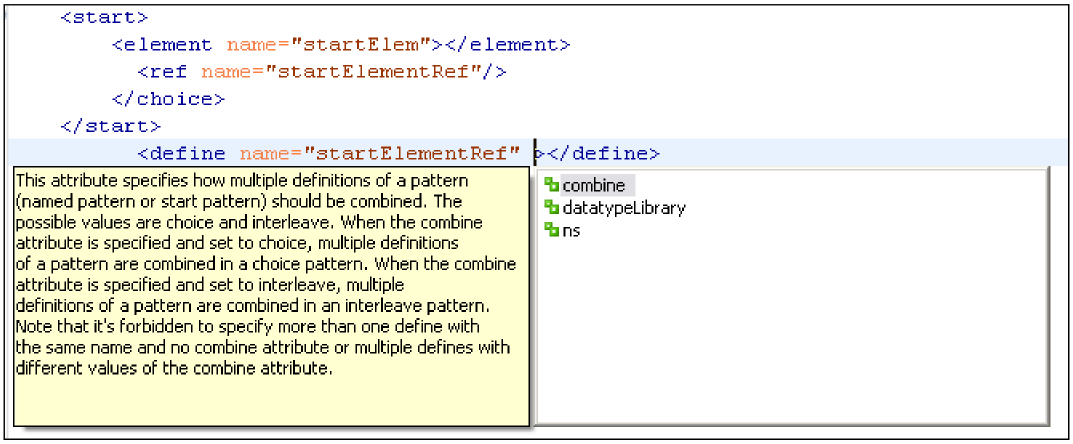
To ensure that line and column numbers are maintained
-l (letter ell) option on the command line.

Note that the value returned is dependent on informati
supplied by the XML parser. For an element node, SAX

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



```
    <start>
        <element name="startElem"></element>
          <ref name="startElementRef"/>
        </choice>
    </start>
          <define name="startElementRef" ></define>
```

This attribute specifies how multiple definitions of a pattern
(named pattern or start pattern) should be combined. The
possible values are choice and interleave. When the combine
attribute is specified and set to choice, multiple definitions
of a pattern are combined in a choice pattern. When the combine
attribute is specified and set to interleave, multiple
definitions of a pattern are combined in an interleave pattern.
Note that it's forbidden to specify more than one define with
the same name and no combine attribute or multiple defines with
different values of the combine attribute.

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

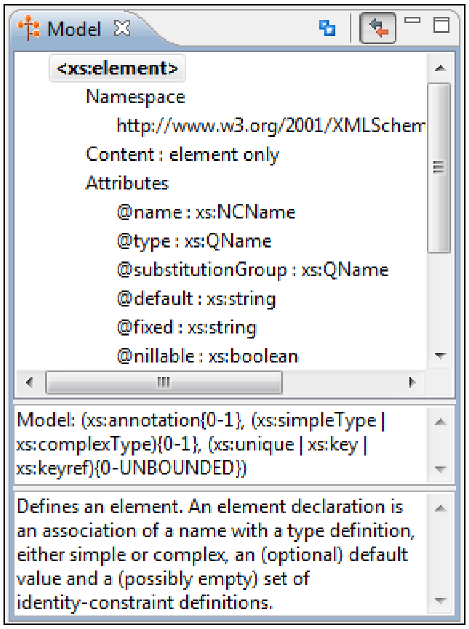
```
<!--doc:Description of the element. -->
```

Content Completion Helper Views

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

The Model View

The Model view presents the structure of the 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**



```
<xs:element>
    Namespace
        http://www.w3.org/2001/XMLSchem
    Content : element only
    Attributes
        @name : xs:NCName
        @type : xs:QName
        @substitutionGroup : xs:QName
        @default : xs:string
        @fixed : xs:string
        @nillable : xs:boolean
```

Model: (xs:annotation{0-1}, (xs:simpleType | xs:complexType){0-1}, (xs:unique | xs:key | xs:keyref){0-UNBOUNDED})

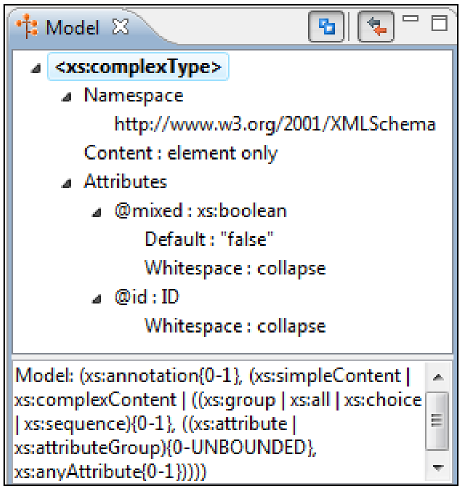
Defines an element. An element declaration is an association of a name with a type definition, either simple or complex, an (optional) default value and a (possibly empty) set of identity-constraint definitions.

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.



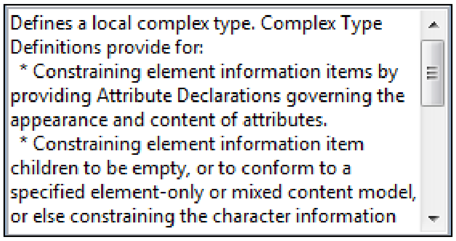
```
<xs:complexType>
    Namespace
        http://www.w3.org/2001/XMLSchema
    Content : element only
    Attributes
        @mixed : xs:boolean
            Default : "false"
            Whitespace : collapse
        @id : ID
            Whitespace : collapse
```

Model: (xs:annotation{0-1}, (xs:simpleContent | xs:complexContent | ((xs:group | xs:all | xs:choice | xs:sequence){0-1}, ((xs:attribute | xs:attributeGroup){0-UNBOUNDED}, xs:anyAttribute{0-1}))))

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.



Defines a local complex type. Complex Type Definitions provide for:
 * 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

Figure 27: The Annotation panel

The Attributes View

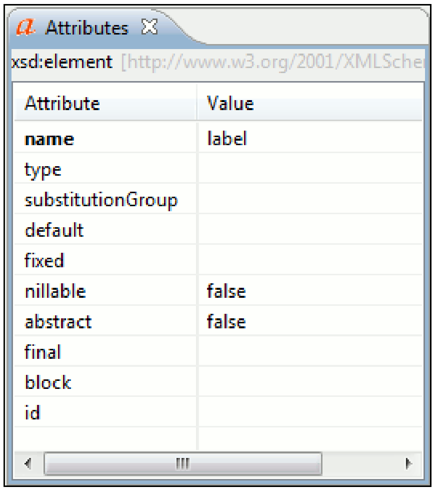
The **Attributes View** presents all possible attributes of the current element.

The view allows you to insert attributes or change the value of the already used attributes for the current editable element. An element is editable if either one of the following is true:

- the CSS stylesheet associated with the document does not specify a **false** value for the *-oxy-editable* property associated with the element.
- the element is entirely included into a deleted *Track Changes* marker.
- the element is part of a content fragment that is referenced in **Author** mode from another document.

The attributes present in the document are 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.



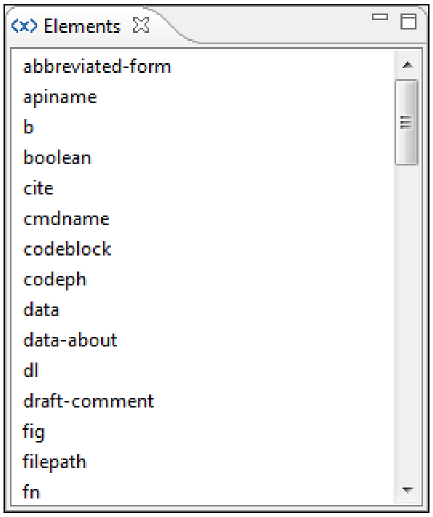
xsd:element [http://www.w3.org/2001/XMLSche

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.

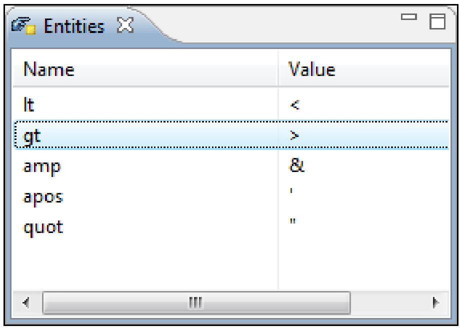


abbreviated-form
apiname
b
boolean
cite
cmdname
codeblock
codeph
data
data-about
dl
draft-comment
fig
filepath
fn

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.



Name	Value
lt	<
gt	>
amp	&
apos	'
quot	"

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 Editor 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*:

- **${caret}** - The position where the caret is inserted. This variable can be used in a *code template* , in Author operations, or in a selection plugin.
- **${selection}** - The XML content of the current selection in the editor panel. This variable can be used in a *code template* and Author operations,

- **${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 Editor plugin checks that the URL is valid before passing it to the transformation.
    - `password` - input characters are hidden.
    - `generic` - the input is treated as generic text that requires no special handling.
  - `'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 Editor plugin installation folder.
- **${pd}** - Current project folder as file path.
- **${oxygenInstallDir}** - Oxygen XML Editor plugin installation folder as file path.
- **${homeDir}** - The path (as file path) of the user home folder.
- **${pn}** - Current project name.
- **${env(VAR_NAME)}** - Value of the *VAR_NAME* environment variable.
- **${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 Editor plugin provides functions that enable easy error identification and rapid error location.

Checking XML Well-Formedness

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

The XML Syntax rules for Well-Formed XML are:

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

The namespace-wellformed rules are:

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

The namespace-valid rules are:

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

If you select menu **Document** > **Validate** > **Check Well-Formedness** **(Alt+Shift+V W (Cmd+Alt+V W on Mac))** or click the toolbar button ✓ **Check Well-Formedness** Oxygen XML Editor 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 Editor 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”.

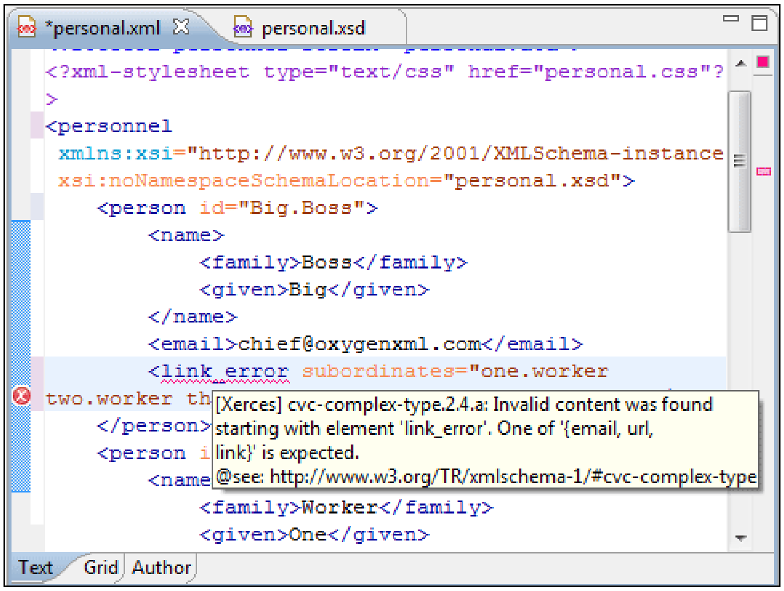
Luckily most standards based DTD's, XML Schema's and Relax NG schemas are supplied with reference documentation. This enables us to lookup the element and read about it. In this case you should learn about the child elements of `listitem` and their nesting rules. Once you have correctly inserted the 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 Editor plugin *can be configured* to mark validation errors in the document as you are editing. If you *enable the **Automatic validation** option* any validation errors and warnings will be *highlighted automatically in the editor panel*. The automatic validation starts parsing the document and marking the errors after a *configurable delay* from the last key typed. Errors are highlighted with underline markers in the main editor panel and small rectangles on the right side ruler of the editor panel, *in the same way as for manual validation invoked by the user.*



```
<?xml-stylesheet type="text/css" href="personal.css"?
>
<personnel
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
 xsi:noNamespaceSchemaLocation="personal.xsd">
    <person id="Big.Boss">
        <name>
            <family>Boss</family>
            <given>Big</given>
        </name>
        <email>chief@oxygenxml.com</email>
        <link_error subordinates="one.worker
two.worker th
    </person>
    <person i
        <name
            <family>Worker</family>
            <given>One</given>
```

[Xerces] cvc-complex-type.2.4.a: Invalid content was found starting with element 'link_error'. One of '{email, url, link}' is expected.
@see: http://www.w3.org/TR/xmlschema-1/#cvc-complex-type

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 Editor plugin user preferences. After such a custom validator is *properly configured* it can be applied on the current document with just one click on the **Custom Validation Engines** toolbar. The document is validated against the schema declared in the document.

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

- **LIBXML** - Included in Oxygen XML Editor plugin (Windows edition only). It is associated to XML Editor. It is able to validate the edited document against XML Schema, Relax NG schema full syntax, internal DTD (included in the XML document) or a custom schema type. XML catalogs support (the --catalogs parameter) and XInclude processing (--xinclude) are enabled by default in the preconfigured LIBXML validator. The --postvalid parameter is also set by default which allows LIBXML to validate correctly the main document even if the XInclude fragments contain IDREFS to ID's located in other fragments.

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

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

  Attention:

  On Mac OS X if the full path to the LIBXML executable file is not specified in the **Executable path** text field, some errors may occur 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 Editor plugin . It is associated to XML Editor and XSD Editor. It is able to validate XML Schema schemas and XML documents against XML Schema schemas. The validation is done according to the W3C XML Schema 1.0 specification or according to the W3C XML Schema 1.1 specification. This can be *configured in Preferences*.

- **MSXML 4.0** - Included in Oxygen XML Editor plugin (Windows edition only). It is associated to XML Editor, XSD Editor and XSL Editor. It is able to validate the edited document against XML Schema, internal DTD (included in the XML document), external DTD or a custom schema type.
- **MSXML.NET** - Included in Oxygen XML Editor plugin (Windows edition only). It is associated to XML Editor, XSD Editor and XSL Editor. It is able to validate the edited document against XML Schema, internal DTD (included in the XML document), external DTD or a custom schema type.
- **XSV** - Not included in Oxygen XML Editor plugin . Windows and Linux distributions of XSV can be downloaded from *http://www.cogsci.ed.ac.uk/~ht/xsv-status.html*. The executable path is *already configured in Oxygen XML Editor plugin* for the `[Oxygen-install-folder]/xsv` installation folder. If it is installed in a different folder the predefined executable path must be *corrected in Preferences.* It is associated to XML Editor and XSD Editor. It is able to validate the edited document against XML Schema or a custom schema type.
- **SQC (Schema Quality Checker from IBM)** - Not included in Oxygen XML Editor plugin . It can be downloaded *from here* (it comes as a .zip file, at the time of this writing SQC2.2.1.zip is about 3 megabytes). The executable path and working directory are already configured for the SQC installation directory `[Oxygen-install-folder]/sqc`. If it is installed in a different folder the predefined executable path and working directory must be *corrected in the Preferences page.* It is associated to XSD Editor.

Linked Output Messages of an External Engine

Validation engines display messages in an output view at the bottom of the Oxygen XML Editor plugin window. If such an output message (warning, error, fatal error, etc) spans between three to 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.

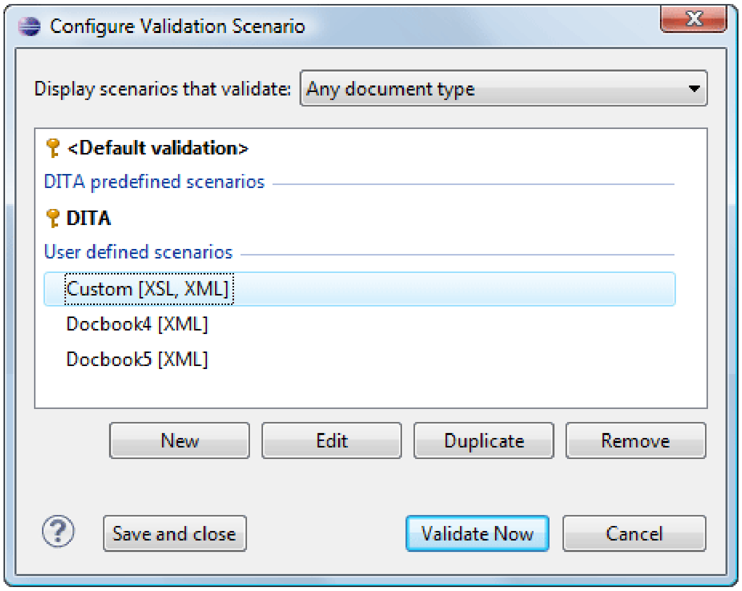
Other examples of documents which can benefit of a validation scenario are:

- 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 Editor 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 Editor plugin plugin.
   The following dialog is displayed. It contains the following types of scenarios:
   - **Default validation** scenario validates the input using Oxygen XML Editor 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 Editor plugin creates a customizable duplicate;
   - **User defined** scenarios are organized under a single category, but you can use the drop-down option box to filter them by the type of file they validate.

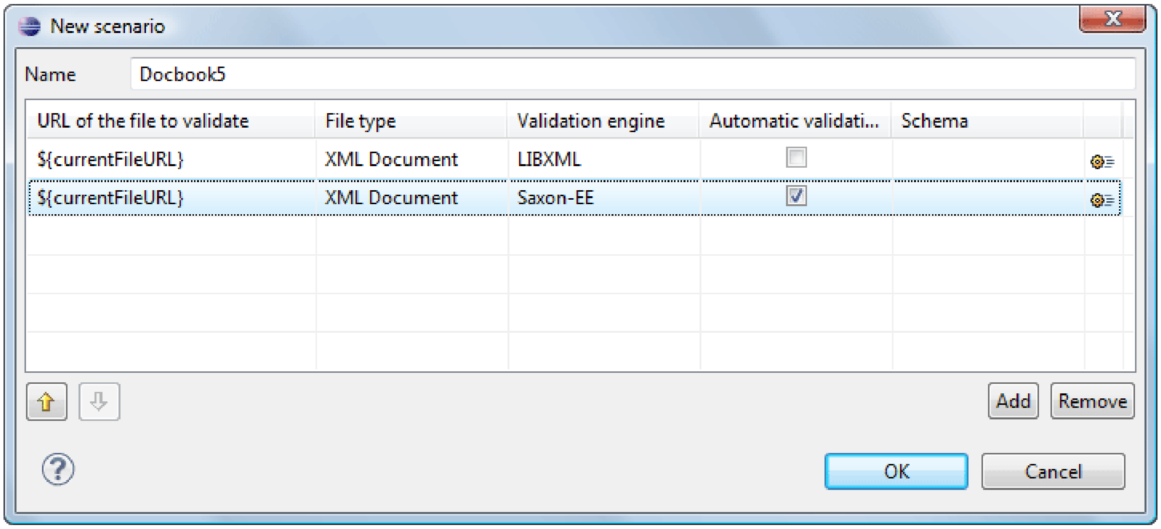


Display scenarios that validate: Any document type

<Default validation>
DITA predefined scenarios
DITA
User defined scenarios
Custom [XSL, XML]
Docbook4 [XML]
Docbook5 [XML]

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.



Name: Docbook5

URL of the file to validate	File type	Validation engine	Automatic validati...	Schema
${currentFileURL}	XML Document	LIBXML	☐	
${currentFileURL}	XML Document	Saxon-EE	☑	

Figure 33: Add / Edit a Validation Unit

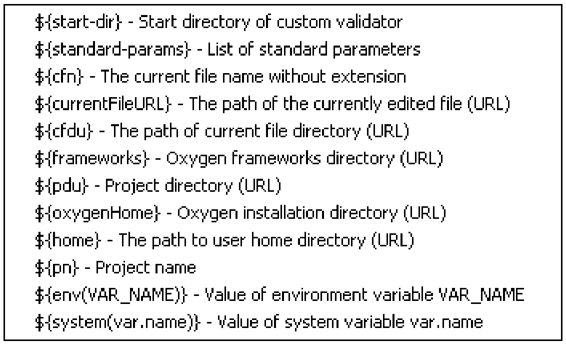
The table holds the following information:

- **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 Editor plugin automatically selects the file type depending on the value of the **URL of the file to validate** field.
- **Validation engine** - One of the engines available in Oxygen XML Editor plugin for validation of the type of document to which the current module belongs. **Default engine** is the default setting and means that the validation is done by the default engine set in Preferences pages for the type of the current document (XML document, XML Schema, XSLT stylesheet, XQuery file, etc) instead of a validation scenario.
- **Automatic validation** - If this option is checked, then the validation operation defined by this row of the table is applied also by *the automatic validation feature.* If the **Automatic validation** feature is *disabled in Preferences* then this option does not take effect as the Preference setting has higher priority.
- **Schema** - Active when you set the **File type** to **XML Document**.
- **Settings** - Contains an action that allows you to set a schema, when validating XML documents, or a list of extensions when validating XSL or XQuery documents.

4. Edit the URL of the main validation module.

Specify the URL of the main module:

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



${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 *automatic validation feature is turned on in Preferences.*
8. Choose what schema is used during validation: the one detected after parsing the document or a custom one.

Validation Actions in the User Interface

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:

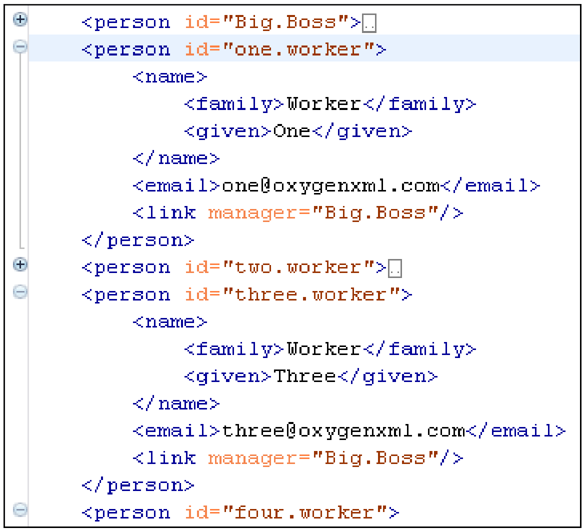
```
<system systemId="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1"
    uri="topic.xsd"/>
```

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:

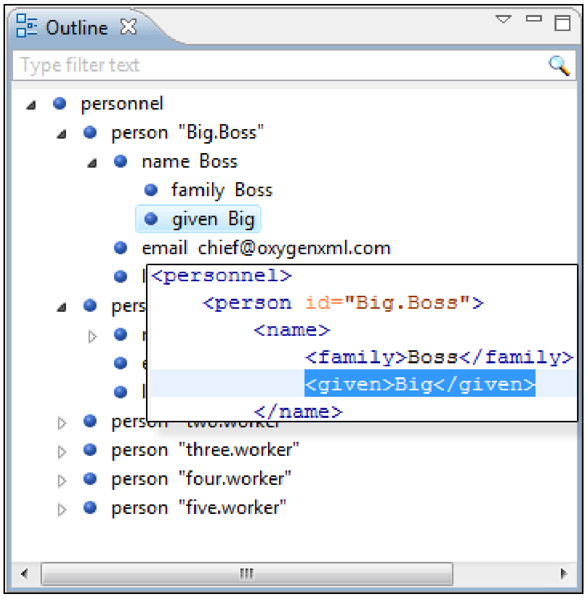
- **(Ctrl+NumPad+/)** > **Document** > **Folding** > **Close Other Folds** **(Ctrl+NumPad+/)** - Folds all the elements except the current element.
- **Document** > **Folding** > **Collapse Child Folds** **(Ctrl+Decimal)** **(Ctrl+NumPad+-)** **((Cmd+NumPad+- on Mac OS))** - Folds the elements indented 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 72
- *Outline Specific Actions* on page 72
- *Modification Follow-up* on page 73
- *Document Structure Change* on page 73
- *Document Tag Selection* on page 73



```
<personnel>
    <person id="Big.Boss">
        <name>
            <family>Boss</family>
            <given>Big</given>
        </name>
```

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

- **Show comments and processing instructions** - show/hide comments and processing instructions in the **Outline** view.
- **Show text** - show/hide additional text content for the displayed elements.
- **Show attributes** - show/hide attribute values for the displayed elements. The displayed attribute values can be changed from *the Outline preferences panel*.
- **Configure displayed attributes** - displays the *XML Structured Outline preferences page*.

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

Modification Follow-up

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

Document Structure Change

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

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

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

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

The Popup Menu of the Outline Tree

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

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

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

The *Cut*, *Copy* and *Delete* items of the popup menu execute the same actions as the Edit menu items with the same name on the elements currently selected in the outline tree (Cut, Copy, Paste).

Document Tag Selection

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

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

Large Documents

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

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

Including Document Parts with DTD Entities

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

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

A master document looks like this:

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

The referred document looks like this:

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

Note:

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

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

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

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

Including Document Parts with XInclude

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

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

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

Include a chapter in an article using XInclude

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

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

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

Main article file looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.docbook.org/xml/4.3/docbookx.dtd"
[ <!ENTITY % xinclude SYSTEM
"../frameworks/docbook/dtd/xinclude.mod">
%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 Editor plugin is turned on by default. You can *toggle it* by going to the entry **Enable XInclude processing** in the menu **Window** > **Preferences ...** > **oXygen** > **XML** > **XML Parser**. When enabled, Oxygen XML Editor plugin will be able to validate and transform documents comprised of parts added using XInclude.

Working with XML Catalogs

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 Editor 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 Editor 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 Editor 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 Editor plugin starting from the two XML Catalog document templates called *OASIS XML Catalog 1.0* and *OASIS XML Catalog 1.1* and available in *the document templates dialog*.

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 Editor 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.

👉 **Important:**

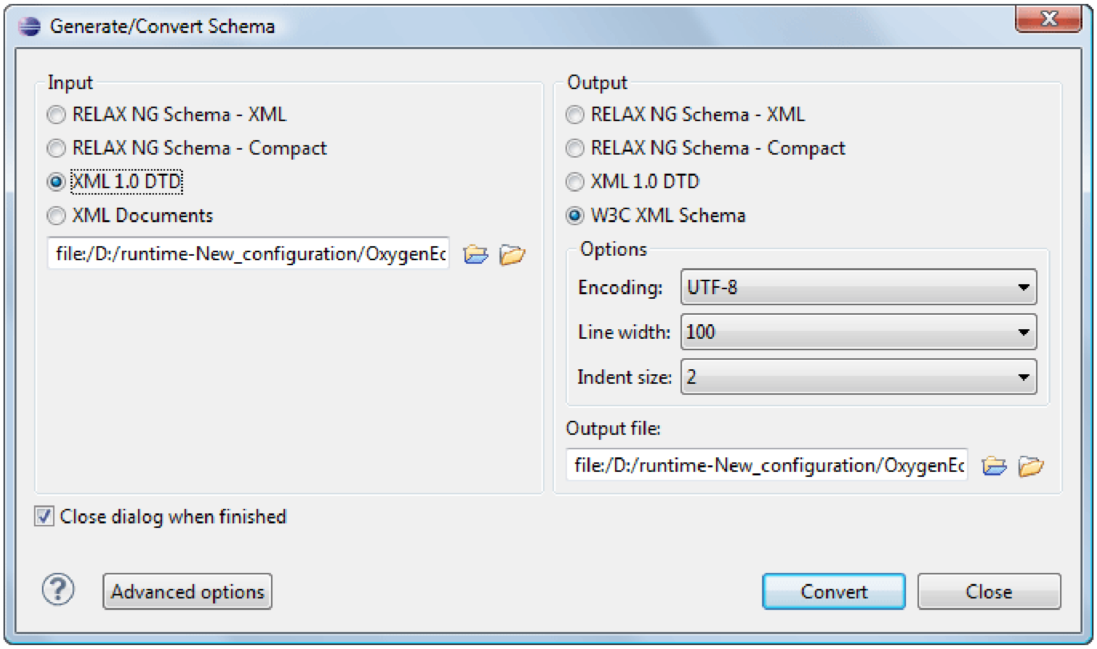
The catalog files settings are available for all editing modes, not only for the **Author** mode.

Converting Between Schema Languages

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



Generate/Convert Schema

Input
○ RELAX NG Schema - XML
○ RELAX NG Schema - Compact
● XML 1.0 DTD
○ XML Documents
file:/D:/runtime-New_configuration/OxygenEc

Output
○ RELAX NG Schema - XML
○ RELAX NG Schema - Compact
○ XML 1.0 DTD
● W3C XML Schema

Options
Encoding: UTF-8
Line width: 100
Indent size: 2

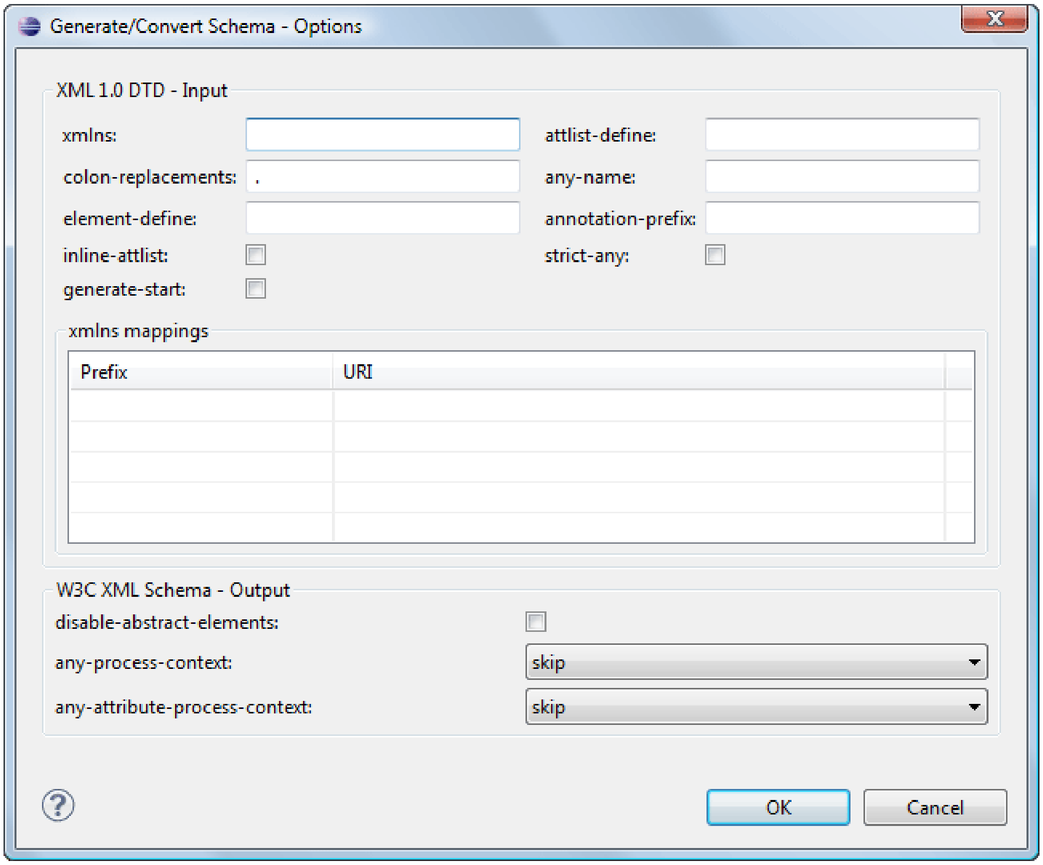
Output file: file:/D:/runtime-New_configuration/OxygenEc

☑ Close dialog when finished

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:



Generate/Convert Schema - Options

XML 1.0 DTD - Input
xmlns:
colon-replacements: .
element-define:
inline-attlist: □
generate-start: □
attlist-define:
any-name:
annotation-prefix:
strict-any: □

xmlns mappings

Prefix	URI

W3C XML Schema - Output
disable-abstract-elements: □
any-process-context: skip
any-attribute-process-context: skip

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 anyAttribute elements. The default is skip (corresponding to RELAX NG semantics).

Formatting and Indenting Documents (Pretty Print)

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

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

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

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

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

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

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

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

//xs:documentation the XPath descendant axis can be used only at the beginning of the expression; the namespace prefix can be attached to any namespace, no namespace binding check is performed when applying the pretty-print operation

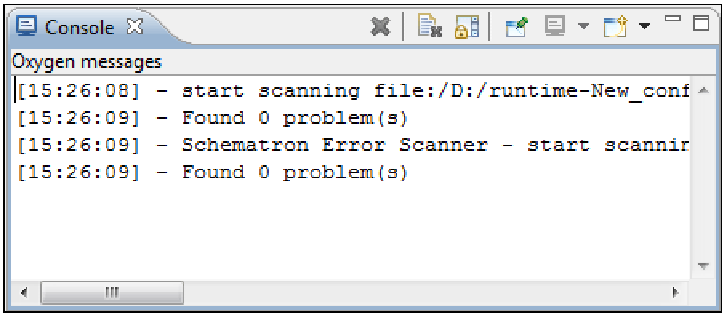
/chapter/abstract/title note the use of the XPath child axis

//section/title the descendant axis can be followed by the child axis

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

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.



```
[15:26:08] - start scanning file:/D:/runtime-New_conf
[15:26:09] - Found 0 problem(s)
[15:26:09] - Schematron Error Scanner - start scannir
[15:26:09] - Found 0 problem(s)
```

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

Edit Actions

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

- **contextual menu of current editor** > **Toggle comment** **(Ctrl + /)** - 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 Editor 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 Editor plugin will introduce a new declaration `xmlns:p5="ns1"` in the current element and will change the prefix from `p1` to `p5`. If `p5` is already associated in the current element with another namespace, let's say `ns5`, then a dialog showing the conflict will be displayed. Pressing the **OK** button, the prefix will be modified from `p1` to `p5` without inserting a new declaration `xmlns:p5="ns1"`. On **Cancel** no modification is made.

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

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

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

Smart Editing

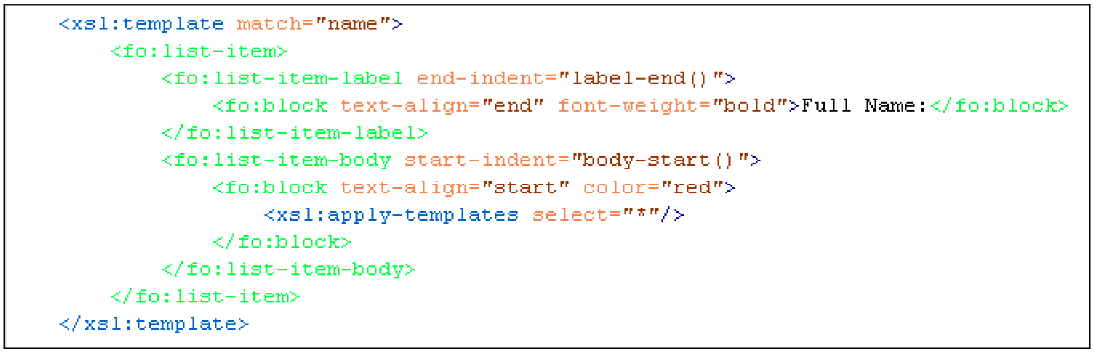
The following helper actions are available in the XML editor:

- *Closing tag auto-expansion* - If you want to insert content into an auto closing tag like <tag/> deleting the / character saves some keystrokes by inserting a separate closing tag automatically and placing the cursor between the start and end tags: <tag></tag>
- *Auto-rename matching tag* - When you edit the name of the start tag, Oxygen XML Editor plugin will mirror-edit the name of the matching end tag. This feature can be controlled from the *Content Completion option page*.
- *Auto-breaking the edited line* - The *Hard line wrap option* breaks the edited line automatically when its length exceeds the maximum line length *defined* for *the pretty-print operation.*
- *Indent on Enter* - The *Indent on Enter option* indents the new line inserted when Enter is pressed.
- *Smart Enter* - The *Smart Enter option* inserts an empty line between the start and end tags. If Enter is pressed between a start and an end tag the action places the cursor in an indented position on the empty line between the lines that contain the start and end tag.
- *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 Editor 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

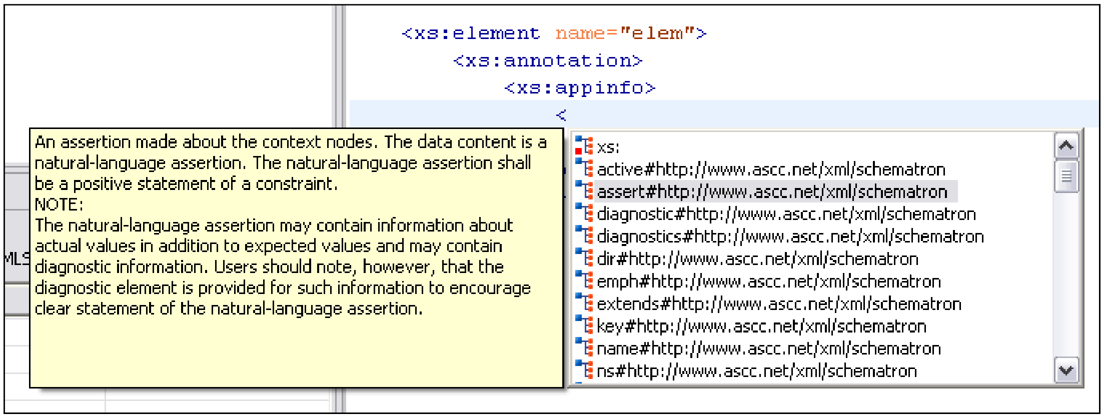
This page is used to edit the XML Schema in a text mode. It offers powerfull content completion support, a synchronized Outline view and multiple *refactory actions*. The outline view has two display modes: the mode and the *components* mode.

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.



```
<xs:element name="elem">
    <xs:annotation>
        <xs:appinfo>
            <
```

An assertion made about the context nodes. The data content is a natural-language assertion. The natural-language assertion shall be a positive statement of a constraint.
NOTE:
The natural-language assertion may contain information about actual values in addition to expected values and may contain diagnostic information. Users should note, however, that the diagnostic element is provided for such information to encourage clear statement of the natural-language assertion.

```
xs:
active#http://www.ascc.net/xml/schematron
assert#http://www.ascc.net/xml/schematron
diagnostic#http://www.ascc.net/xml/schematron
diagnostics#http://www.ascc.net/xml/schematron
dir#http://www.ascc.net/xml/schematron
emph#http://www.ascc.net/xml/schematron
extends#http://www.ascc.net/xml/schematron
key#http://www.ascc.net/xml/schematron
name#http://www.ascc.net/xml/schematron
ns#http://www.ascc.net/xml/schematron
```

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

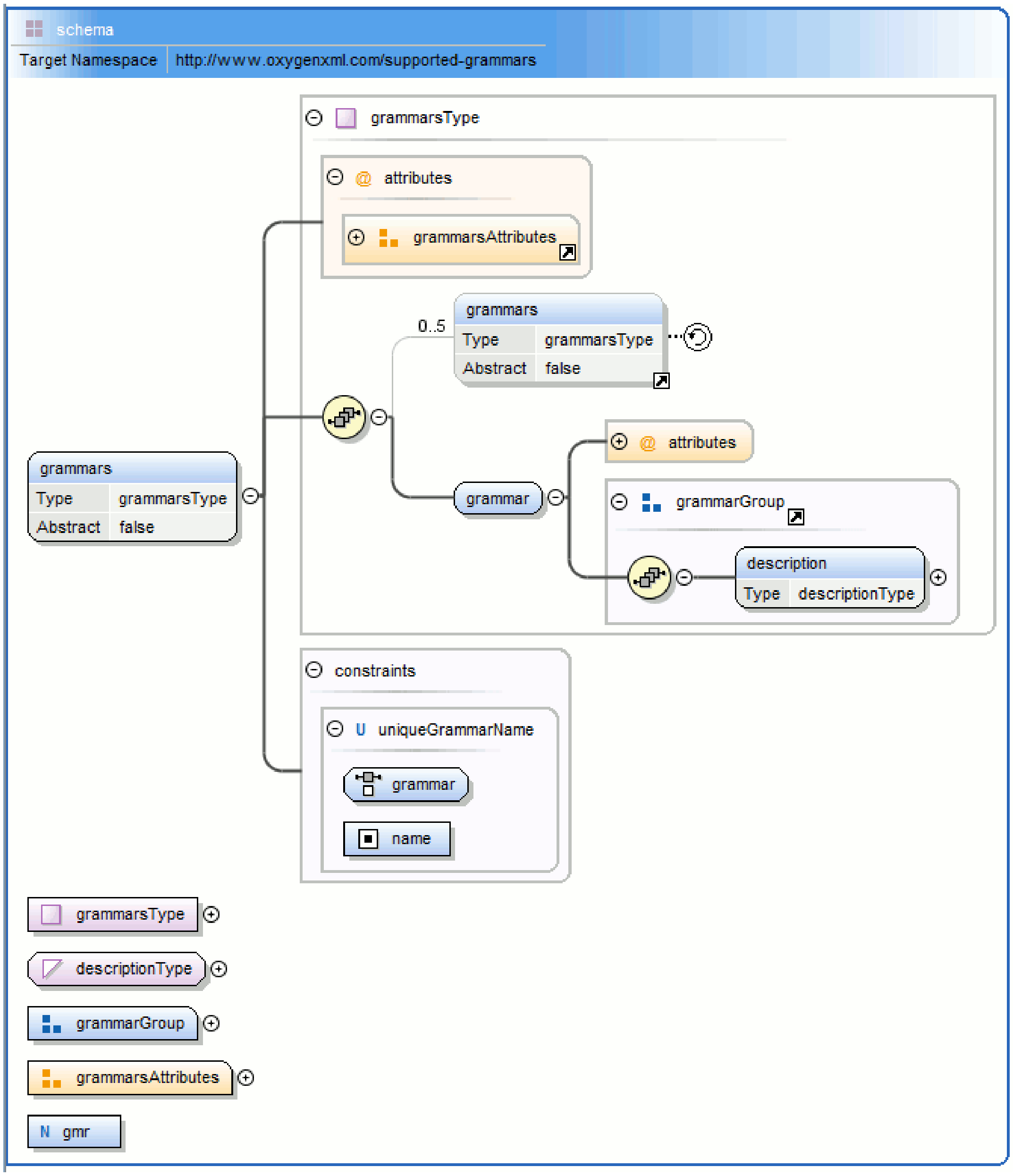
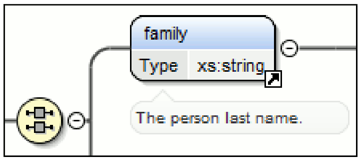


Figure 42: XML Schema Diagram

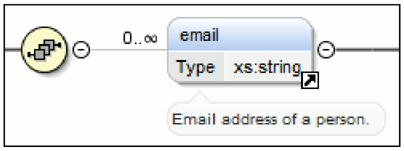
XML Schema Components

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);

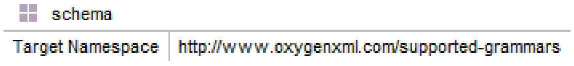


- a thin line to identify a connection with an optional component (in the following image, `email` is an optional element).



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

`xs:schema`



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

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

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

xs:schema properties

Property Name	Description	Possible Values
Target Namespace	The schema target namespace.	Any URI
Element Form Default	Determining whether local element declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Attribute Form Default	Determining whether local attribute declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Block Default	Default value of the `block` attribute of `xs:element` and `xs:complexType`.	#all, extension, restriction, substitution, restriction extension, restriction substitution, extension substitution, restriction extension substitution, [Empty]
Final Default	Default value of the `final` attribute of `xs:element` and `xs:complexType`.	#all, restriction, extension, restriction extension, [Empty]
Version	Schema version	Any token
ID	The schema id	Any ID

Property Name	Description	Possible Values
Component	The edited component name.	Not editable property.
SystemID	The schema system id	Not editable property.

xs:element

email
Type
xs:string
Fixed
true

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.

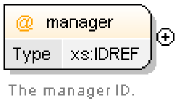
xs:element properties

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.

Property Name	Description	Possible Values	Mentions
Default	Default value of the element. A default value is automatically assigned to the element when no other value is specified.	Any string	The fixed and default attributes are mutually exclusive.
Fixed	A simple content element may be fixed to a specific value using this attribute. A fixed value is also automatically assigned to the element and you cannot specify another value.	Any string	The fixed and default attributes are mutually exclusive.
Min Occurs	Minimum number of occurrences of the element.	A numeric positive value. Default value is 1	Only for references/local elements
Max Occurs	Maximum number of occurrences of the element.	A numeric positive value. Default value is 1	Only for references/local elements
Substitution Group	Qualified name of the head of the substitution group to which this element belongs.	All declared elements	For global and reference elements
Abstract	Controls whether the element may be used directly in instance XML documents. When set to true, the element may still be used to define content models, but it must be substituted through a substitution group in the instance document.	true/false	For global elements and element references
Form	Defines if the element is "qualified" (i.e., belongs to the target namespace) or "unqualified" (i.e., doesn't belong to any namespace).	unqualified/qualified	Only for local elements
Nillable	When this attribute is set to true, the element can be declared as nil using an `xsi:nil` attribute in the instance documents.	true/false	For global elements and element references
Block	Controls whether the element can be subject to a type or substitution group substitution. '#all' blocks any substitution, 'substitution' blocks any substitution through substitution groups and 'extension'/'restriction' block any substitution (both through `xsi:type` and substitution groups) by elements or types, derived	#all, restriction, extension,substitution, extension restriction, extension substitution, restriction substitution, restriction extension substitution	For global elements and element references

Property Name	Description	Possible Values	Mentions
	respectively by extension or restriction from the type of the element. Its default value is defined by the `blockDefault` attribute of the parent `xs:schema`.		
Final	Controls whether the element can be used as the head of a substitution group for elements whose types are derived by extension or restriction from the type of the element. Its default value is defined by the `finalDefault` attribute of the parent `xs:schema`.	#all, restriction, extension, restriction extension, [Empty]	For global elements and element references
ID	The component id.	Any id	For all elements.
Component	The edited component name.	Not editable property.	For all elements.
Namespace	The component namespace.	Not editable property.	For all elements.
System ID	The component system id.	Not editable property.	For all elements.

`xs:attribute`



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

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

xs:attribute properties

Property Name	Description	Possible Value	Mentions
Name	Attribute name. Always required.	Any NCName for global/local attributes, all declared attributes' QName for references.	For all local or global attributes. If missing, will be displayed as '[attribute]' in the diagram.
Is Reference	When set, the local attribute is a reference.	true/false	For local attributes.
Type	Qualified name of a simple type.	All global simple types and built-in simple types. In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for creating anonymous simple types more easily.	For all attributes. For references, the type is set to the referred attribute.
Default	Default value. When specified, an attribute is	Any string	For all local or global attributes. For references the

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

`xs:complexType`

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

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

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

xs:complexType properties

Property Name	Description	Possible Values	Mentions
Name	The name of the complex type. Always required.	Any NCName	Only for global complex types. If missing, will be displayed as '[complexType]' in diagram.
Base Type Definition	The name of the extended/restricted types.	Any from the declared simple or complex types.	For complex types with simple or complex content.
Derivation Method	The derivation method.	restriction/ extension	Only when base type is set. If the base type is a simple type, the derivation method is always extension.
Content	The content of the complex type.	simple/ complex	For complex types which extend/restrict a base type. It is automatically detected.
Content Mixed	Specifies if the complex content model will be mixed.	true/false	For complex contents.
Mixed	Specifies if the complex type content model will be mixed.	true/false	For global and anonymous complex types.
Abstract	When set to `true`, this complex type cannot be used directly in the instance documents and needs to be substituted using an `xsi:type` attribute.	true/false	For global and anonymous complex types.
Block	Controls whether a substitution (either through a `xsi:type` or substitution groups) can be performed for a complex type, which is an extension or a restriction of the current complex type. This attribute can only block such substitutions (it cannot "unblock" them), which can also be blocked in the element definition. The default value is defined by the `blockDefault` attribute of `xs:schema`.	all, extension, restriction, extension restriction, [Empty]	For global complex types.
Final	Controls whether the complex type can be further derived by extension or restriction to create new complex types.	all, extension, restriction, extension restriction, [Empty]	For global complex types.
ID	The component id.	Any id	For all complex types.
Component	The edited component name.	Not editable property.	For all complex types.

Property Name	Description	Possible Values	Mentions
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

name
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.

Name	Description	Possible Values	Scope
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 (by list, union, derivation or all).	#all, list, restriction, union, list restriction, list union, restriction union. In addition, [Empty] proposal is present for set empty string as value.	Only for global simple types.
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`

mixed-level

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.

Property Name	Description	Possible Values	Mentions
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`

area-properties
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.
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`

include: xhtml11-model-1.xsd

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`

import: http://www.renderx.com/XSL/Extensions (rxxsd.xsd)

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

Property Name	Description	Possible Values
Component	The component name	Not editable property.

xs:redefine

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

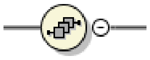


Figure 43: An `xs:sequence` in diagram

`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*.

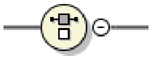


Figure 44: An `xs:choice` in diagram

`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*.

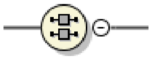


Figure 45: An `xs:all` in diagram

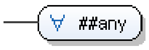
`xs:all` specifies that the child elements can appear in any order. Each child element can occur 0 or 1 time. See more info at *http://www.w3.org/TR/xmlschema-1/#element-all*.

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.
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*.

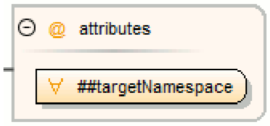
The graphical representation also contains the value for the `minOccurs` and `maxOccurs` properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the wildcard are drawn using dotted lines if the wildcard is optional.

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

Property Name	Description	Possible Values
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`

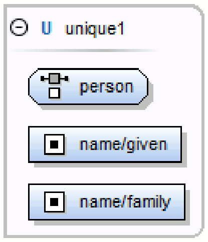


Enables the author to extend the XML document with attributes not specified by the schema. See more info at *http://www.w3.org/TR/xmlschema-1/#element-anyAttribute*.

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`



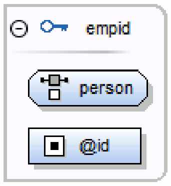
Defines that an element or an attribute value must be unique within the scope. See more info at *http://www.w3.org/TR/xmlschema-1/#element-unique*.

xs:unique properties

Property Name	Description	Possible Values
Name	The unique name. Always required.	Any NCName.
ID	The component id.	Any ID.

Property Name	Description	Possible Values
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

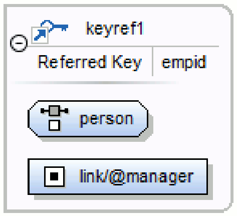


Specifies an attribute or element value as a key (unique, non-nullable and always present) within the containing element in an instance document. See more info at *http://www.w3.org/TR/xmlschema-1/#element-key*.

xs:key properties

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

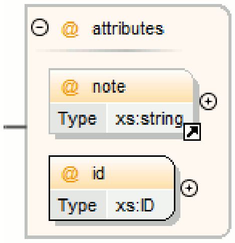
Property Name	Description	Possible Values
XPath	Relative XPath expression identifying the field(s) composing the key, key reference, or unique constraint.	An XPath expression.
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not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

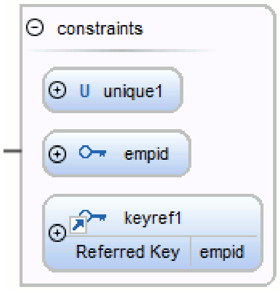


Groups all attributes and attribute groups belonging to a complex type.

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

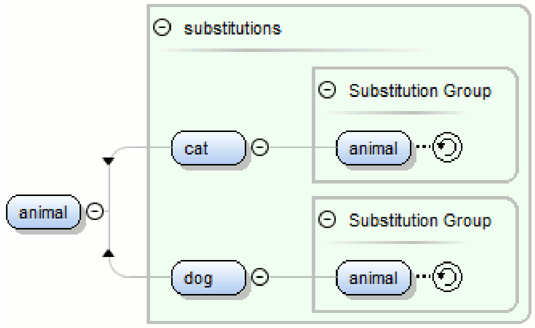


Groups all constraints (*xs:key*, *xs:keyRef* or *xs:unique*) belonging to an element.

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

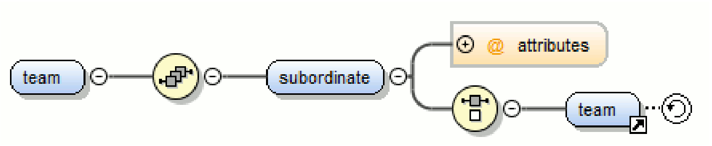
The following editing and navigation features work for all types of schema components:

- 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 Editor plugin .

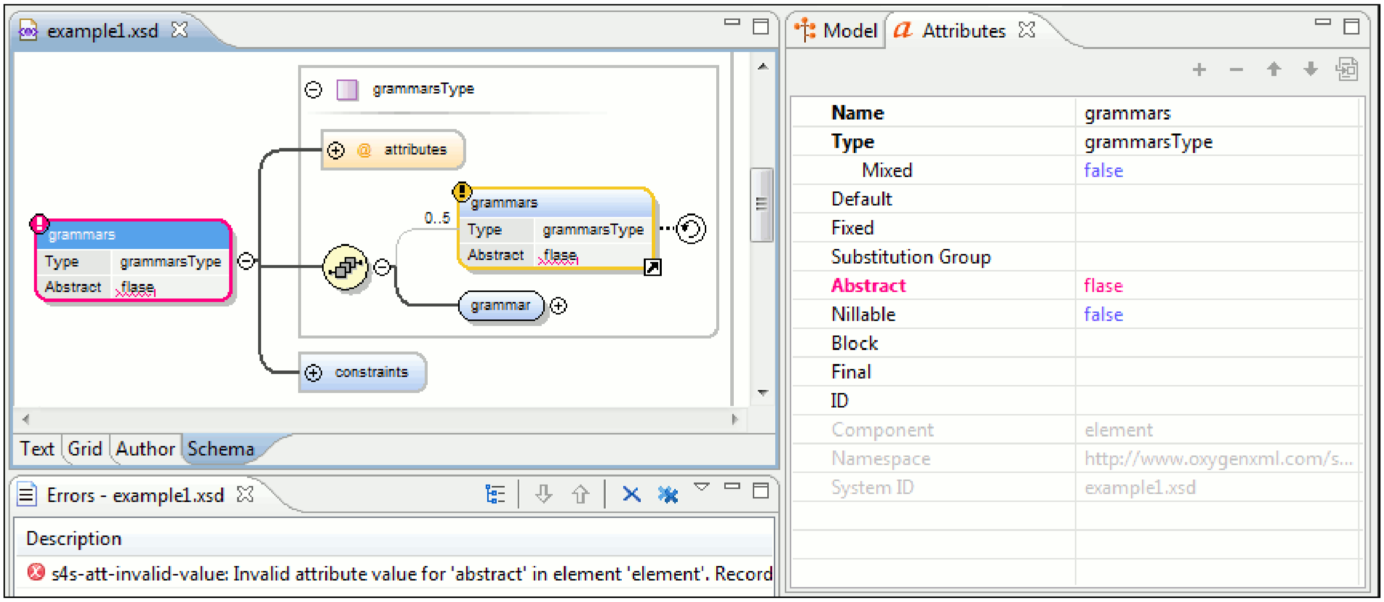
  Tip: If an XML Schema referenced by the current opened schema was modified on disk, the change will be detected and you will be asked to refresh the current schema contents.

- Recursive references are marked with a recurse symbol: ⟲. Click this symbol to navigate between the element declaration and its reference.



Schema Validation

Validation for the **Design** mode is seamlessly integrated in the Oxygen XML Editor plugin *XML documents validation* capability.



Name	grammars
Type	grammarsType
Mixed	false
Default	
Fixed	
Substitution Group	
Abstract	flase
Nillable	false
Block	
Final	
ID	
Component	element
Namespace	http://www.oxygenxml.com/s...
System ID	example1.xsd

Description

s4s-att-invalid-value: Invalid attribute value for 'abstract' in element 'element'. Record

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/redefine 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.

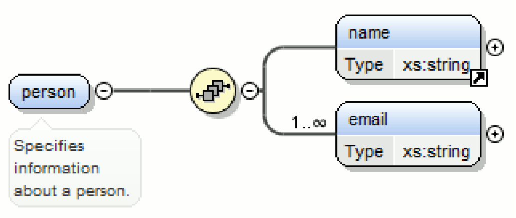
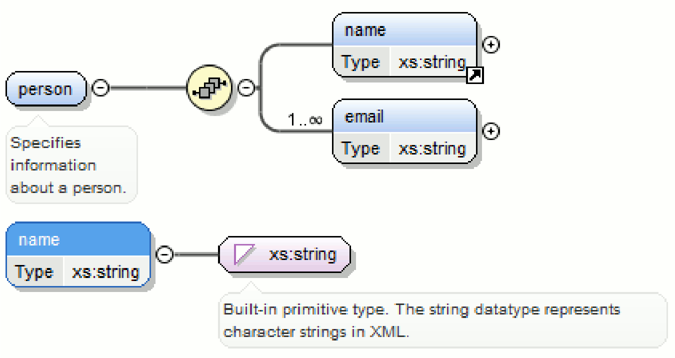


Figure 47: Extracting a Global Element

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


name
Type xs:string
person
Specifies information about a person.
1..∞
email
Type xs:string
name
Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.

- **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.

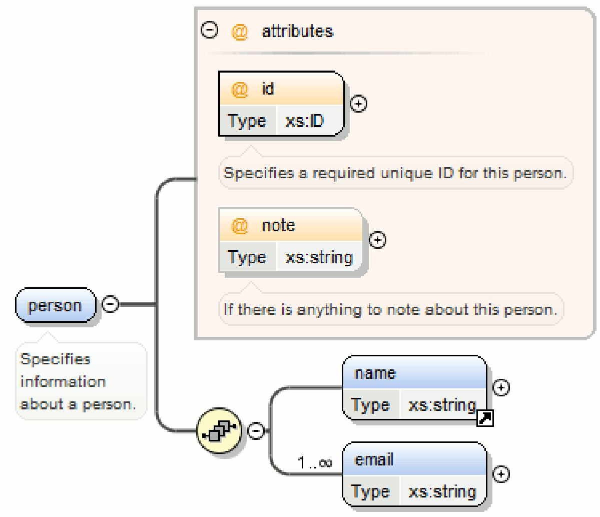
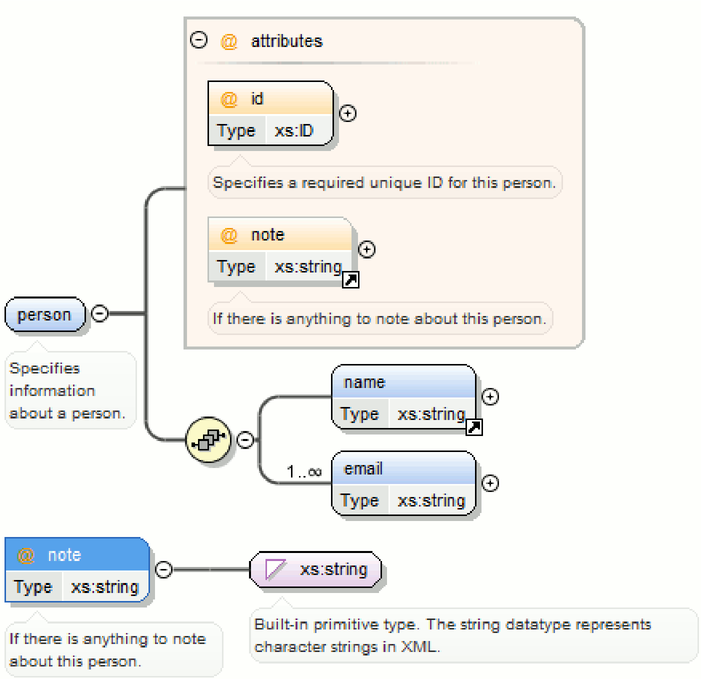
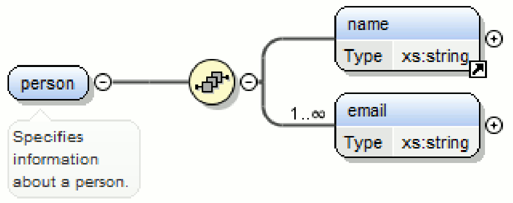

attributes
id
Type xs:ID
Specifies a required unique ID for this person.
note
Type xs:string
If there is anything to note about this person.
person
Specifies information about a person.
name
Type xs:string
1..∞
email
Type xs:string

Figure 48: Extracting a Global Attribute

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


attributes
id
Type xs:ID
Specifies a required unique ID for this person.
note
Type xs:string
If there is anything to note about this person.
person
Specifies information about a person.
name
Type xs:string
1..∞
email
Type xs:string
note
Type xs:string
xs:string
If there is anything to note about this person.
Built-in primitive type. The string datatype represents character strings in XML.

- **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:

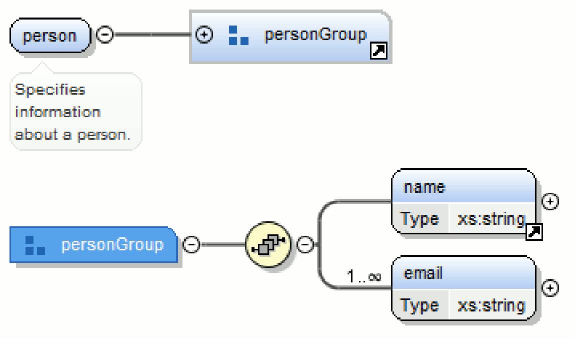


Figure 49: Extracting a Global Group

- **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.

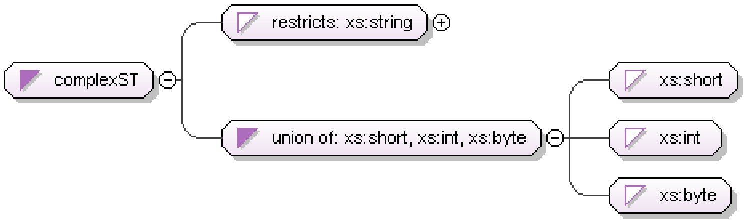
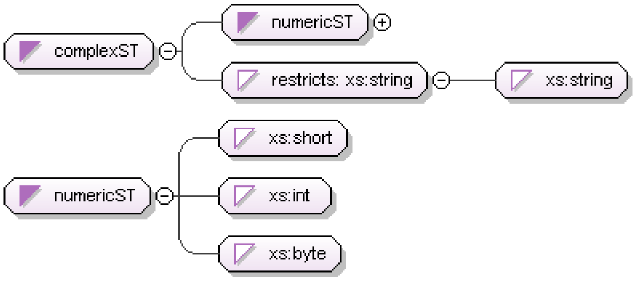


Figure 50: Extracting a Global Simple Type

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



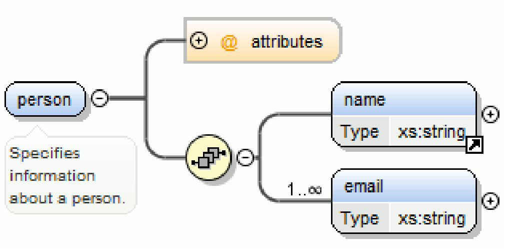
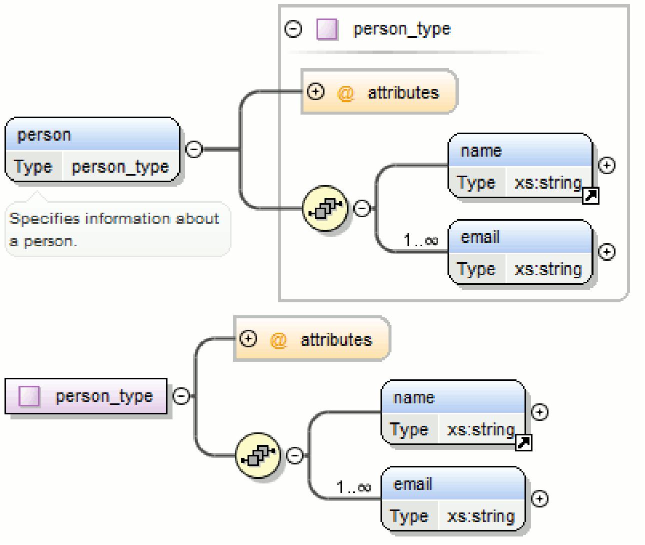


Figure 51: Extracting a Global Complex Type

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

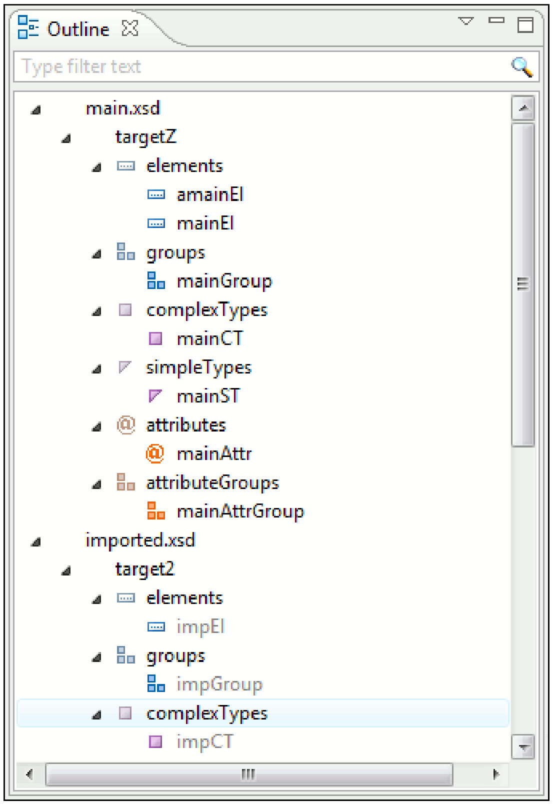


- **Rename Component** - Rename the selected component.
- **Cut (Ctrl - X)** - Cut the selected component(s).

- **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**.



- main.xsd
  - targetZ
    - elements
      - amainEl
      - mainEl
    - groups
      - mainGroup
    - complexTypes
      - mainCT
    - simpleTypes
      - mainST
    - attributes
      - mainAttr
    - attributeGroups
      - mainAttrGroup
- imported.xsd
  - target2
    - elements
      - impEl
    - groups
      - impGroup
    - complexTypes
      - impCT

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:

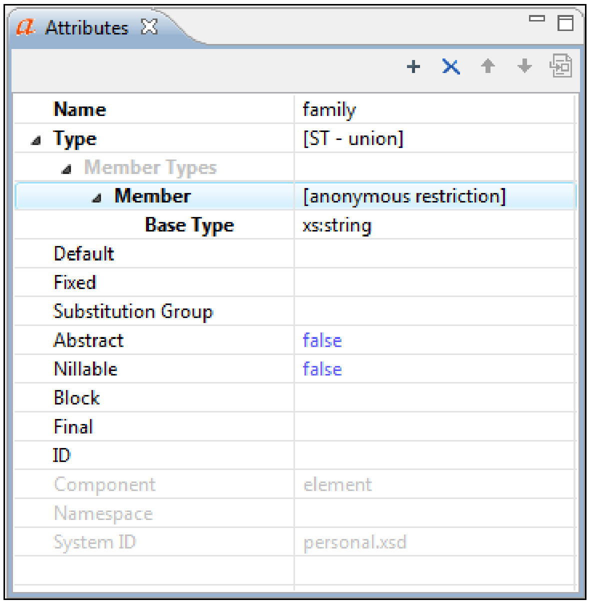
- ***** - any string
- **?** - any character
- **,** - patterns separator

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.

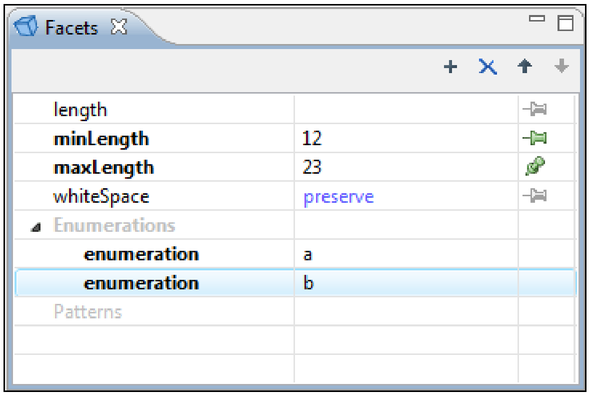
The **Attributes** view has five actions available on the toolbar and also on the contextual menu:

- + **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**.



length	
minLength	12
maxLength	23
whiteSpace	preserve
Enumerations	
enumeration	a
enumeration	b
Patterns	

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.

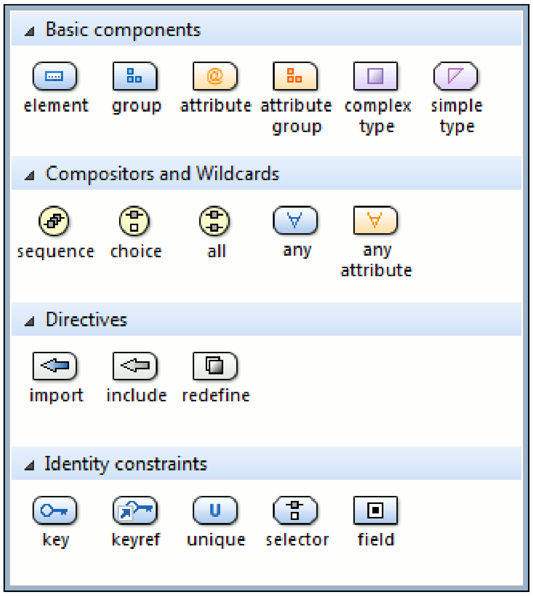
Editing Patterns

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

The Palette View

Designed to offer quick access to XML Schema components, the **Palette** view improves the usability of the XML Schema diagram builder allowing you drag components from the **Palette** view and drop them into the **Design** 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 Editor plugin provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

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 Editor plugin warns you if the current module is not part of the dependencies graph computed for the main schema. In this case, it considers the current module as the main schema.

The advantages of editing in the context of main file include:

- correct validation of a module in the context of a larger schema structure;
- content completion assistant displays all the referable components valid in the current context. This include components defined in modules other than the currently edited one;
- the **Outline** displays the components collected from the entire schema structure;

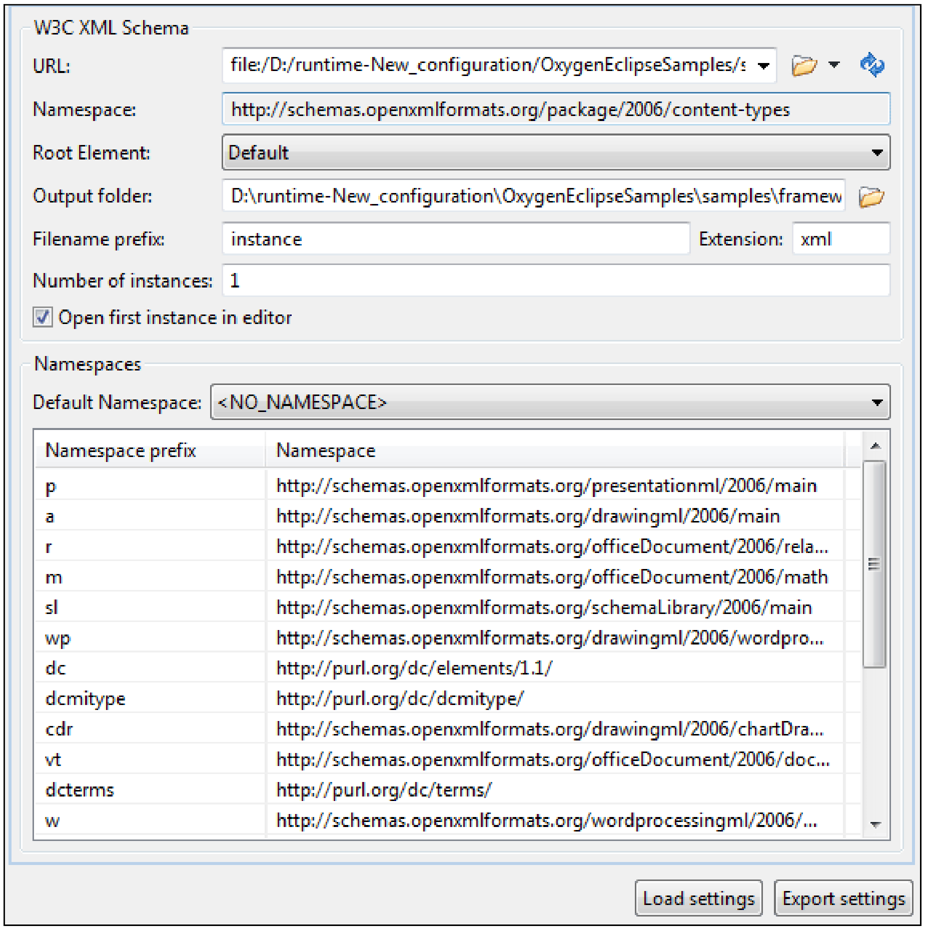
Create an XML Schema From a Relational Database Table

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\framew

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/...

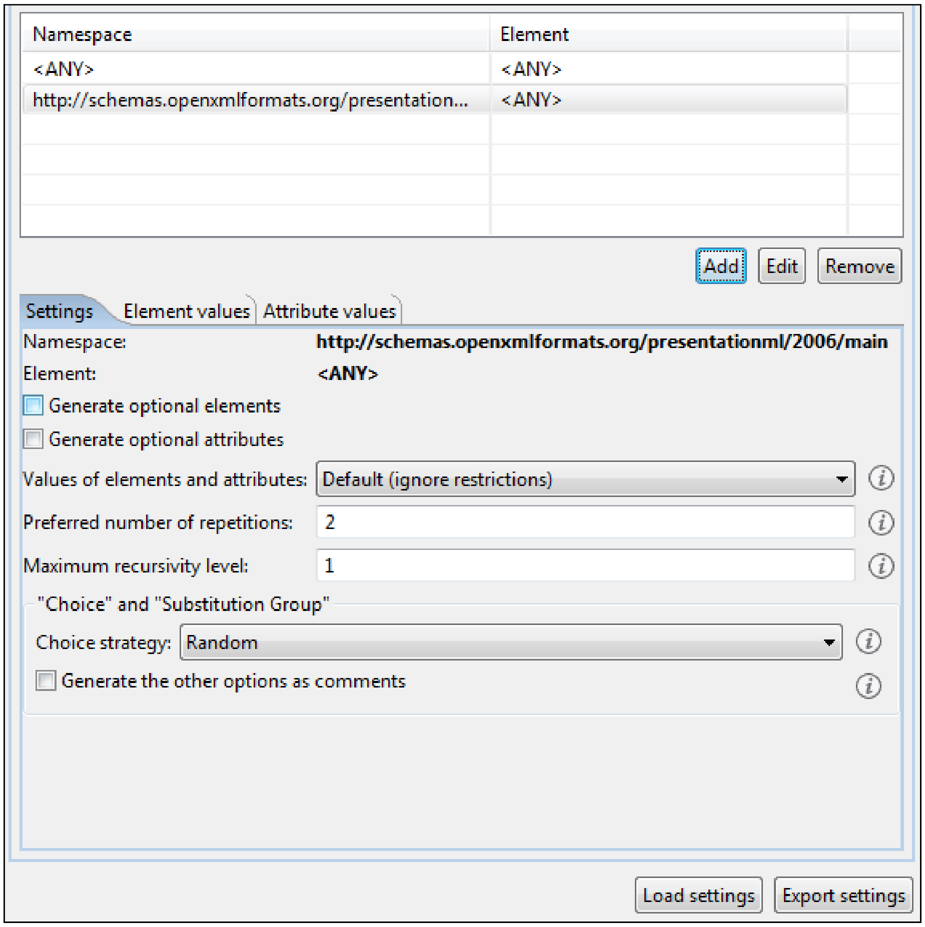
Figure 56: The Generate Sample XML Files Dialog

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.



Namespace	Element
<ANY>	<ANY>
http://schemas.openxmlformats.org/presentation...	<ANY>

Namespace: **http://schemas.openxmlformats.org/presentationml/2006/main**

Element: **<ANY>**

□ Generate optional elements

□ Generate optional attributes

Values of elements and attributes: Default (ignore restrictions)

Preferred number of repetitions: 2

Maximum recursivity level: 1

Choice strategy: Random

□ Generate the other options as comments

Figure 57: The Generate Sample XML Files Dialog

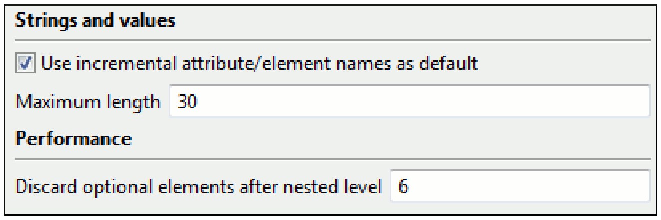
- **Namespace / Element table** - Allows you to set a namespace for each element name that appears in an XML document instance. The following prefix-to-namespace associations are available:
  - All elements from all namespaces (<ANY> - <ANY>). This is the default setting and can be customized from the *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

☑ Use incremental attribute/element names as default

Maximum length: 30

Performance

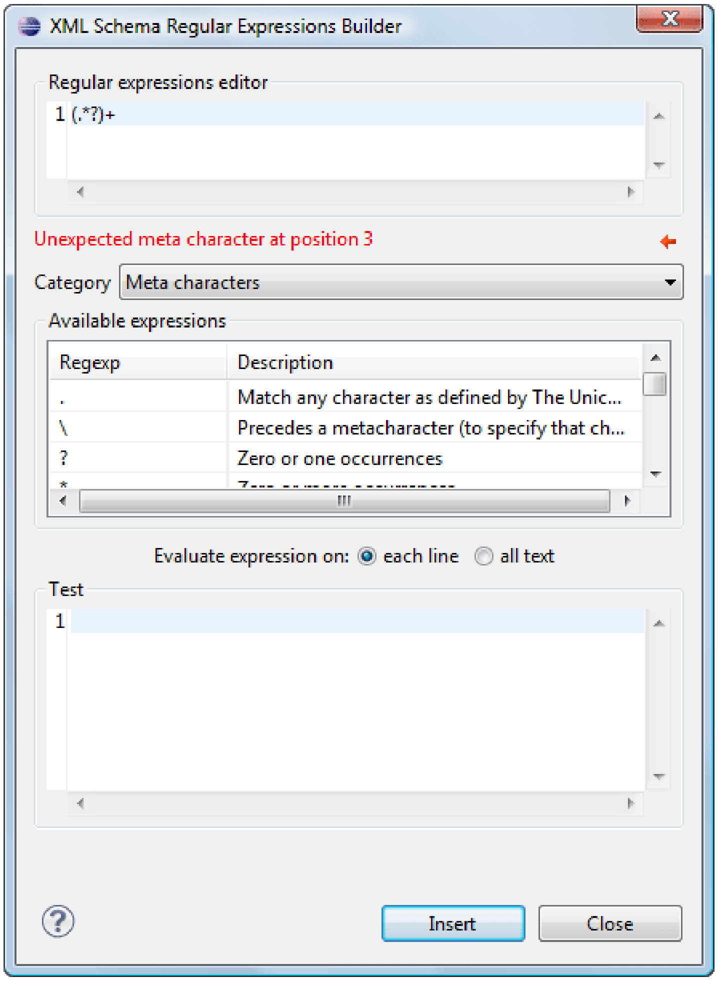
Discard optional elements after nested level: 6

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**.


XML Schema Regular Expressions Builder
Regular expressions editor
Category
Available expressions
Test
Insert
Close

1 (.*?)+

Unexpected meta character at position 3

Category: Meta characters

Regexp	Description
.	Match any character as defined by The Unic...
\	Precedes a metacharacter (to specify that ch...
?	Zero or one occurrences
*	Zero or more occurrences

Evaluate expression on: ● each line ○ all text

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**.

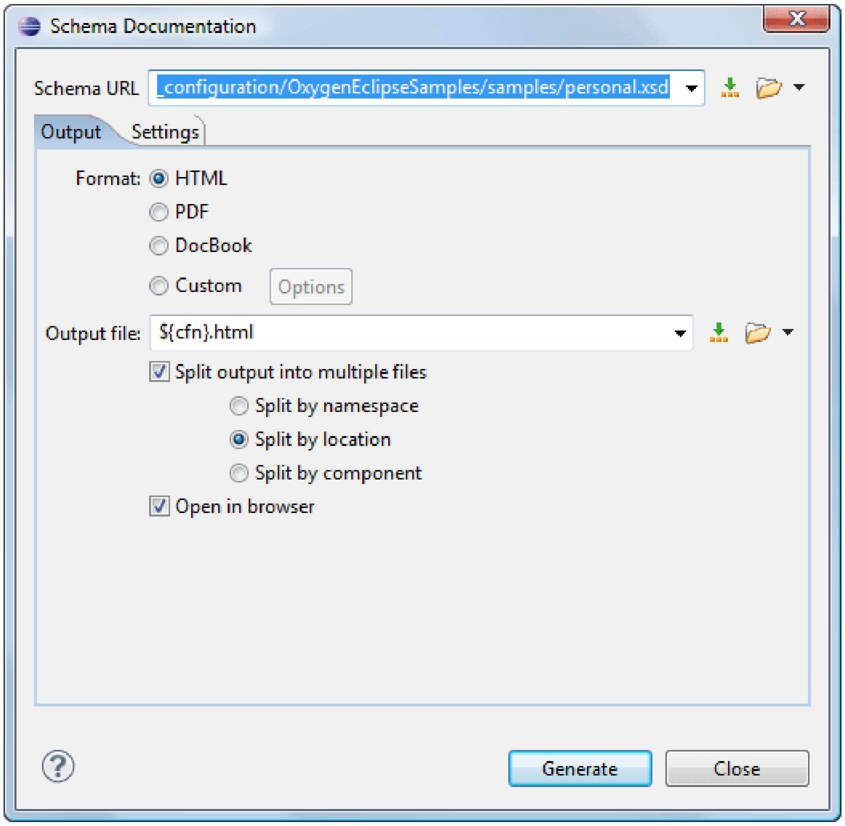
The regular expression builder cannot be used to insert regular expressions in *the grid version* or *the schema version* of a document editor. Accordingly, the **Insert** button of the dialog will be disabled if the current document is edited in grid mode.

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 Editor 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.



Schema Documentation

Schema URL: _configuration/OxygenEclipseSamples/samples/personal.xsd

Format: ● HTML
○ PDF
○ DocBook
○ Custom

Output file: ${cfn}.html

☑ Split output into multiple files
○ Split by namespace
● Split by location
○ Split by component

☑ Open in browser

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.

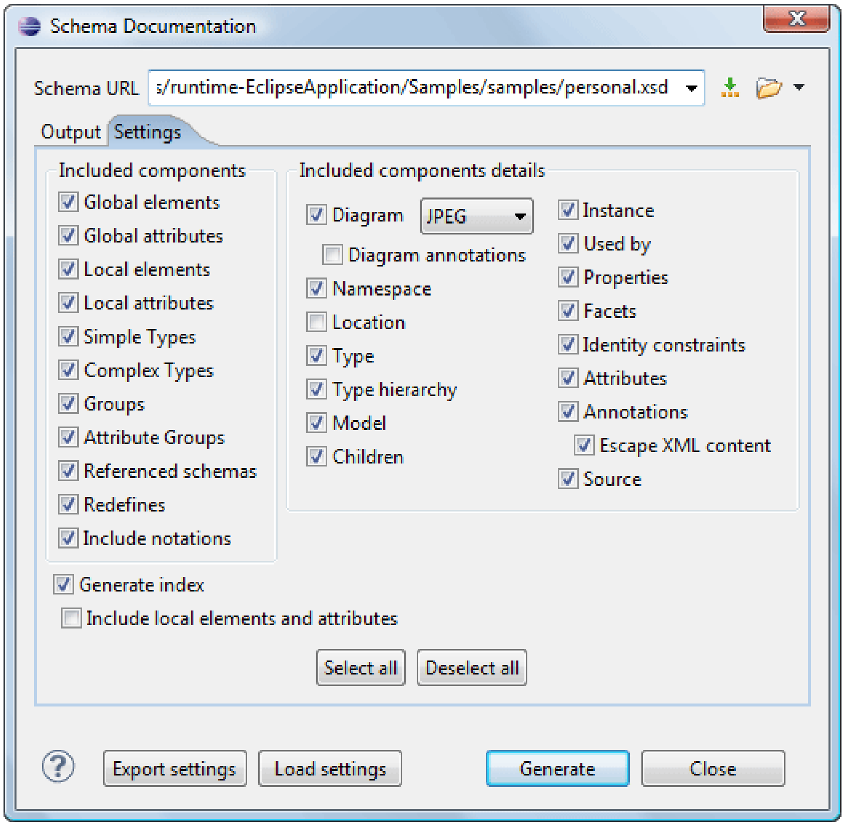
The following options are available in the **Settings** tab:

- **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 proving a custom stylesheet.
- **Output file** - Name of the output file.
- **Split output into multiple files** - Instructs the application to split the output into multiple files. You can choose to split them by namespace, location or component name.
- **Open in Browser/System Application** - Opens the result in the system application associated with the output file type.

  Note: 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.



Schema URL s/runtime-EclipseApplication/Samples/samples/personal.xsd

Included components

☑ Global elements
☑ Global attributes
☑ Local elements
☑ Local attributes
☑ Simple Types
☑ Complex Types
☑ Groups
☑ Attribute Groups
☑ Referenced schemas
☑ Redefines
☑ Include notations

Included components details

☑ Diagram JPEG
□ Diagram annotations
☑ Namespace
□ Location
☑ Type
☑ Type hierarchy
☑ Model
☑ Children
☑ Instance
☑ Used by
☑ Properties
☑ Facets
☑ Identity constraints
☑ Attributes
☑ Annotations
☑ Escape XML content
☑ Source

☑ Generate index
□ Include local elements and attributes

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.

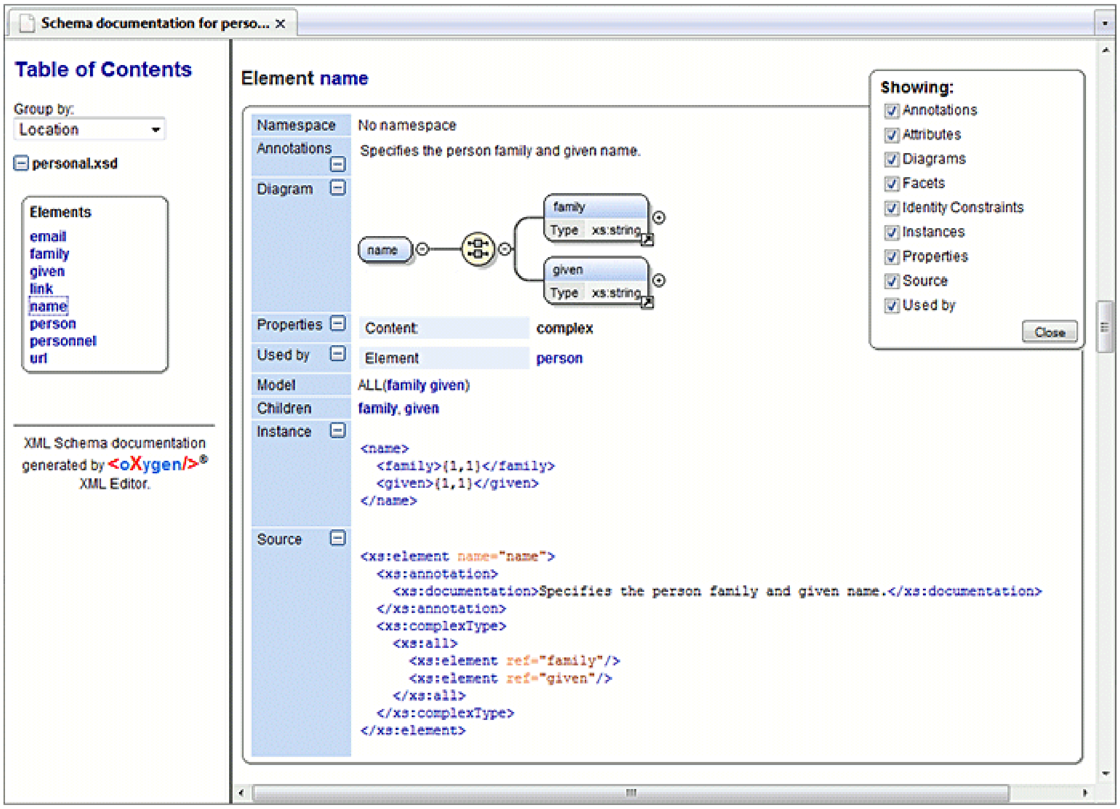
- **Model** - Displays the model (sequence, choice, all) presented in BNF form. Different separator characters are used depending on the information item used:
    - `xs:all` - its children will be separated by space characters;
    - `xs:sequence` - its children will be separated by comma characters;
    - `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 `<b>`, `<i>`, `<u>`, `<ul>`, `<li>`, etc.) are rendered in the generated HTML documentation.

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



Schema documentation for perso...

Table of Contents

Group by: Location

personal.xsd

Elements

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

XML Schema documentation generated by <oXygen/>® XML Editor.

Element 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	`<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>
```

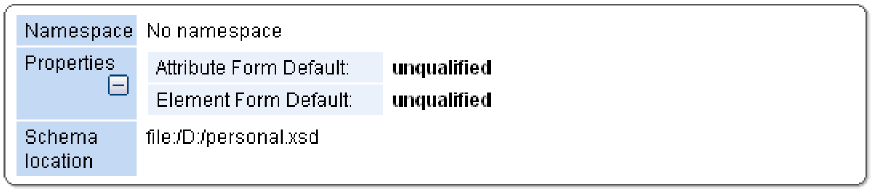
Showing:

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

Close

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.

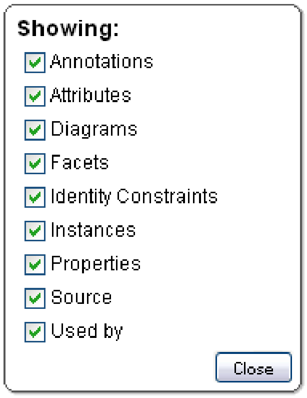


Namespace	No namespace	
Properties	Attribute Form Default:	**unqualified**
	Element Form Default:	**unqualified**
Schema location	file:/D:/personal.xsd	

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:

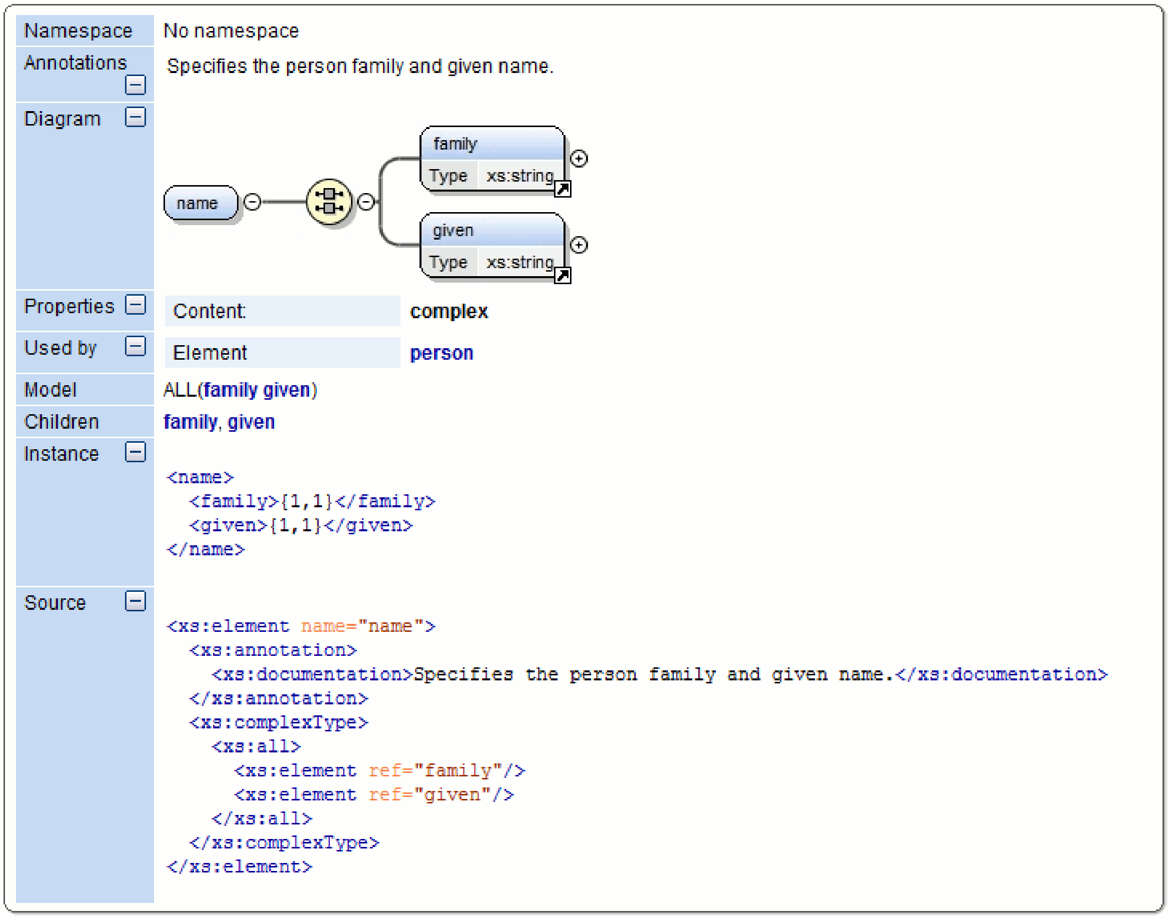


Showing:

☑ Annotations
☑ Attributes
☑ Diagrams
☑ Facets
☑ Identity Constraints
☑ Instances
☑ Properties
☑ Source
☑ Used by

Figure 63: The Showing View

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



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

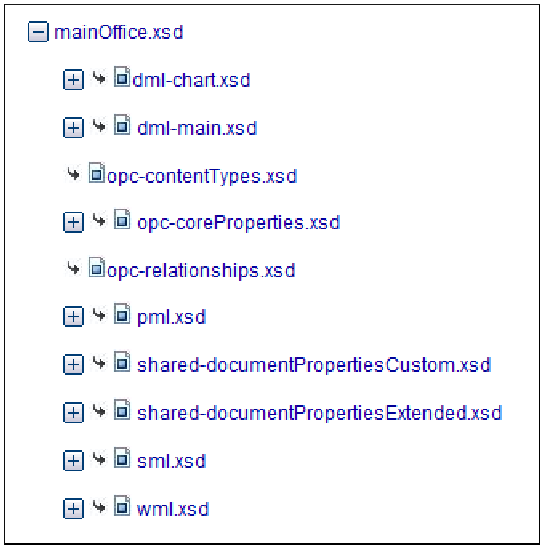
```
<name>
  <family>{1,1}</family>
  <given>{1,1}</given>
</name>
```

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>
```

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 Editor plugin (*DocBook PDF* or *DocBook HTML*) or configure your own scenario for it.

For the custom format, you can specify your stylesheet to transform the intermediary XML generated in the documentation process. You have to write your stylesheet based on the schema `xsdDocSchema.xsd` from `{INSTALLATION_DIRECTORY}/frameworks/schema_documentation`. You can create a custom format starting from one of the stylesheets used in the predefined HTML, PDF, and DocBook formats. These stylesheets are available in `{INSTALLATION_DIRECTORY}/frameworks/schema_documentation/xsl`.

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

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

The script command-line parameter is the relative path to the exported XML settings file. The files which are specified with relative paths in the exported XML settings are made absolute, relative to the folder where the script is ran from.

XML Configuration File

```
<serialized>
    <map>
        <entry>
```

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

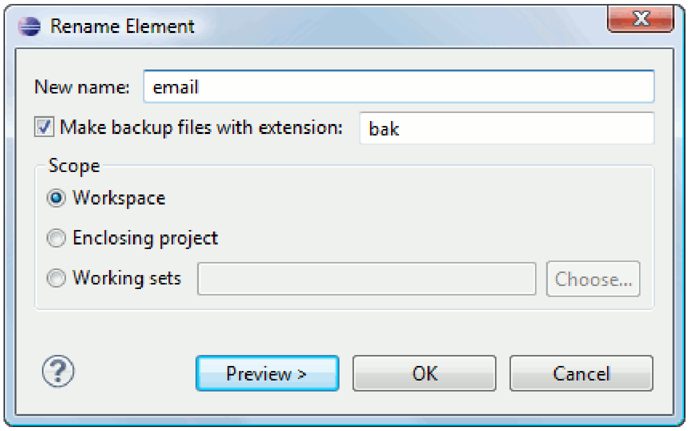
All the following actions can be applied on `attribute`, `attributeGroup`, `element`, `group`, `key`, `unique`, `keyref`, `notation`, `simple`, or `complex` types:

- **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:



Rename Element

New name: email

☑ Make backup files with extension: bak

Scope

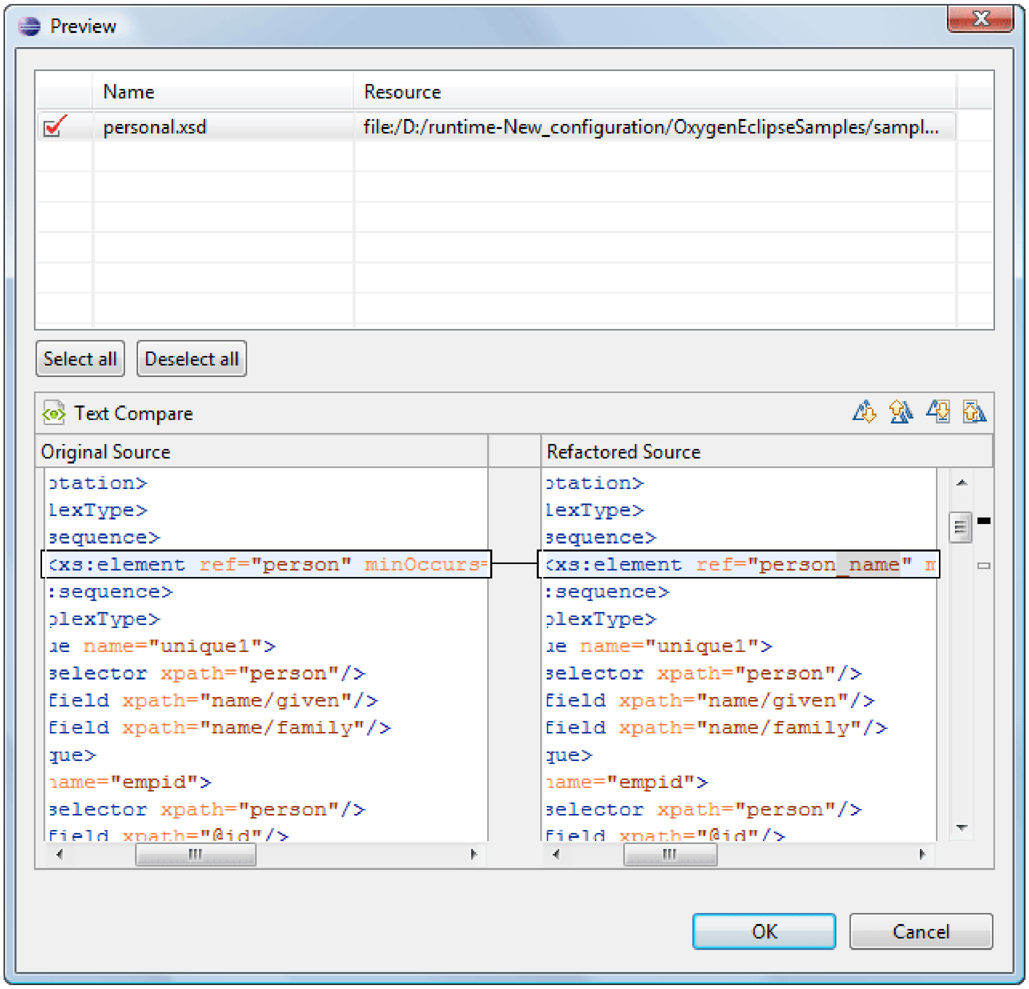
● Workspace

○ Enclosing project

○ Working sets

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:



	Name	Resource
☑	personal.xsd	file:/D:/runtime-New_configuration/OxygenEclipseSamples/sampl...

Original Source:

```
otation>
lexType>
sequence>
<xs:element ref="person" minOccurs=
:sequence>
plexType>
ue name="unique1">
selector xpath="person"/>
field xpath="name/given"/>
field xpath="name/family"/>
que>
name="empid">
selector xpath="person"/>
field xpath="@id"/>
```

Refactored Source:

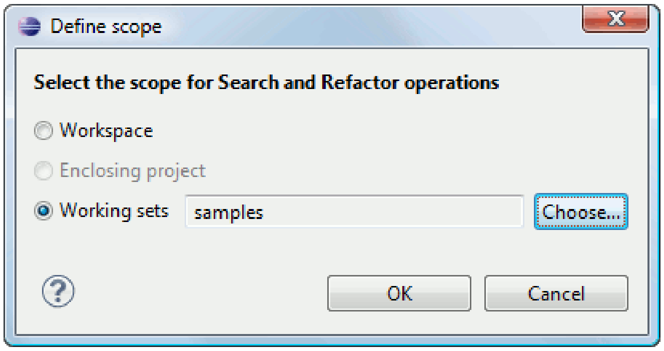
```
otation>
lexType>
sequence>
<xs:element ref="person_name" m
:sequence>
plexType>
ue name="unique1">
selector xpath="person"/>
field xpath="name/given"/>
field xpath="name/family"/>
que>
name="empid">
selector xpath="person"/>
field xpath="@id"/>
```

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



Select the scope for Search and Refactor operations

○ Workspace

○ Enclosing project

● Working sets samples

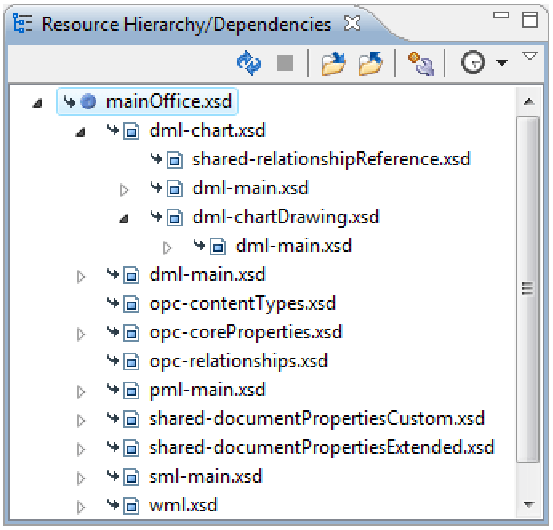
The defined scope is applied to all future search and refactor operations until you alter it again. Contextual menu actions allow you to add or delete files, folders and other resources to the working set structure.

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.

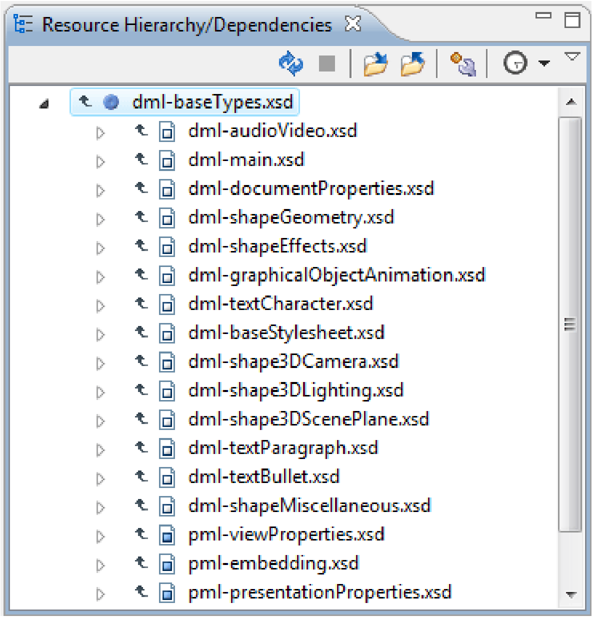


Resource Hierarchy/Dependencies

- mainOffice.xsd
  - dml-chart.xsd
    - shared-relationshipReference.xsd
    - dml-main.xsd
    - dml-chartDrawing.xsd
      - dml-main.xsd
  - dml-main.xsd
  - opc-contentTypes.xsd
  - opc-coreProperties.xsd
  - opc-relationships.xsd
  - pml-main.xsd
  - shared-documentPropertiesCustom.xsd
  - shared-documentPropertiesExtended.xsd
  - sml-main.xsd
  - wml.xsd

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.



Resource Hierarchy/Dependencies

- dml-baseTypes.xsd
  - dml-audioVideo.xsd
  - dml-main.xsd
  - dml-documentProperties.xsd
  - dml-shapeGeometry.xsd
  - dml-shapeEffects.xsd
  - dml-graphicalObjectAnimation.xsd
  - dml-textCharacter.xsd
  - dml-baseStylesheet.xsd
  - dml-shape3DCamera.xsd
  - dml-shape3DLighting.xsd
  - dml-shape3DScenePlane.xsd
  - dml-textParagraph.xsd
  - dml-textBullet.xsd
  - dml-shapeMiscellaneous.xsd
  - pml-viewProperties.xsd
  - pml-embedding.xsd
  - pml-presentationProperties.xsd

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 ↻.

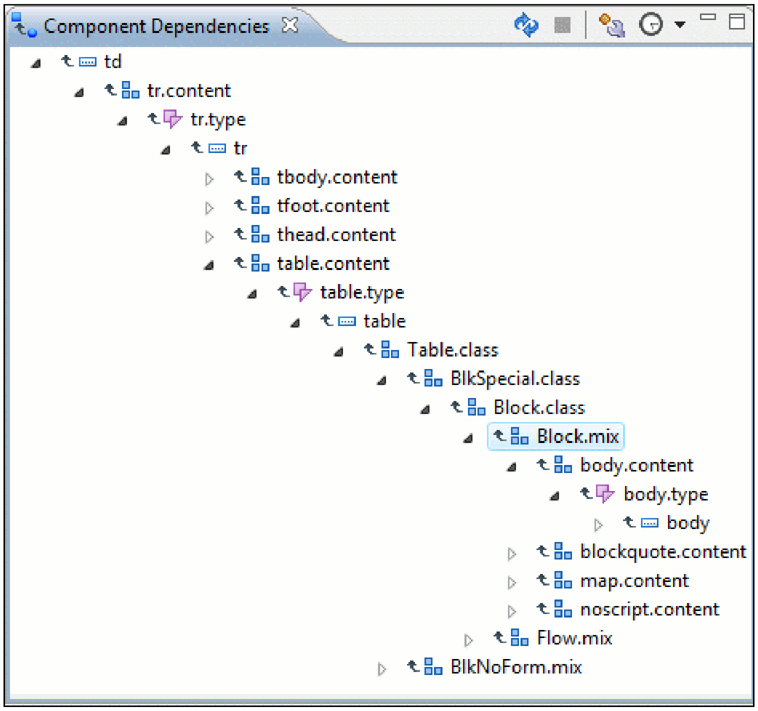
Component Dependencies View

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).



Component Dependencies

```
td
    tr.content
        tr.type
            tr
                tbody.content
                tfoot.content
                thead.content
                table.content
                    table.type
                        table
                            Table.class
                                BlkSpecial.class
                                    Block.class
                                        Block.mix
                                            body.content
                                                body.type
                                                    body
                                            blockquote.content
                                            map.content
                                            noscript.content
                                        Flow.mix
                                BlkNoForm.mix
```

Figure 70: Component Dependencies View - Hierarchy for `xhtml11.xsd`

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

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

The contextual menu contains the following actions:

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

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

Highlight Component Occurrences

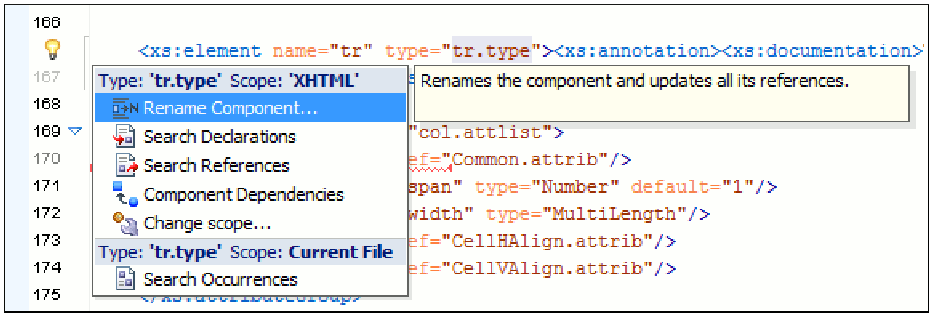
When a component (for example types, elements, attributes) is found at current cursor position, Oxygen XML Editor plugin performs a search over the entire document to find the component declaration and all its references. When found, they are highlighted both in the document and in the stripe bar, at the right side of the document. Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.

This feature is set on automatic search by default and can be configured in the **Options** > **Preferences** > **Editor** > **Mark Occurrences** page. A search can also be triggered with the **Search** > **Search Occurrences in File** **(Ctrl+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).



```
<xs:element name="tr" type="tr.type"><xs:annotation><xs:documentation>
"col.attlist">
ef="Common.attrib"/>
span" type="Number" default="1"/>
width" type="MultiLength"/>
ef="CellHAlign.attrib"/>
ef="CellVAlign.attrib"/>
</xs:attributeGroup>
```

Renames the component and updates all its references.

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.

Linking Between Development and Authoring

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

Editing Relax NG Schemas

Oxygen XML Editor plugin provides a special type of editor for Relax NG schemas. This editor presents the usual text view of an XML document synchronized in real time with an outline view. The outline view has two display modes: the *standard outline* mode and the *components* mode.

Relax NG Schema Diagram

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

Introduction

Oxygen XML Editor plugin provides a simple, expressive, and easy to read **Schema Diagram** view for Relax NG schemas.

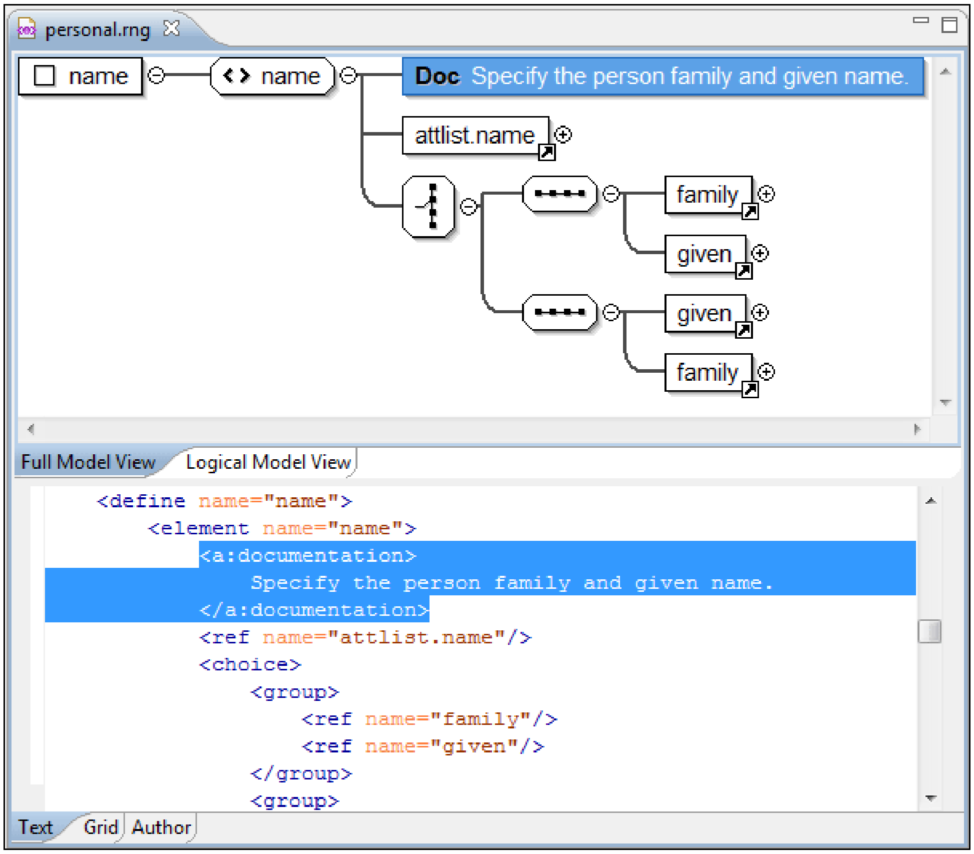
With this new feature you can easily develop complex schemas, print them on multiple pages or save them as JPEG, PNG, or BMP images. It helps both schema authors in developing the schema and content authors who are using the schema to understand it.

Oxygen XML Editor plugin is the only XML editor to provide a side by side source and diagram presentation and have them real-time synchronized:

- the changes you make in the Editor are immediately visible in the Diagram (no background parsing).
- changing the selected element in the diagram selects the underlaying code in the source editor.

Full Model View

When you create a new schema document or open an existing one, the editor panel is divided in two sections: one containing the schema diagram and the second the source code. The **Diagram** view has two tabbed panes offering a **Full Model View** and a **Logical Model View**.



```
    <define name="name">
        <element name="name">
            <a:documentation>
                Specify the person family and given name.
            </a:documentation>
            <ref name="attlist.name"/>
            <choice>
                <group>
                    <ref name="family"/>
                    <ref name="given"/>
                </group>
                <group>
```

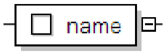
Figure 72: Relax NG Schema Editor - Full Model View

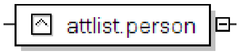
The following references can be expanded in place: patterns, includes, and external references. This expansion mechanism, coupled with the synchronization support, makes the schema navigation easy.

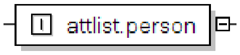
All the element and attribute names are editable: double-click any name to start editing it.

Symbols Used in the Schema Diagram

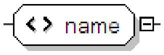
The **Full Model View** renders all the Relax NG Schema patterns with intuitive symbols:

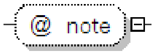
- 

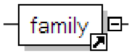
  - `define` pattern with the `name` attribute set to the value shown inside the rectangle (in this example `name`).
- 

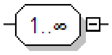
  - `define` pattern with the `combine` attribute set to `interleave` and the `name` attribute set to the value shown inside the rectangle (in this example `attlist.person`).
- 

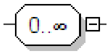
  - `define` pattern with the `combine` attribute set to `choice` and the `name` attribute set to the value shown inside the rectangle (in this example `attlist.person`).

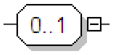
- 

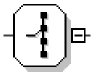
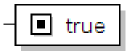
 - element pattern with the name attribute set to the value shown inside the rectangle (in this example name).
- 

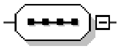
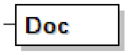
 - attribute pattern with the name attribute set to the value shown inside de rectangle (in this case note).
- 

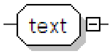
 - ref pattern with the name attribute set to the value shown inside the rectangle (in this case family).
- 

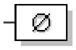
 - oneOrMore pattern.
- 

 - zeroOrMore pattern.
- 

 - optional pattern.
- 
 - choice pattern.
- 

 - value pattern, used for example inside a choice pattern.
- 
 - group pattern.
- 

 - pattern from the Relax NG Annotations namespace (http://relaxng.org/ns/compatibility/annotations/1.0) which is treated as a documentation element in a Relax NG schema.
- 

 - text pattern.
- 

 - empty pattern.

Logical Model View

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

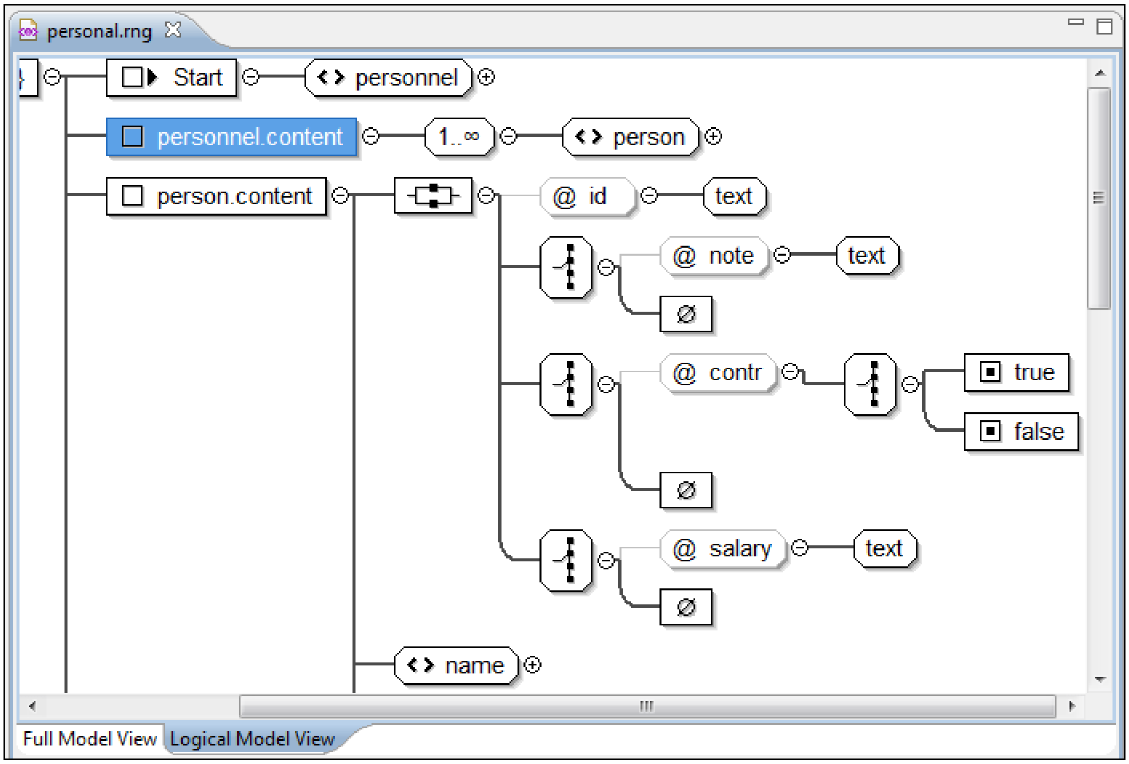


Figure 73: Logical Model View for a Relax NG Schema

Actions Available in the Diagram View

The contextual menu offers the following actions:

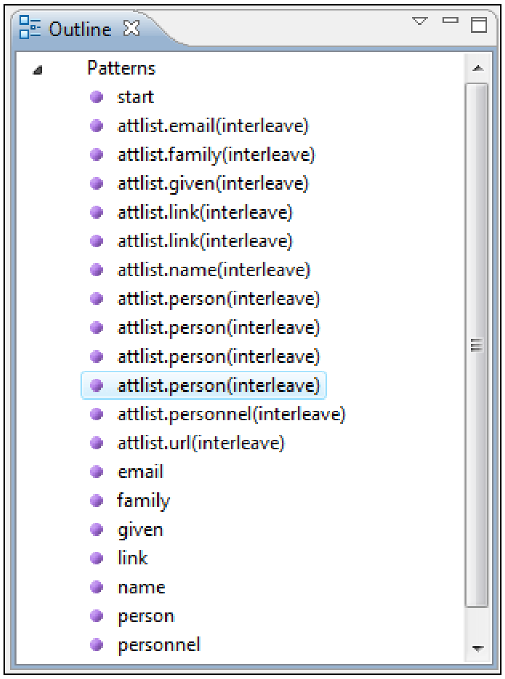
- **Append child** - Appends a child to the selected component.
- **Insert Before** - Inserts a component before the selected component.
- **Insert After** - Inserts a component after the selected component.
- **Edit attributes** - Edits the attributes of the selected component.
- **Remove** - Removes 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.



Patterns
start
attlist.email(interleave)
attlist.family(interleave)
attlist.given(interleave)
attlist.link(interleave)
attlist.link(interleave)
attlist.name(interleave)
attlist.person(interleave)
attlist.person(interleave)
attlist.person(interleave)
attlist.person(interleave)
attlist.personnel(interleave)
attlist.url(interleave)
email
family
given
link
name
person
personnel

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.
- 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 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.

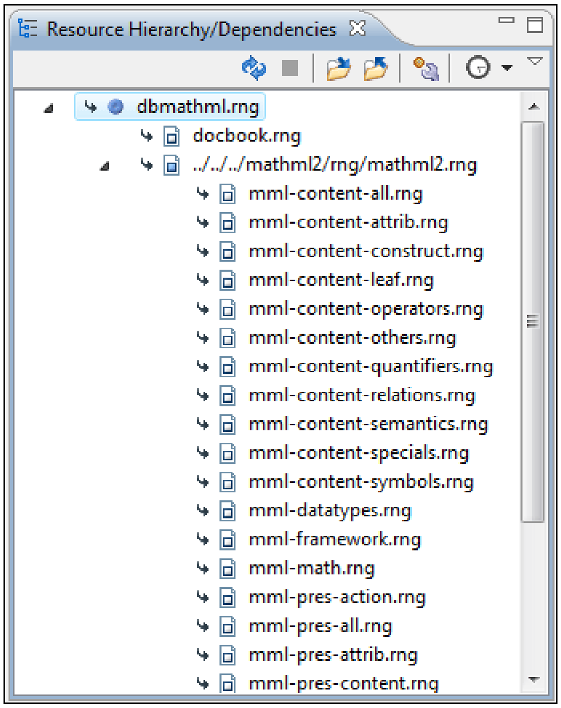
All the following actions can be applied on named `define` parameters only.

- **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.


Resource Hierarchy/Dependencies

- dbmathml.rng
  - docbook.rng
  - ../../../mathml2/rng/mathml2.rng
    - mml-content-all.rng
    - mml-content-attrib.rng
    - mml-content-construct.rng
    - mml-content-leaf.rng
    - mml-content-operators.rng
    - mml-content-others.rng
    - mml-content-quantifiers.rng
    - mml-content-relations.rng
    - mml-content-semantics.rng
    - mml-content-specials.rng
    - mml-content-symbols.rng
    - mml-datatypes.rng
    - mml-framework.rng
    - mml-math.rng
    - mml-pres-action.rng
    - mml-pres-all.rng
    - mml-pres-attrib.rng
    - mml-pres-content.rng

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.

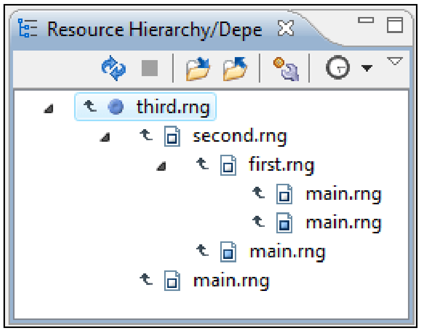


Figure 76: Resource Hierarchy/Dependencies View - Dependencies for `third.rng`

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.

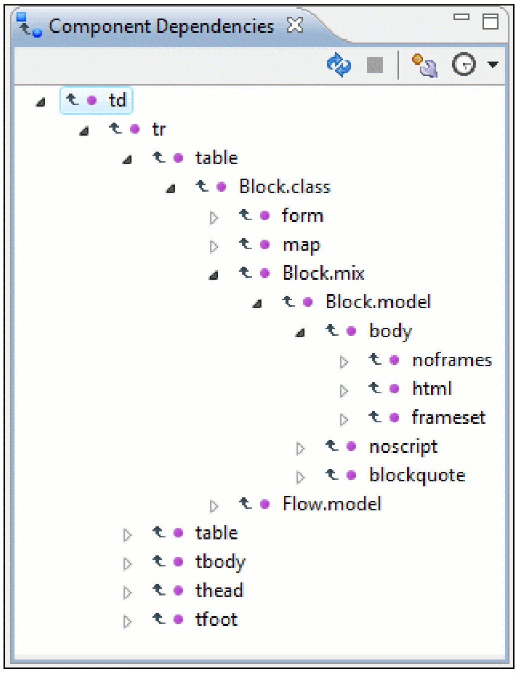


Figure 77: Component Dependencies View - Hierarchy for `xhtml.rng`

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:

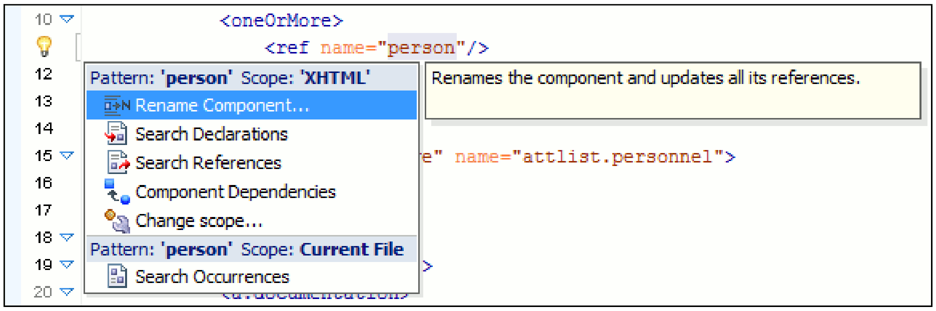
- **Go to First Reference** - Selects the first reference of the referred component from the current selected component in the dependencies tree.
- **Go to Component** - Shows the definition of the current selected component in the dependencies tree.

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).



```
<oneOrMore>
    <ref name="person"/>
e" name="attlist.personnel">
```

Renames the component and updates all its references.

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.*

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

To load the custom library, restart the Eclipse platform.

Linking Between Development and Authoring

The Author mode is available on the Relax NG schema presenting the schema similar with the Relax NG compact syntax. It links to imported schemas and external references. Embedded Schematron is supported only in Relax NG schemas with XML syntax.

Editing NVDL Schemas

Some complex XML documents are composed by combining elements and attributes from different namespaces. More, the schemas 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 Editor plugin provides a special type of editor for NVDL schemas. This editor presents the usual text view of an XML document synchronized in real time with an outline view. The outline view has two display modes: the *standard outline* mode and the *components* mode.

NVDL Schema Diagram

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

Introduction

Oxygen XML Editor plugin provides a simple, expressive, and easy to read Schema Diagram View for NVDL schemas.

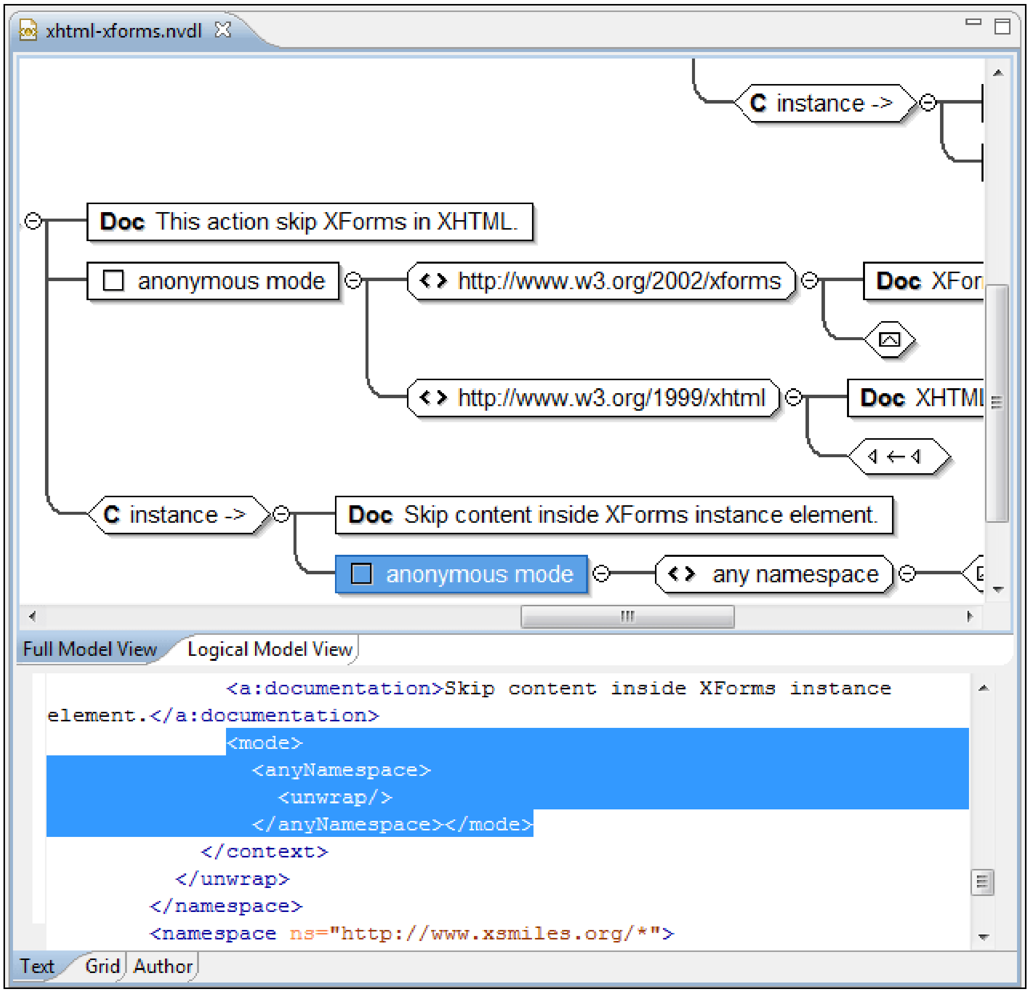
With this new feature you can easily develop complex schemas, print them on multiple pages or save them as JPEG, PNG, and BMP images. It helps both schema authors in developing the schema and content authors that are using the schema to understand it.

Oxygen XML Editor plugin is the only XML Editor to provide a side by side source and diagram presentation and have them real-time synchronized:

- the changes you make in the Editor are immediately visible in the Diagram (no background parsing).
- changing the selected element in the diagram, selects the underlaying code in the source editor.

Full Model View

When you create a schema document or open an existing one, the editor panel is divided in two sections: one containing the schema diagram and the second the source code. The diagram view has two tabbed panes offering a **Full Model View** and a **Logical Model View**.



```
       <a:documentation>Skip content inside XForms instance
element.</a:documentation>
       <mode>
        <anyNamespace>
         <unwrap/>
        </anyNamespace></mode>
      </context>
     </unwrap>
    </namespace>
    <namespace ns="http://www.xsmiles.org/*">
```

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 Editor plugin NVDL editor of is:

- **contextual menu of current editor** > **Show Definition** - Moves the cursor to its definition in the schema used by NVDL in order to validate it.

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.

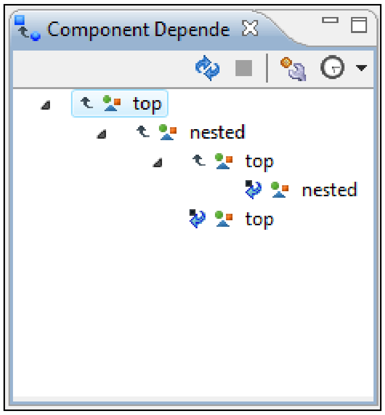


Figure 80: Component Dependencies View - Hierarchy for `test.nvdl`

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

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

The following actions are available in the contextual menu:

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

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

Linking Between Development and Authoring

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

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

There are two validators configured by default:

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

Associate a Validation Scenario

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 Editor 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 Editor 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 Editor plugin warns you if the current module is not part of the dependencies graph computed for the main stylesheet. In this case, it considers the current module as the main stylesheet.

The advantages of editing in the context of main file include:

- correct validation of a module in the context of a larger stylesheet structure;
- content completion assistant displays all components valid in the current context;
- the **Outline** displays the components collected from the entire stylesheet structure.

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:

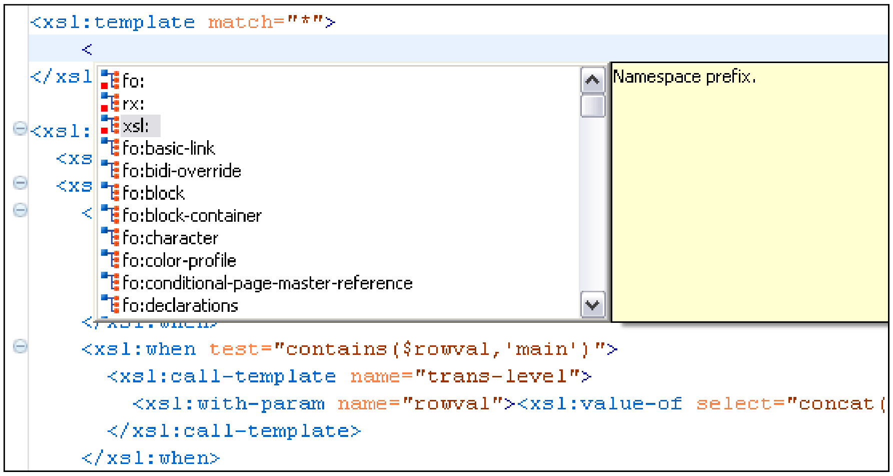
- template modes
- template name
- variable names
- parameter names

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 Editor plugin provides an easy mode to map them by proposing a prefix for these namespaces.



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

<xsl:
  <xs
  <xs
    <
    </xsl:when>
    <xsl:when test="contains($rowval,'main')">
      <xsl:call-template name="trans-level">
        <xsl:with-param name="rowval"><xsl:value-of select="concat(
      </xsl:call-template>
    </xsl:when>
```

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).

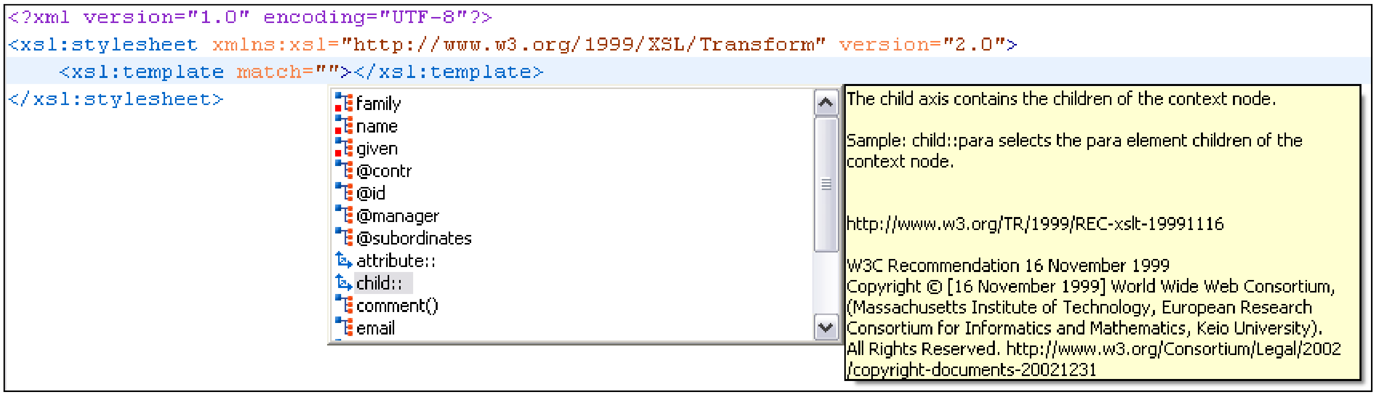
For example, if the document associated with the edited stylesheet is:

```
<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.



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

The child axis contains the children of the context node.

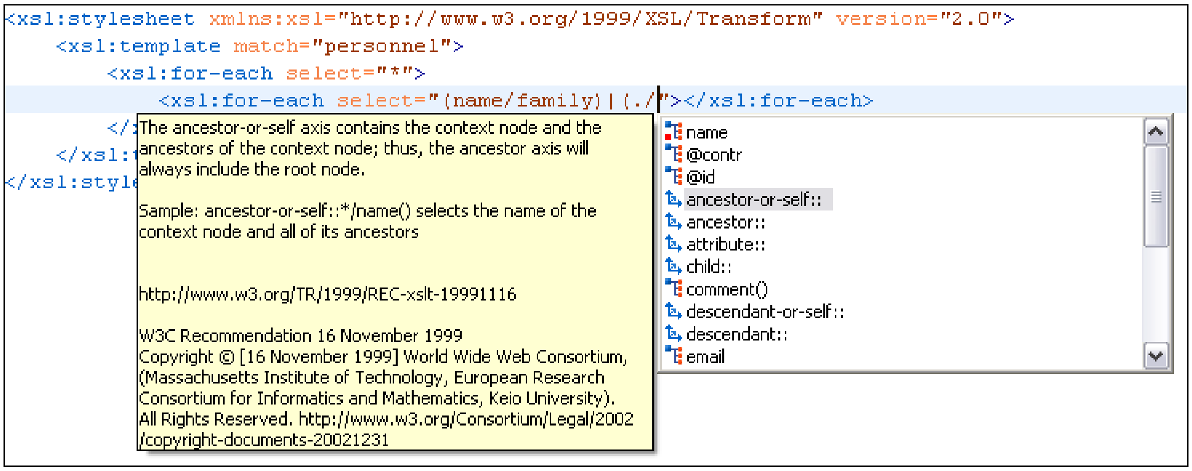
Sample: child::para selects the para element children of the context node.

http://www.w3.org/TR/1999/REC-xslt-19991116

W3C Recommendation 16 November 1999
Copyright © [16 November 1999] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

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:



```
<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 select="(name/family)|(./"></xsl:for-each>
        </
    </xsl:
</xsl:styl
```

The ancestor-or-self axis contains the context node and the ancestors of the context node; thus, the ancestor axis will always include the root node.

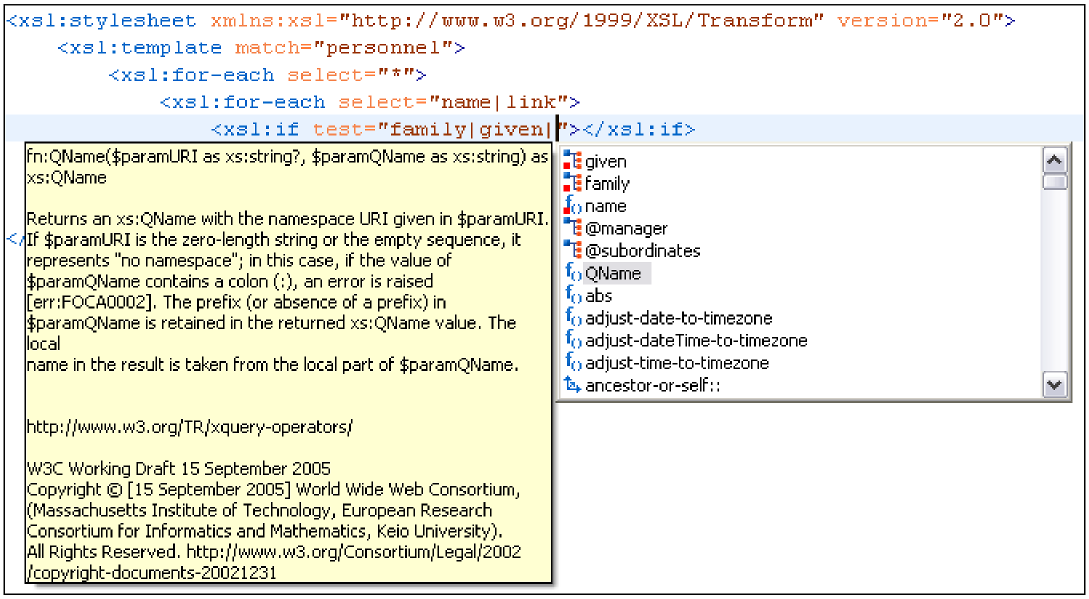
Sample: ancestor-or-self::*/name() selects the name of the context node and all of its ancestors

http://www.w3.org/TR/1999/REC-xslt-19991116

W3C Recommendation 16 November 1999
Copyright © [16 November 1999] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

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.



```
<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 select="name|link">
                <xsl:if test="family|given|"></xsl:if>
</
```

fn:QName($paramURI as xs:string?, $paramQName as xs:string) as xs:QName

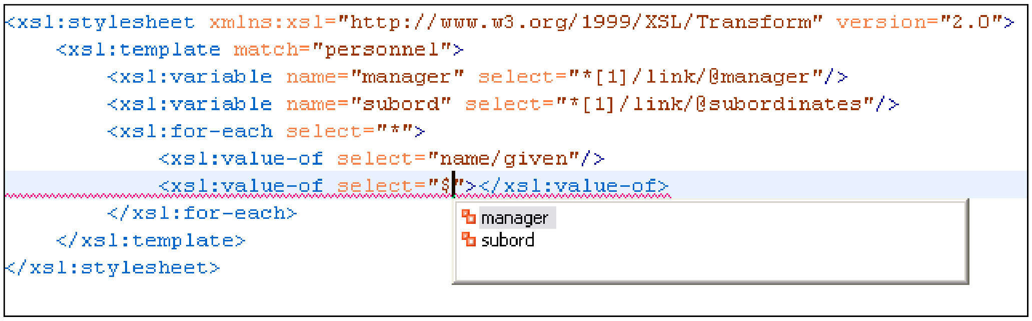
Returns an xs:QName with the namespace URI given in $paramURI. If $paramURI is the zero-length string or the empty sequence, it represents "no namespace"; in this case, if the value of $paramQName contains a colon (:), an error is raised [err:FOCA0002]. The prefix (or absence of a prefix) in $paramQName is retained in the returned xs:QName value. The local
name in the result is taken from the local part of $paramQName.

http://www.w3.org/TR/xquery-operators/

W3C Working Draft 15 September 2005
Copyright © [15 September 2005] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

Figure 84: Content Completion in the `test` Attribute

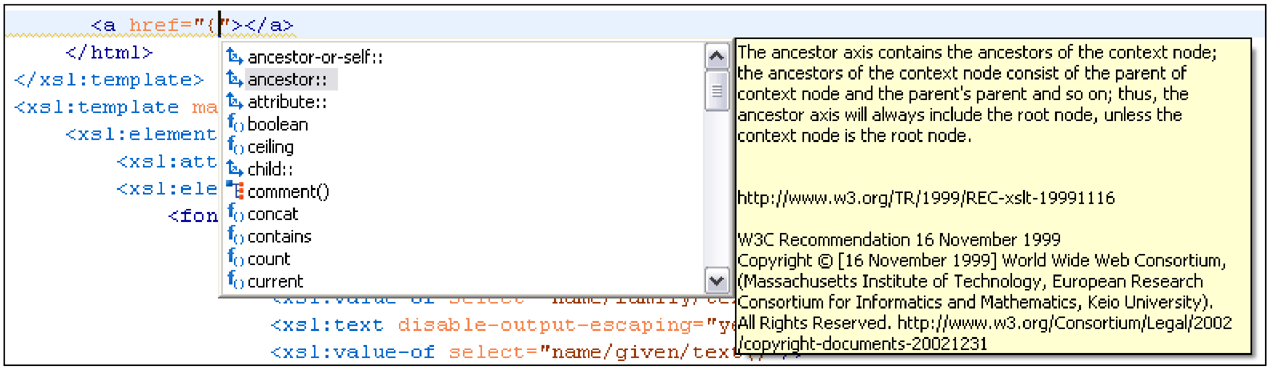
XSLT variable references are easier to insert in XPath expressions with the help of the content completion popup triggered by the $ character which signals the start of such a reference in an XPath expression.



```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:template match="personnel">
        <xsl:variable name="manager" select="*[1]/link/@manager"/>
        <xsl:variable name="subord" select="*[1]/link/@subordinates"/>
        <xsl:for-each select="*">
            <xsl:value-of select="name/given"/>
            <xsl:value-of select="$"></xsl:value-of>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

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.



```
        <a href="{"></a>
      </html>
</xsl:template>
<xsl:template ma
      <xsl:element
            <xsl:att
            <xsl:ele
                  <fon
                        <xsl:text disable-output-escaping="y
                        <xsl:value-of select="name/given/tex
```

The ancestor axis contains the ancestors of the context node; the ancestors of the context node consist of the parent of context node and the parent's parent and so on; thus, the ancestor axis will always include the root node, unless the context node is the root node.

http://www.w3.org/TR/1999/REC-xslt-19991116

W3C Recommendation 16 November 1999
Copyright © [16 November 1999] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

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 Editor plugin tracks the current entered argument by displaying a tooltip containing the function signature. The currently edited argument is highlighted with a bolder font.

When moving the caret through the expression, the tooltip is updated to reflect the argument found at the caret position.

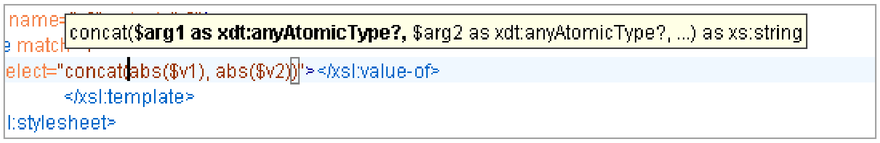
We want to concatenate the absolute values of two variables, named *v1* and *v2*.

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

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

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

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

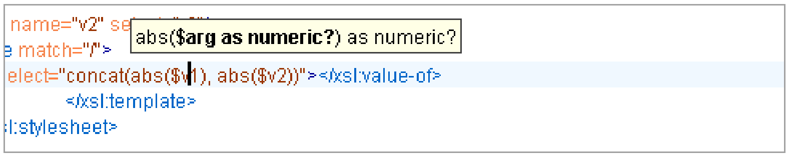


```
name=
e match
elect="concat(abs($v1), abs($v2))"></xsl:value-of>
      </xsl:template>
l:stylesheet>
```

concat(**$arg1 as xdt:anyAtomicType?,** $arg2 as xdt:anyAtomicType?, ...) as xs:string

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

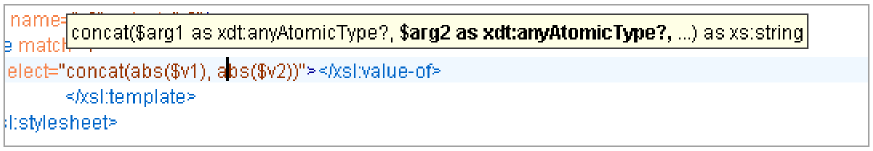
Moving the caret on the first variable `$v1`, the editor identifies the `abs` as context function and shows its signature:



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

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.



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

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 Editor plugin comes with a large set of ready-to use templates for XSL and XML Schema documents.

The XSL code template called Template-Match-Mode

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

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

</xsl:template>
```

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

The XSLT/XQuery Input View

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

The XSLT Input View

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

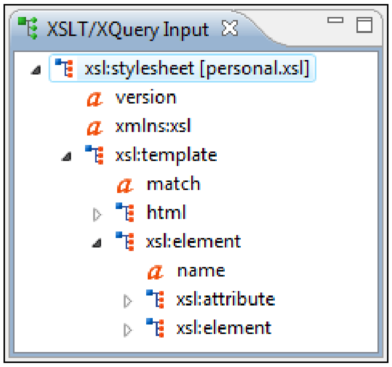


Figure 90: XSLT Input View

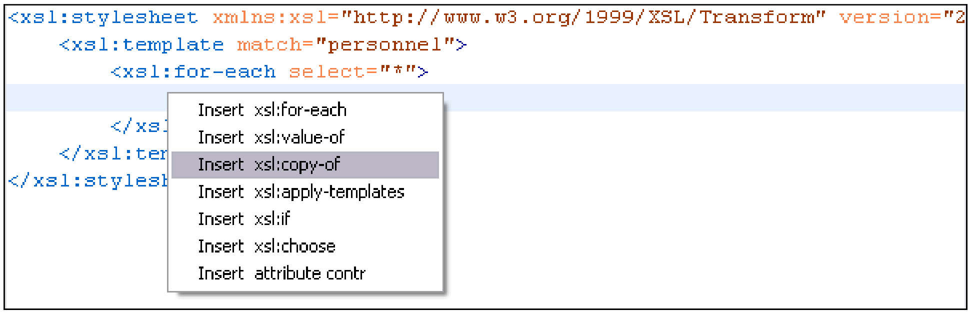
For example, for the following XML document:

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

and the following XSLT stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

        version="2.0">
    <xsl:template match="personnel">
        <xsl:for-each select="*">

        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

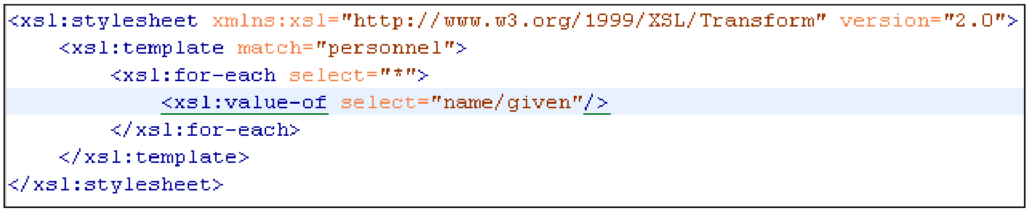
if you drag the `given` element and drop it inside the `xsl:for-each` element, the following popup menu is displayed:



```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2
    <xsl:template match="personnel">
        <xsl:for-each select="*">

        </xs
    </xsl:ter
</xsl:stylesh
```

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**.

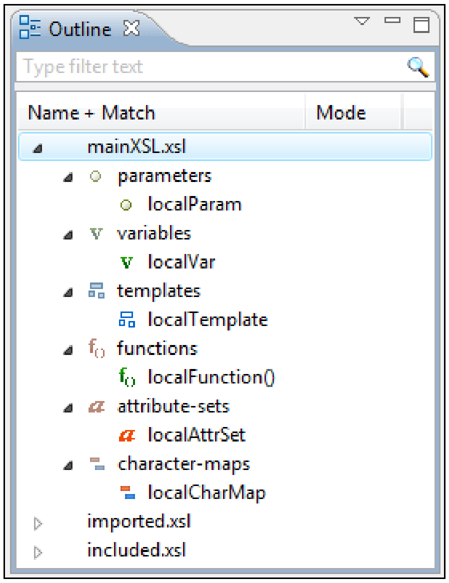


Figure 93: The XSLT Outline View

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 de component name, match or mode in the text field, you can switch to the tree structure using:

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

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

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

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

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

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

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

XSLT Stylesheet Documentation Support

Oxygen XML Editor plugin offers built-in support for documenting XSLT stylesheets. If the expanded *QName* of the element has a non-null namespace URI, the `xsl:stylesheet` element may contain any element not from the XSLT namespace. Such elements are referred to as user-defined data elements. Such elements can contain the documentation for the stylesheet and its elements (top-level elements whose names are in the XSLT namespace). Oxygen XML Editor plugin offers its own XML schema that defines such documentation elements. The schema is named `stylesheet_documentation.xsd` and can be found in `[oXygen-install-folder]/frameworks/stylesheet_documentation`. The user can also specify a custom schema in *XSL Content Completion options*.

When content completion is invoked inside an XSLT editor by pressing **(CTRL+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.

In **Author** mode, the following stylesheet documentation actions are available in the contextual menu, **Component Documentation** submenu:

- **Add component documentation** - Adds documentation blocks for the component at caret position.
- **Paragraph** - Inserts a new documentation paragraph.
- **Bold** - Makes the selected documentation text bold.
- **Italic** - Makes the selected documentation text italic.
- **List** - Inserts a new list.
- **List Item** - Inserts a list item.
- **Reference** - Inserts a documentation reference.

If the caret is positioned inside the `xsl:stylesheet` element context, documentation blocks are generated for all XSLT elements. If the caret is positioned inside a specific XSLT element (like a template or a function), a documentation block is generated for that element only.

Example of a documentation block using Oxygen XML Editor plugin built-in schema

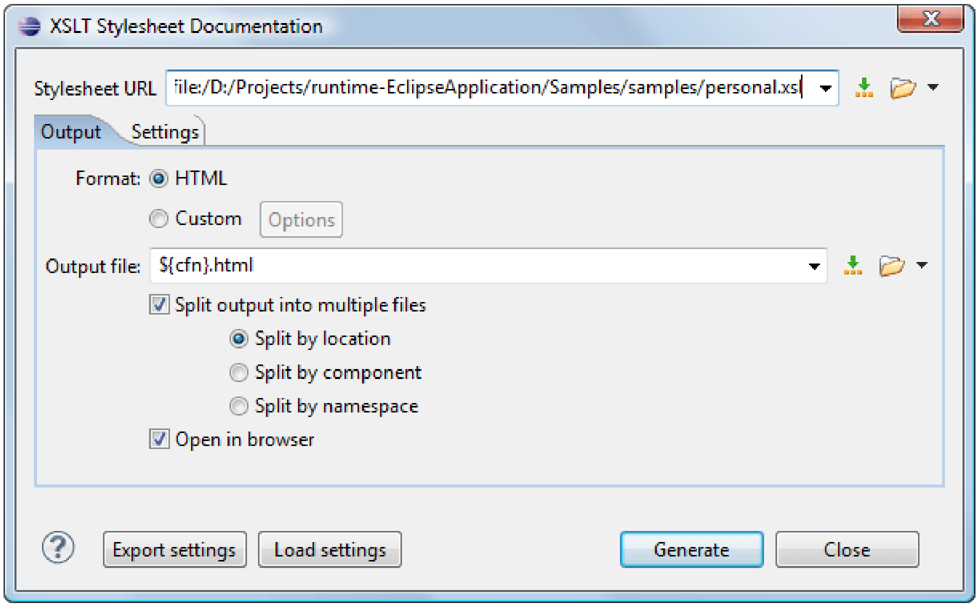
```
<xd:doc>
  <xd:desc>
    <xd:p>Search inside parameter <xd:i>string</xd:i> for the last
 occurrence of parameter
    <xd:i>searched</xd:i>. The substring starting from the 0
position to the identified last
      occurrence will be returned. <xd:ref
name="f:substring-after-last" type="function"
xmlns:f="http://www.oxygenxml.com/doc/xsl/functions">See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

```
    returned.</xd:p>
  </xd:return>
</xd:doc>
```

Generating Documentation for an XSLT Stylesheet

Oxygen XML Editor 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.



Stylesheet URL: file:/D:/Projects/runtime-EclipseApplication/Samples/samples/personal.xs

Format: ● HTML
○ Custom

Output file: ${cfn}.html

☑ Split output into multiple files
● Split by location
○ Split by component
○ Split by namespace

☑ Open in browser

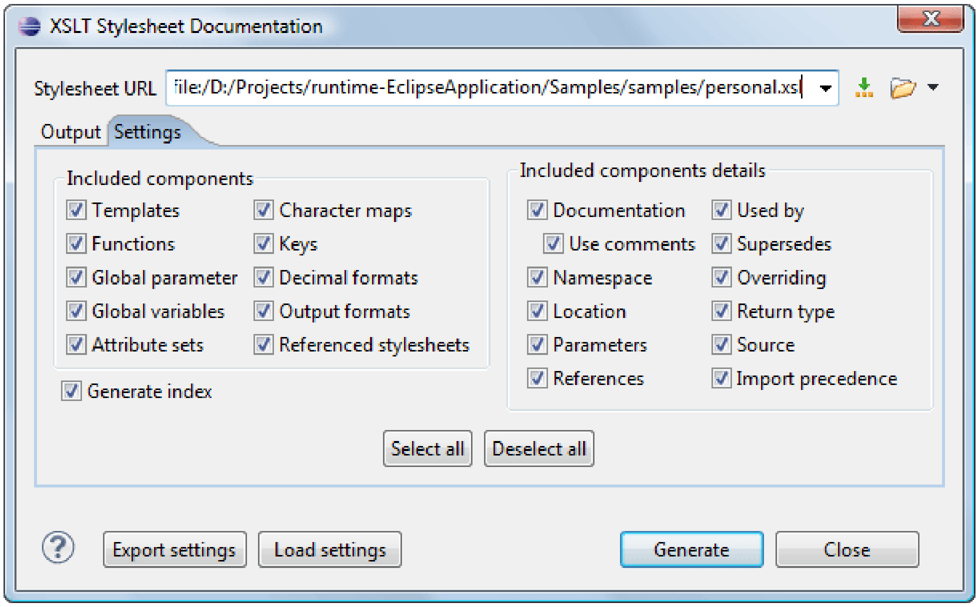
Figure 94: The Output Panel of the XSLT Stylesheet Documentation Dialog

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.



Stylesheet URL file:/D:/Projects/runtime-EclipseApplication/Samples/samples/personal.xsl

Included components

☑ Templates
☑ Functions
☑ Global parameter
☑ Global variables
☑ Attribute sets
☑ Character maps
☑ Keys
☑ Decimal formats
☑ Output formats
☑ Referenced stylesheets

☑ Generate index

Included components details

☑ Documentation
☑ Use comments
☑ Namespace
☑ Location
☑ Parameters
☑ References
☑ Used by
☑ Supersedes
☑ Overriding
☑ Return type
☑ Source
☑ Import precedence

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 Editor plugin built-in XSLT documentation schema.
  - A subset of Docbook 5 elements. The recognized elements are: `section`, `sect1` to `sect5`, `emphasis`, `title`, `ulink`, `programlisting`, `para`, `orderedlist`, `itemizedlist`;
  - A subset of DITA elements. The recognized elements are: `concept`, `topic`, `task`, `codeblock`, `p`, `b`, `i`, `ul`, `ol`, `pre`, `sl`, `sli`, `step`, `steps`, `li`, `title`, `xref`;
  - Full XHTML 1.0 support;
  - XSLStyle documentation environment. XSLStyle uses Docbook or DITA languages inside its own user-defined data elements. The supported Docbook and DITA elements are the ones mentioned above;
  - Doxsl documentation framework. Supported elements are : `codefrag`, `description`, `para`, `docContent`, `documentation`, `parameter`, `function`, `docSchema`, `link`, `list`, `listitem`, `module`, `parameter`, `template`, `attribute-set`;

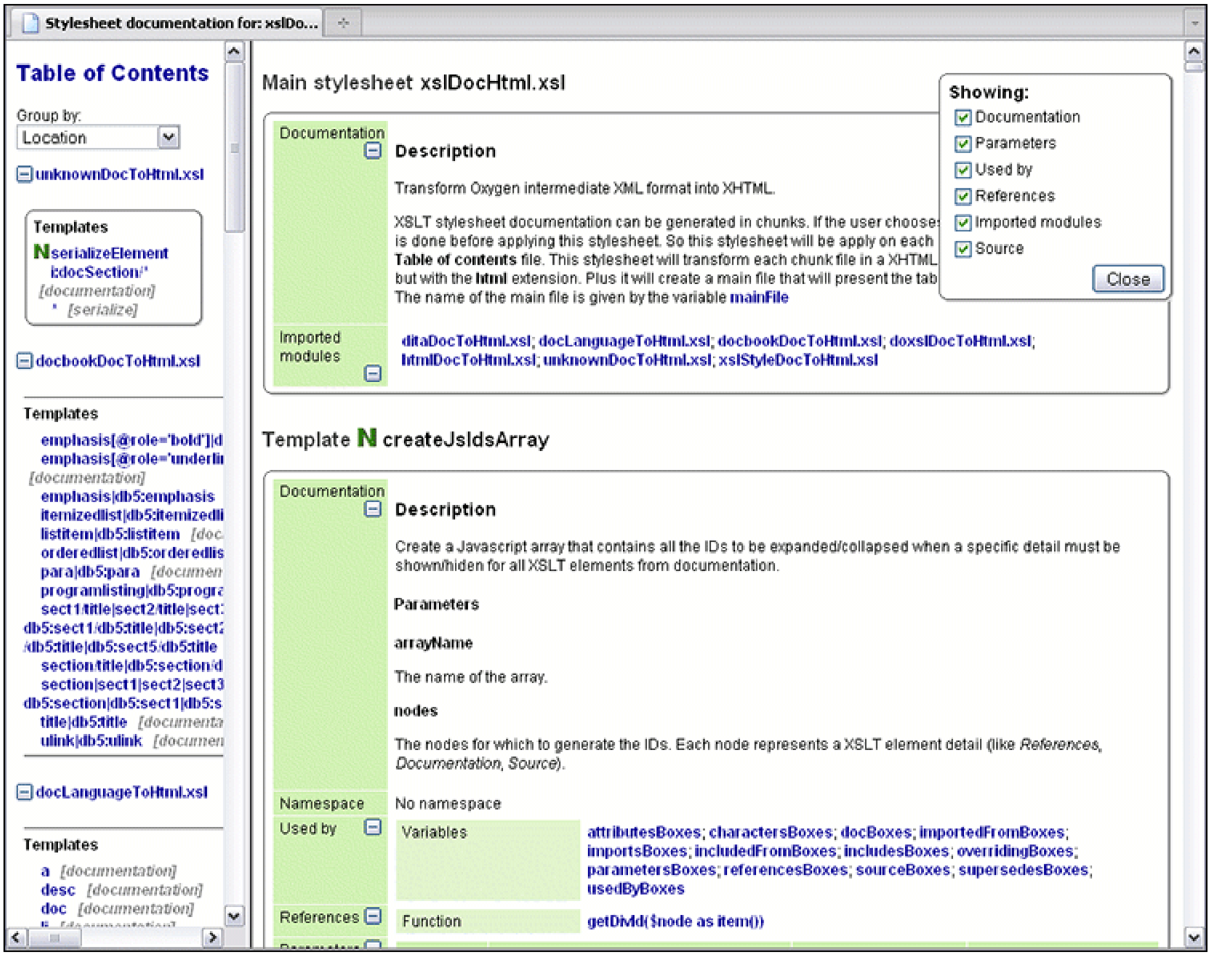
  Other XSLT documentation blocks that are not recognized will just be serialized inside an HTML `pre` element. You can change this behavior by using a *custom format* instead of the built-in *HTML format* and providing your own XSLT stylesheets.

- **Use comments** - Controls whether the comments that precede an XSLT element is treated as documentation for the element they precede. Comments that precede or succeed the `xsl:stylesheet` element, are treated as documentation for the whole stylesheet. Please note that comments that precede an import or include directive are not collected as documentation for the imported/included module. Also comments from within the body of the XSLT elements are not collected at all.
- **Namespace** - Shows the namespace for named XSLT elements.
- **Location** - Shows the stylesheet location for each XSLT element.
- **Parameters** - Shows parameters of templates and functions.
- **References** - Shows the named XSLT elements that are referred from within an element.
- **Used by** - Shows the list of all the XSLT elements that refer the current named element.
- **Supersedes** - Shows the list of all the XSLT elements that are superseded the current element.

- **Overriding** - Shows the list of all the XSLT elements that override the current element.
- **Return type** - Shows the return type of the function.
- **Source** - Shows the text stylesheet source for each XSLT element.
- **Import precedence** - Shows the computed import precedence as declared in XSL transformation specifications.
- **Generate index** - Creates an index with all the XSLT elements included in the documentation.
- **Load settings / Export settings** - The current settings can be saved for further usage (for example for generating documentation from command-line interface) with the **Export settings** button, and reloaded when necessary with the **Load settings** button.

Generate Documentation in HTML Format

The generated documentation looks like:



unknownDocToHtml.xsl

Templates
N serializeElement
k:docSection/*
[documentation]
* [serialize]

docbookDocToHtml.xsl

Templates
emphasis[@role='bold']|d
emphasis[@role='underli
[documentation]
emphasis|db5:emphasis
itemizedlist|db5:itemizedli
listitem|db5:listitem [doc
orderedlist|db5:orderedlis
para|db5:para [documen
programlisting|db5:progra
sect1/title|sect2/title|sect
db5:sect1/db5:title|db5:sect
/db5:title|db5:sect5/db5:title
section/title|db5:section/d
section|sect1|sect2|sect3
db5:section|db5:sect1|db5:s
title|db5:title [documenta
ulink|db5:ulink [documen

docLanguageToHtml.xsl

Templates
a [documentation]
desc [documentation]
doc [documentation]

Main stylesheet xslDocHtml.xsl

Documentation

Description

Transform Oxygen intermediate XML format into XHTML.

XSLT stylesheet documentation can be generated in chunks. If the user choose
is done before applying this stylesheet. So this stylesheet will be apply on each
Table of contents file. This stylesheet will transform each chunk file in a XHTML
but with the **html** extension. Plus it will create a main file that will present the tab
The name of the main file is given by the variable mainFile

Imported modules: ditaDocToHtml.xsl; docLanguageToHtml.xsl; docbookDocToHtml.xsl; doxslDocToHtml.xsl; htmlDocToHtml.xsl; unknownDocToHtml.xsl; xslStyleDocToHtml.xsl

Template N createJsIdsArray

Documentation

Description

Create a Javascript array that contains all the IDs to be expanded/collapsed when a specific detail must be shown/hiden for all XSLT elements from documentation.

Parameters

arrayName

The name of the array.

nodes

The nodes for which to generate the IDs. Each node represents a XSLT element detail (like *References*, *Documentation*, *Source*).

Namespace: No namespace

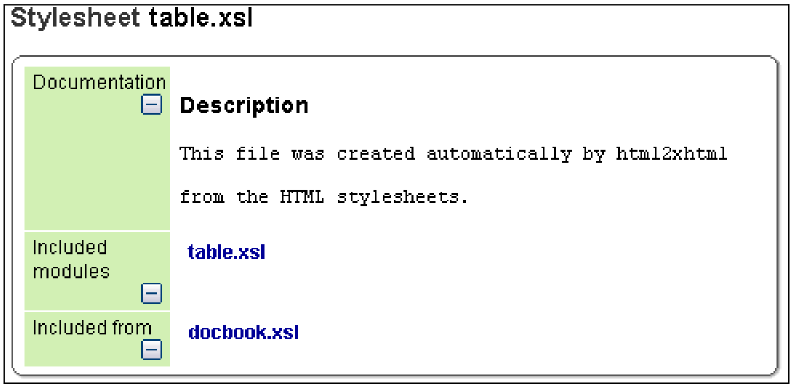
Used by: Variables: attributesBoxes; charactersBoxes; docBoxes; importedFromBoxes; importsBoxes; includedFromBoxes; includesBoxes; overridingBoxes; parametersBoxes; referencesBoxes; sourceBoxes; supersedesBoxes; usedByBoxes

References: Function: getDivId($node as item())

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.



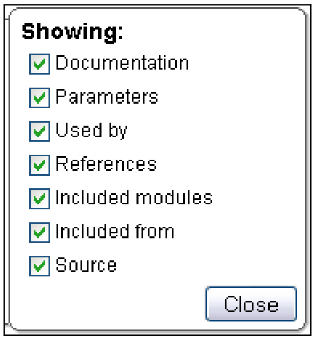
Stylesheet table.xsl

Documentation	**Description** This file was created automatically by html2xhtml from the HTML stylesheets.
Included modules	table.xsl
Included from	docbook.xsl

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.

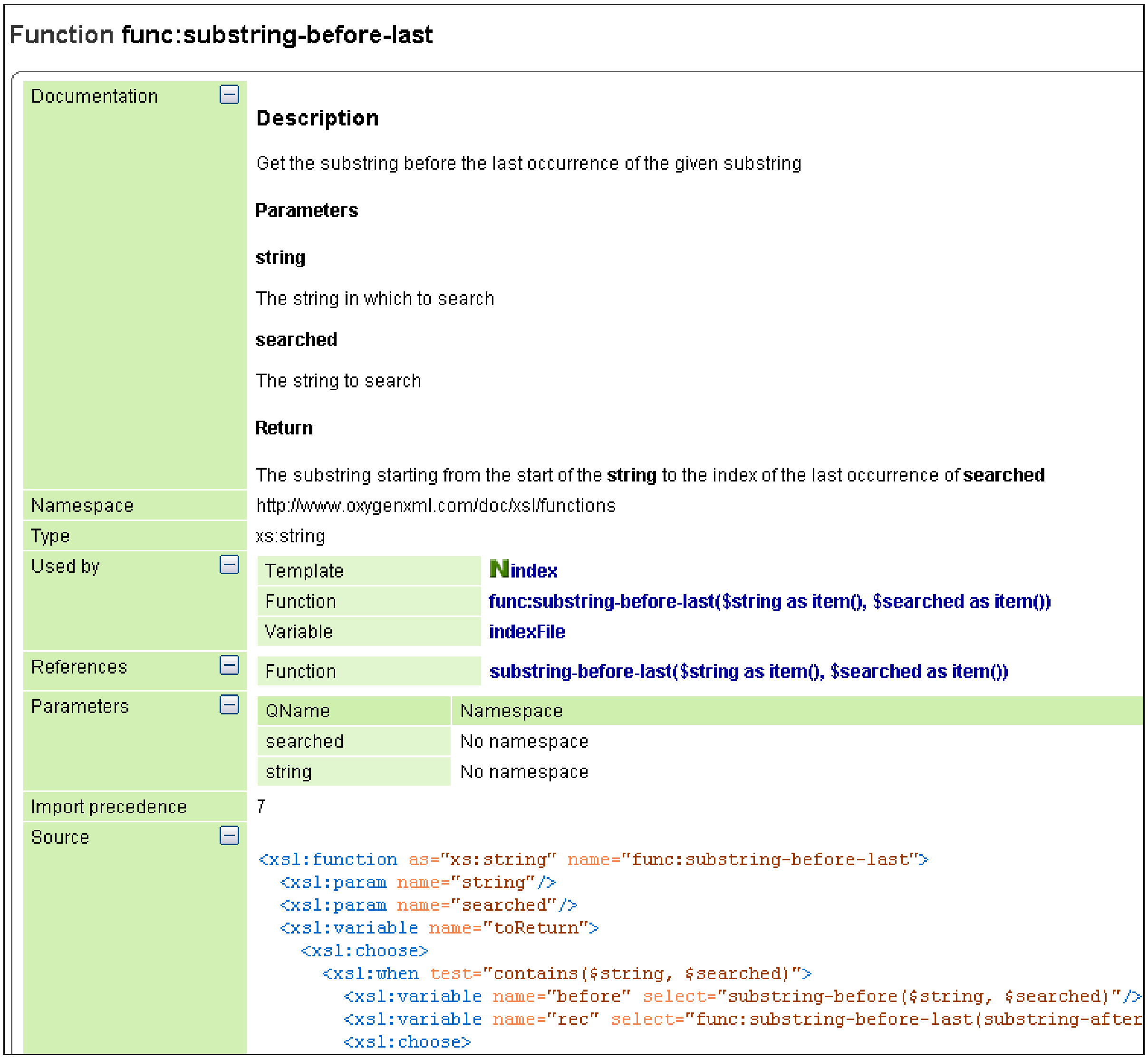


Showing:

- ☑ Documentation
- ☑ Parameters
- ☑ Used by
- ☑ References
- ☑ Included modules
- ☑ Included from
- ☑ Source

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

Description

Get the substring before the last occurrence of the given substring

Parameters

string

The string in which to search

searched

The string to search

Return

The substring starting from the start of the **string** to the index of the last occurrence of **searched**

Namespace	http://www.oxygenxml.com/doc/xsl/functions
Type	xs:string

Used by

Template	**N** index
Function	func:substring-before-last($string as item(), $searched as item())
Variable	indexFile

References

Function	substring-before-last($string as item(), $searched as item())

Parameters

QName	Namespace
searched	No namespace
string	No namespace

Import precedence	7

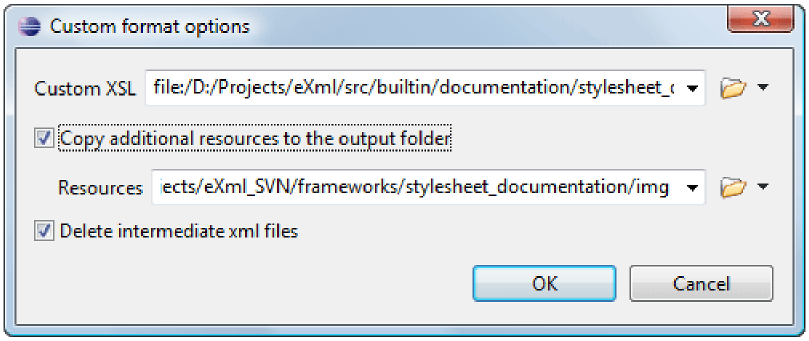
Source

```
<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>
```

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`.



Custom XSL file:/D:/Projects/eXml/src/builtin/documentation/stylesheet_c

☑ Copy additional resources to the output folder

Resources iects/eXml_SVN/frameworks/stylesheet_documentation/img

☑ Delete intermediate xml files

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

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

Example of an XML Configuration File

```
<serialized>
    <map>
        <entry>
            <String
xml:space="preserve">xsd.documentation.options</String>
            <xsdDocumentationOptions>
                <field name="outputFile">
                    <String
xml:space="preserve">${cfn}.html</String>
                </field>
                <field name="splitMethod">
                    <Integer xml:space="preserve">1</Integer>
                </field>
                <field name="openOutputInBrowser">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="format">
                    <Integer xml:space="preserve">1</Integer>
                </field>
                <field name="customXSL">
                    <null/>
                </field>
                <field name="deleteXMLFiles">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeIndex">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeGlobalElements">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <field name="includeGlobalAttributes">
                    <Boolean xml:space="preserve">true</Boolean>
                </field>
                <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 Editor plugin performs a search over the entire document to find the component declaration and all its references. When found, they are highlighted both in the document and in the stripe bar, at the right side of the document. Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.

This feature is set on automatic search by default and can be configured in the **Options** > **Preferences** > **Editor** > **Mark Occurrences** page. A search can also be triggered with the **Search** > **Search Occurrences in File** **(Ctrl+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:

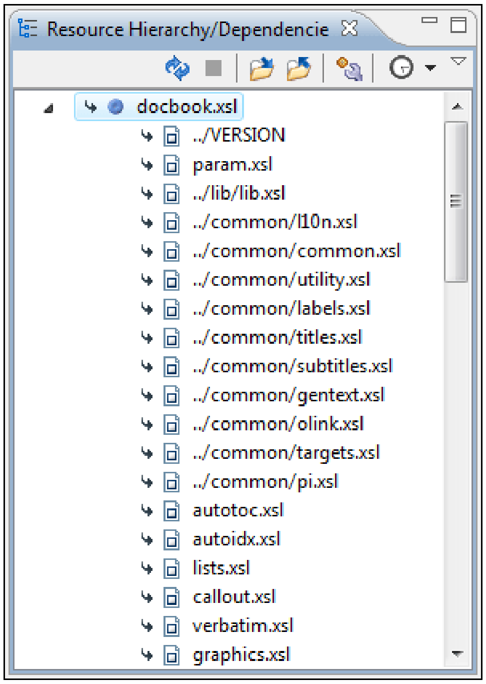
  ```
  <person>
      <xsl:attribute name="id">
          <xsl:text>Big</xsl:text>
          <xsl:value-of select="test"/>
          <xsl:text>Boss</xsl:text>
      </xsl:attribute>
  </person>
  ```

- **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**.

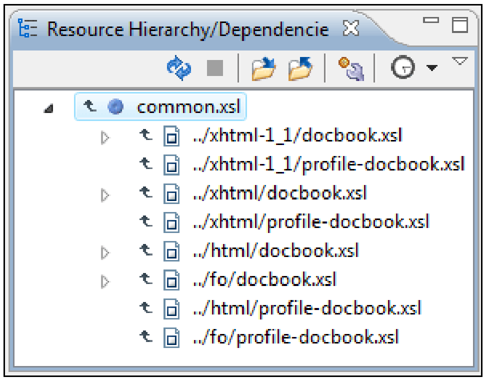
If you want to see the hierarchy of a stylesheet, select the desired stylesheet in the project view and choose **Resource Hierarchy** from the contextual menu.



docbook.xsl
../VERSION
param.xsl
../lib/lib.xsl
../common/l10n.xsl
../common/common.xsl
../common/utility.xsl
../common/labels.xsl
../common/titles.xsl
../common/subtitles.xsl
../common/gentext.xsl
../common/olink.xsl
../common/targets.xsl
../common/pi.xsl
autotoc.xsl
autoidx.xsl
lists.xsl
callout.xsl
verbatim.xsl
graphics.xsl

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.



common.xsl
../xhtml-1_1/docbook.xsl
../xhtml-1_1/profile-docbook.xsl
../xhtml/docbook.xsl
../xhtml/profile-docbook.xsl
../html/docbook.xsl
../fo/docbook.xsl
../html/profile-docbook.xsl
../fo/profile-docbook.xsl

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:

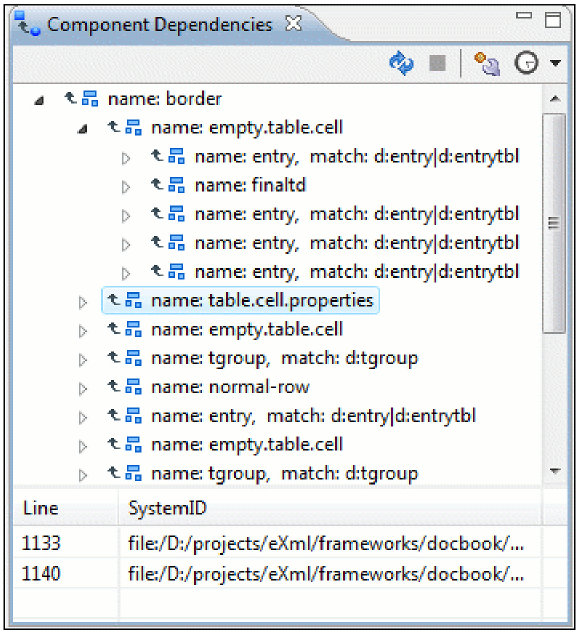
- **Open** - Opens the stylesheet. Alternatively, you can open the stylesheet by a double-click on the Hierarchy/Dependencies structure.
- **Copy location** - Copies the location of the stylesheet.
- **Show Resource Hierarchy** - Shows the hierarchy for the selected stylesheet.

- **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).



Component Dependencies

- name: border
  - name: empty.table.cell
    - name: entry, match: d:entry|d:entrytbl
    - name: finaltd
    - name: entry, match: d:entry|d:entrytbl
    - name: entry, match: d:entry|d:entrytbl
    - name: entry, match: d:entry|d:entrytbl
  - name: table.cell.properties
  - name: empty.table.cell
  - name: tgroup, match: d:tgroup
  - name: normal-row
  - name: entry, match: d:entry|d:entrytbl
  - name: empty.table.cell
  - name: tgroup, match: d:tgroup

Line	SystemID
1133	file:/D:/projects/eXml/frameworks/docbook/...
1140	file:/D:/projects/eXml/frameworks/docbook/...

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:

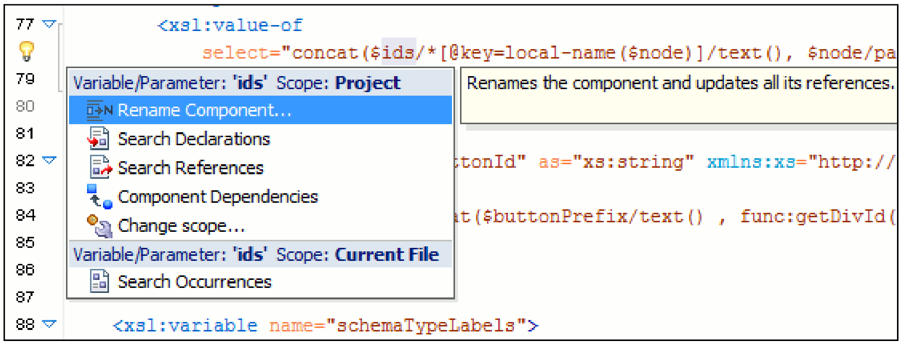
If a component contains multiple references to another, a small table is shown containing all references.

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).



```
        <xsl:value-of
            select="concat($ids/*[@key=local-name($node)]/text(), $node/pa
```

```
:tonId" as="xs:string" xmlns:xs="http://
```

```
at($buttonPrefix/text() , func:getDivId(
```

```
    <xsl:variable name="schemaTypeLabels">
```

Renames the component and updates all its references.

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.

Linking Between Development and Authoring

The Author mode is available for the XSLT editor presenting the stylesheets in a nice visual rendering.

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.

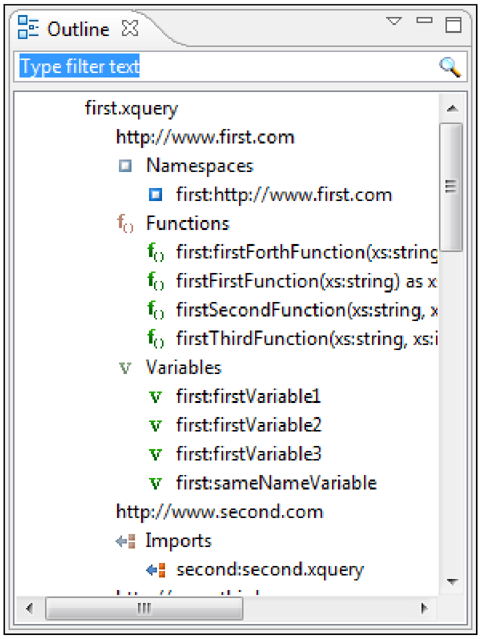


Figure 105: XQuery Outline View

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:

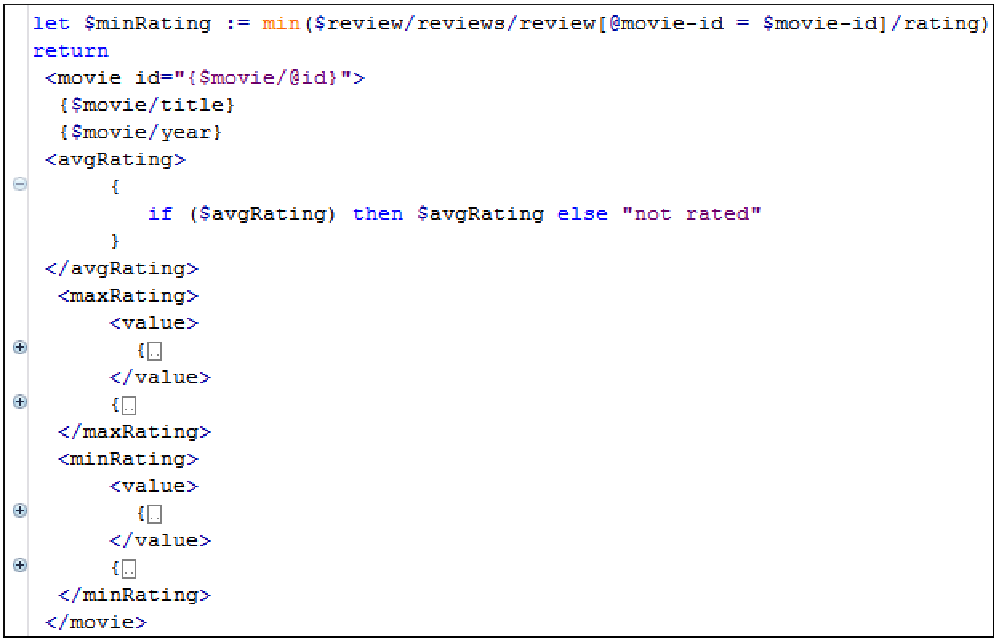
- ***** - any string
- **?** - any character
- **,** - patterns separator

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.



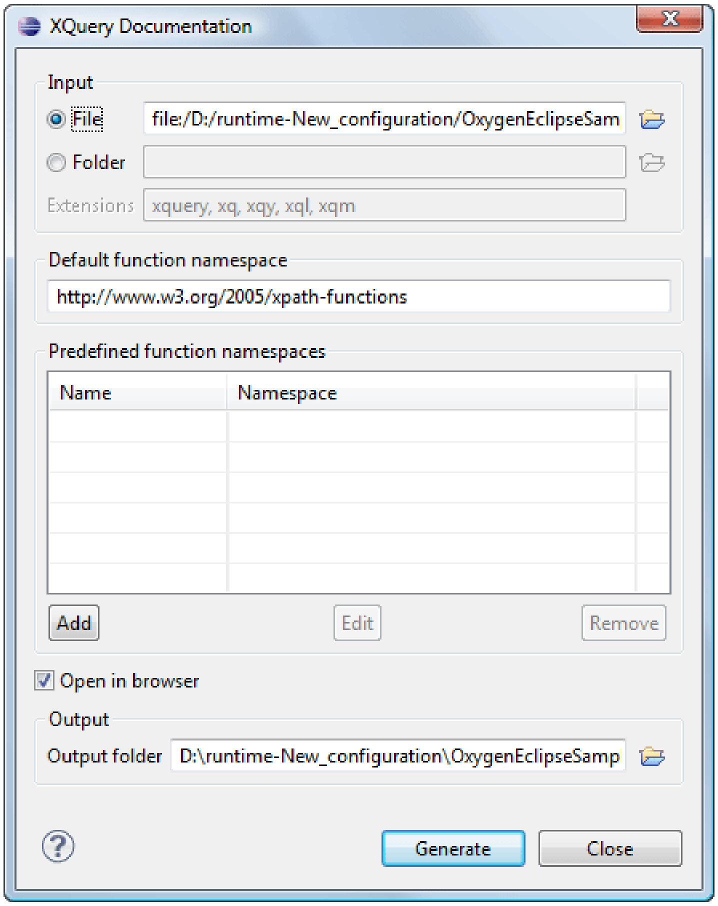
```
let $minRating := min($review/reviews/review[@movie-id = $movie-id]/rating)
return
 <movie id="{$movie/@id}">
  {$movie/title}
  {$movie/year}
 <avgRating>
      {
         if ($avgRating) then $avgRating else "not rated"
      }
 </avgRating>
  <maxRating>
      <value>
        {...
      </value>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

Figure 106: Folding in XQuery Documents

There is available the action **Go to Matching Bracket**<u>**Ctrl+Shift+G**</u> on contextual menu of XQuery editor for going to matching character when cursor is located at '{' character or '}' character. It helps for finding quickly matching character of current folding element.

Generating HTML Documentation for an XQuery Document

To generate HTML documentation for an XQuery document, use the dialog **XQuery Documentation**. It is opened with the action **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:



Input

● File file:/D:/runtime-New_configuration/OxygenEclipseSam

○ Folder

Extensions xquery, xq, xqy, xql, xqm

Default function namespace

http://www.w3.org/2005/xpath-functions

Predefined function namespaces

Name	Namespace

☑ Open in browser

Output

Output folder D:\runtime-New_configuration\OxygenEclipseSamp

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

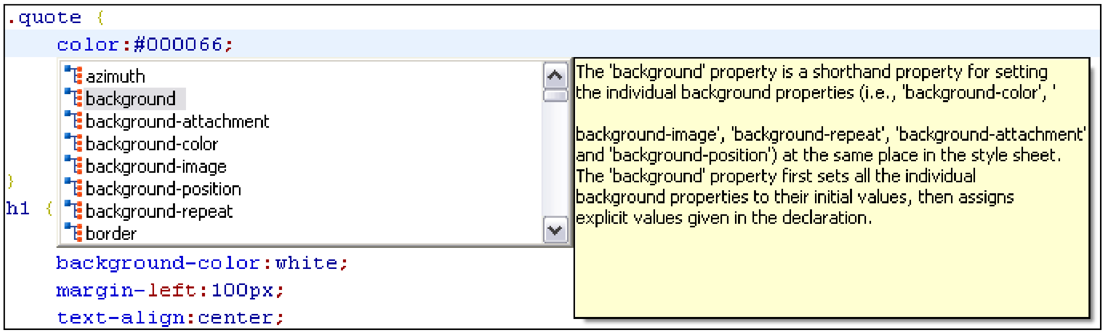
The CSS properties accepted by the validator are the ones included in the current CSS profile that is selected in *the CSS validation preferences*. The profile **CSS 3 with Oxygen extensions** includes all the CSS 3 standard properties and the *CSS extensions specific for Oxygen* that can be used in *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.



```
.quote {
    color:#000066;
}
h1 {
    background-color:white;
    margin-left:100px;
    text-align:center;
```

The 'background' property is a shorthand property for setting the individual background properties (i.e., 'background-color', '

background-image', 'background-repeat', 'background-attachment' and 'background-position') at the same place in the style sheet. The 'background' property first sets all the individual background properties to their initial values, then assigns explicit values given in the declaration.

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.

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*.

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`:

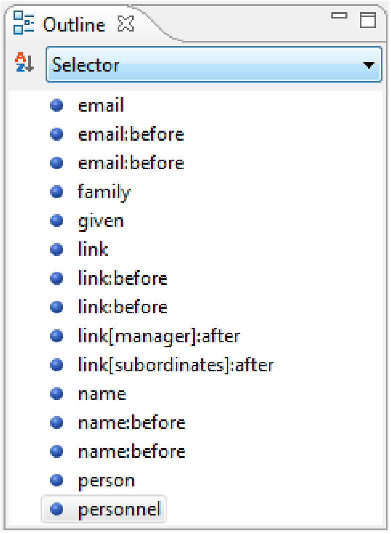
```
<?xml version="1.0" encoding="UTF-8"?>
<css_keywords>
    <property name="custom">
        <summary>Description for custom property.</summary>
        <value name="customValue1"/>
        <value name="customValue2"/>
    </property>
</css_keywords>
```

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.



- email
- email:before
- email:before
- family
- given
- link
- link:before
- link:before
- link[manager]:after
- link[subordinates]:after
- name
- name:before
- name:before
- person
- personnel

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.

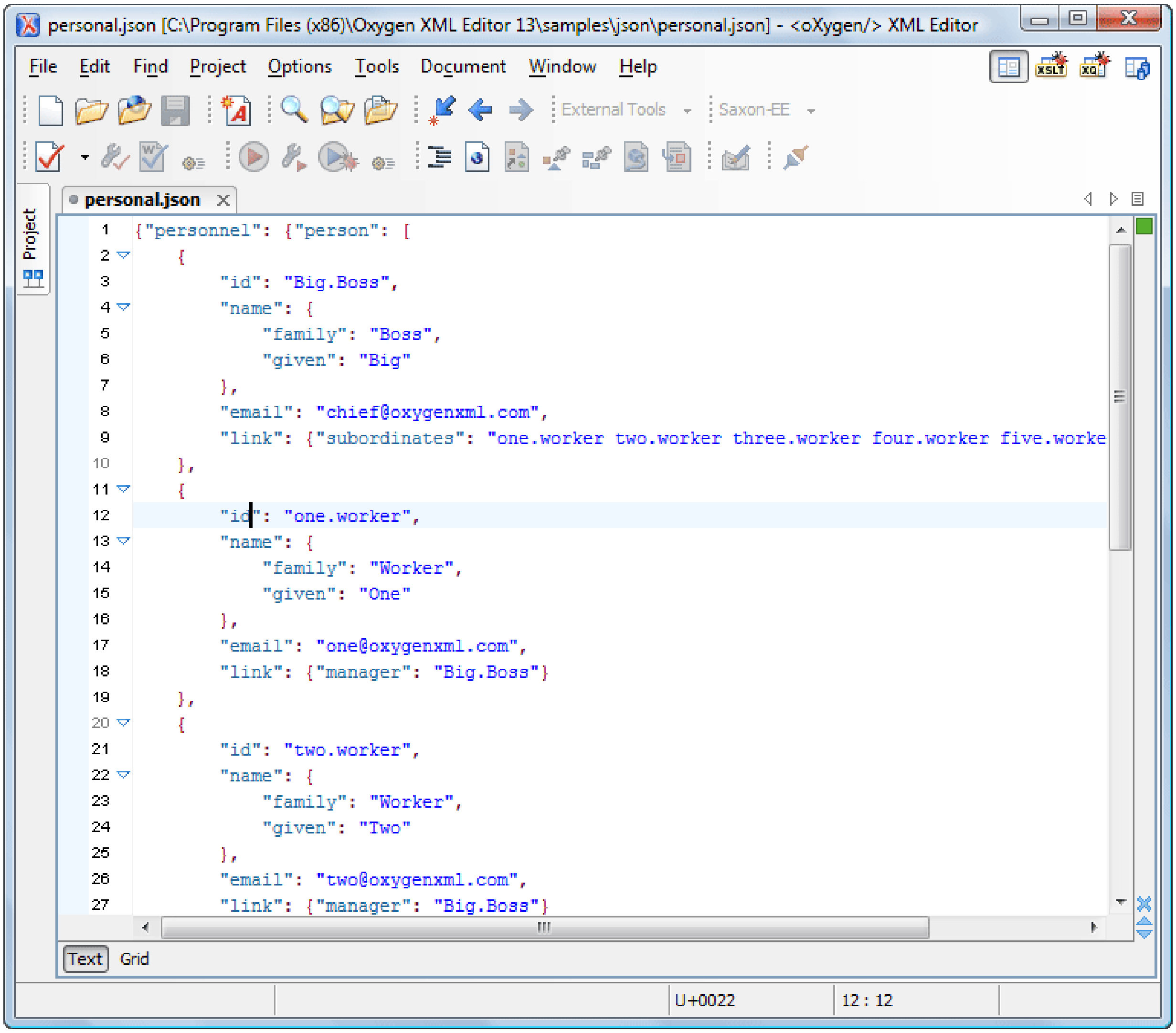
Editing JSON Documents

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** .



```
{"personnel": {"person": [
    {
        "id": "Big.Boss",
        "name": {
            "family": "Boss",
            "given": "Big"
        },
        "email": "chief@oxygenxml.com",
        "link": {"subordinates": "one.worker two.worker three.worker four.worker five.worke
    },
    {
        "id": "one.worker",
        "name": {
            "family": "Worker",
            "given": "One"
        },
        "email": "one@oxygenxml.com",
        "link": {"manager": "Big.Boss"}
    },
    {
        "id": "two.worker",
        "name": {
            "family": "Worker",
            "given": "Two"
        },
        "email": "two@oxygenxml.com",
        "link": {"manager": "Big.Boss"}
```

Figure 110: JSON Editor Text Mode

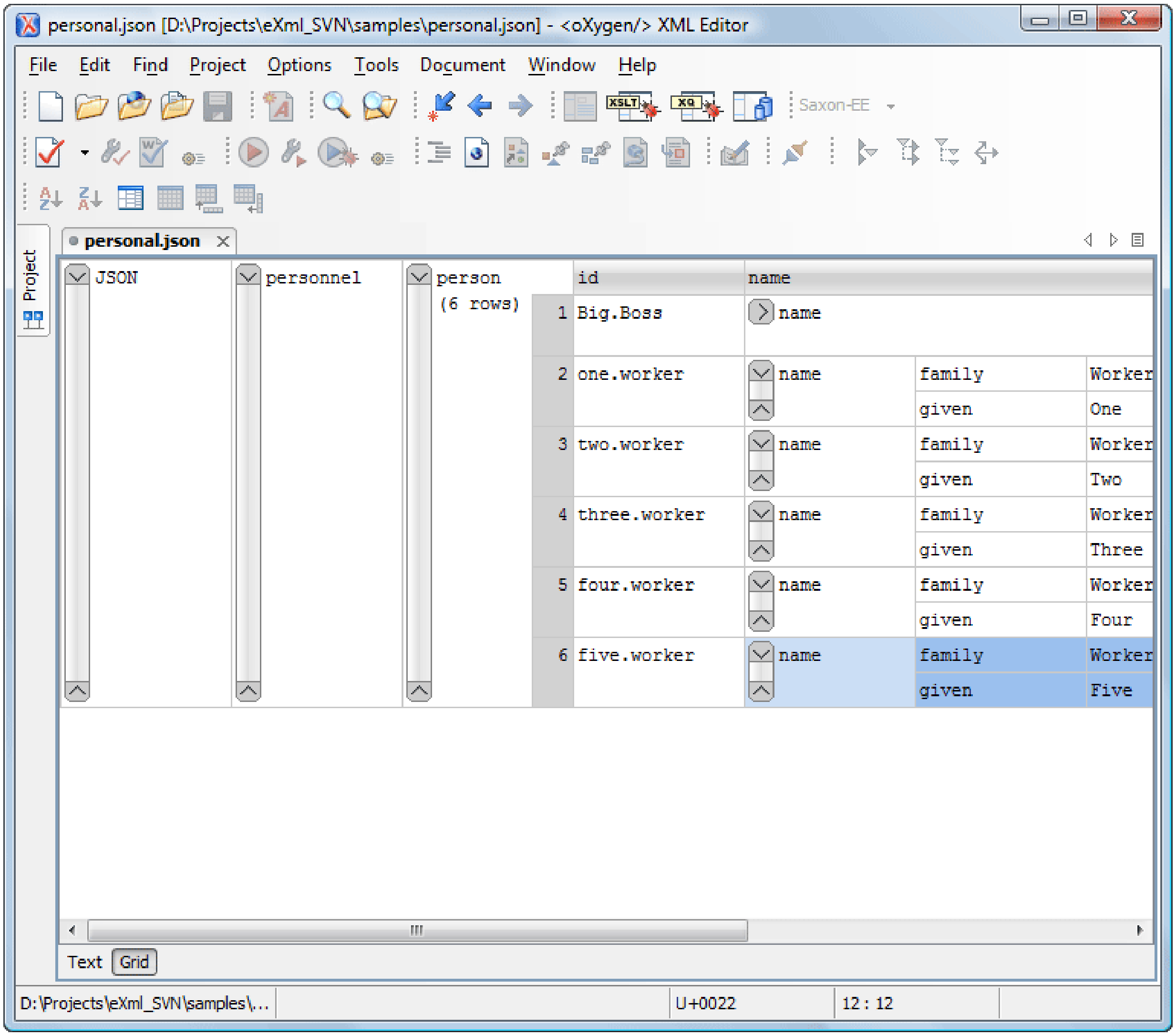
Syntax highlight in JSON Documents

Oxygen XML Editor plugin supports *Syntax Highlight* for JavaScript / JSON editors and provides default configurations for the JSON set of tokens. You can customize the foreground color, background color and the font style for each JSON token type.

Folding in JSON

In a large JSON document, the data enclosed in the '{' and '}' characters can be collapsed so that only the needed data remain in focus. The *folding features available for XML documents* are available in JSON documents.

JSON Editor Grid Mode



JSON
personnel
person (6 rows)

	id	name		
1	Big.Boss	name		
2	one.worker	name	family	Worker
			given	One
3	two.worker	name	family	Worker
			given	Two
4	three.worker	name	family	Worker
			given	Three
5	four.worker	name	family	Worker
			given	Four
6	five.worker	name	family	Worker
			given	Five

D:\Projects\eXml_SVN\samples\... U+0022 12 : 12

Figure 111: JSON Editor Grid Mode

Oxygen XML Editor plugin allows you to view and edit the JSON documents in the *Grid Mode*. The JSON is represented in Grid mode as a compound layout of nested tables in which the JSON data and structure can be easily manipulated with table-specific operations or drag and drop operations on the grid components. You can also use the following JSON-specific contextual actions:

- **Array** - Useful when you want to convert a JSON *value* to *array*.
- **Insert value before** - Inserts a value before the currently selected one.
- **Insert value after** - Inserts a value after the currently selected one.
- **Append value as child** - Appends a value as a child of the currently selected value.

You can *customize the JSON grid appearance* according to your needs. For instance you can change the font, the cell background, foreground, or even the colors from the table header gradients. The default width of the columns can also be changed.

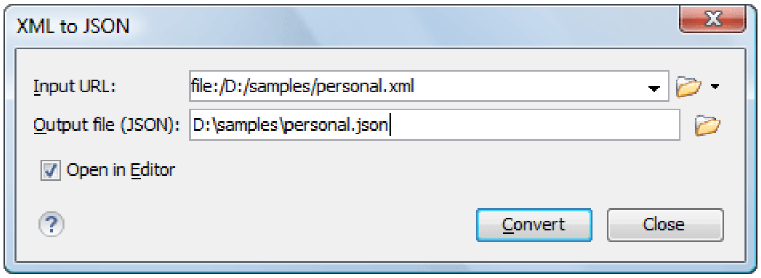
Validating JSON Documents

Oxygen XML Editor plugin includes a built-in JSON validator (based on the free JAVA source code available on www.json.org), integrated with the general validation support.

Convert XML to JSON

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

1. Go to menu **XML Tools** > **XML to JSON...**.
   The **XML to JSON** dialog is displayed:



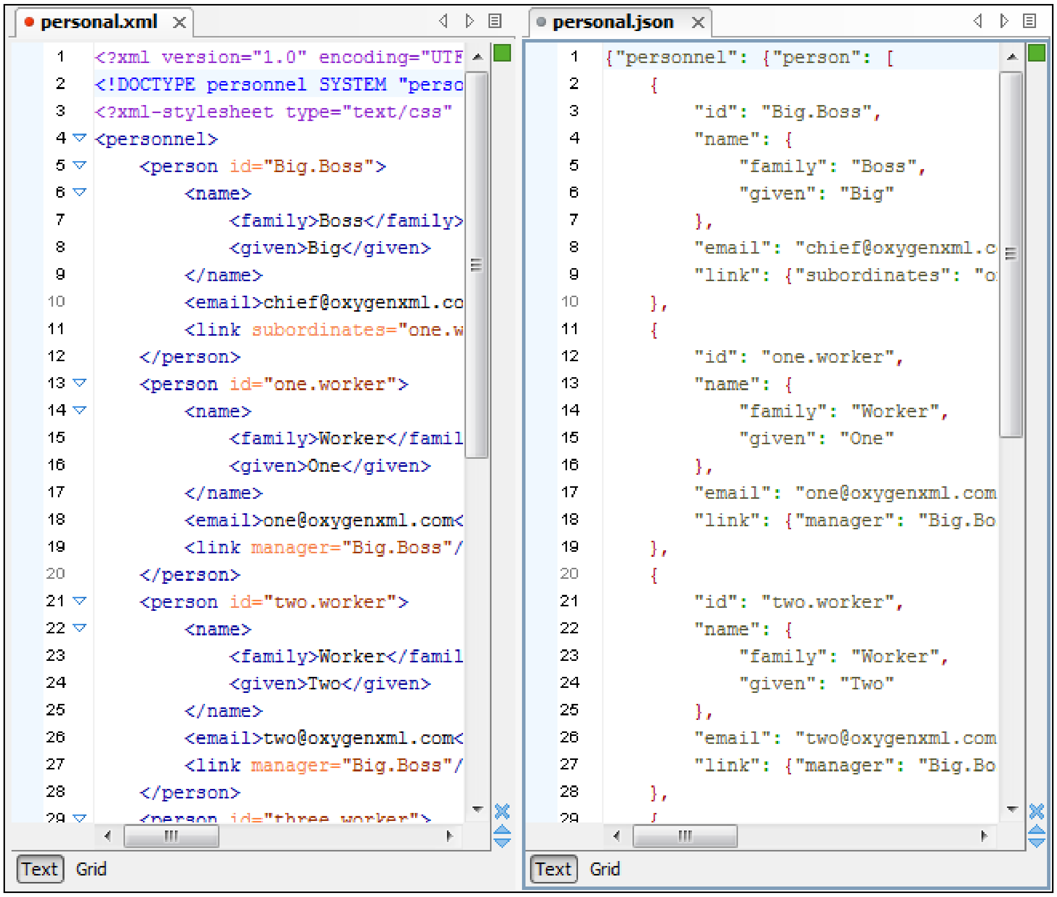
Input URL: file:/D:/samples/personal.xml

Output file (JSON): D:\samples\personal.json

☑ Open in Editor

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.



```
<?xml version="1.0" encoding="UTF
<!DOCTYPE personnel SYSTEM "perso
<?xml-stylesheet type="text/css"
<personnel>
    <person id="Big.Boss">
        <name>
            <family>Boss</family>
            <given>Big</given>
        </name>
        <email>chief@oxygenxml.co
        <link subordinates="one.w
    </person>
    <person id="one.worker">
        <name>
            <family>Worker</famil
            <given>One</given>
        </name>
        <email>one@oxygenxml.com<
        <link manager="Big.Boss"/
    </person>
    <person id="two.worker">
        <name>
            <family>Worker</famil
            <given>Two</given>
        </name>
        <email>two@oxygenxml.com<
        <link manager="Big.Boss"/
    </person>
    <person id="three.worker">
```

```
{"personnel": {"person": [
    {
        "id": "Big.Boss",
        "name": {
            "family": "Boss",
            "given": "Big"
        },
        "email": "chief@oxygenxml.c
        "link": {"subordinates": "o
    },
    {
        "id": "one.worker",
        "name": {
            "family": "Worker",
            "given": "One"
        },
        "email": "one@oxygenxml.com
        "link": {"manager": "Big.Bo
    },
    {
        "id": "two.worker",
        "name": {
            "family": "Worker",
            "given": "Two"
        },
        "email": "two@oxygenxml.com
        "link": {"manager": "Big.Bo
    },
    {
```

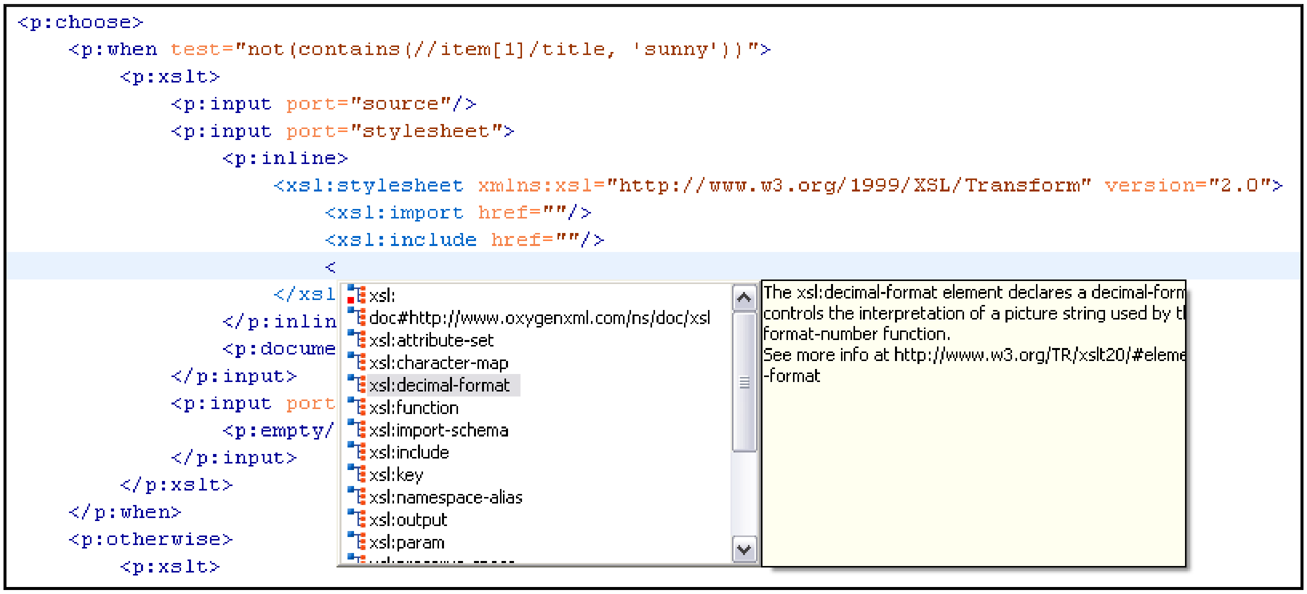
Editing XProc Scripts

An XProc script is edited as an XML document that is validated against a RELAX NG schema. If the script has an associated transformation scenario, then the XProc engine from the scenario is invoked as validating engine. The default engine for XProc scenarios is the Calabash engine which comes with Oxygen XML Editor plugin version 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.



```
<p:choose>
    <p:when test="not(contains(//item[1]/title, 'sunny'))">
        <p:xslt>
            <p:input port="source"/>
            <p:input port="stylesheet">
                <p:inline>
                    <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
                        <xsl:import href=""/>
                        <xsl:include href=""/>
                        <
                    </xsl
                </p:inlin
                <p:docume
            </p:input>
            <p:input port
                <p:empty/
            </p:input>
        </p:xslt>
    </p:when>
    <p:otherwise>
        <p:xslt>
```

The xsl:decimal-format element declares a decimal-form
controls the interpretation of a picture string used by t
format-number function.
See more info at http://www.w3.org/TR/xslt20/#eleme
-format

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 Editor 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 Editor 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 Editor 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 Editor 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 Editor plugin looks for the following information:

- the `role` attribute, which can have one of the following values:
  - `warn` or `warning`, to set the severity level to *warning*;
  - `error`, to set the severity level to *error*;
  - `fatal`, to set the severity level to *fatal*;
  - `info` or `information`, to set the severity level to *info*.
- the start of the message, after trimming leading white-spaces. Oxygen XML Editor plugin looks to match the following exact string of characters (case sensitive):
  - `Warning:`, to set the severity level to *warning*;

- `Error:`, to set the severity level to *error*;
- `Fatal:`, to set the severity level to *fatal*;
- `Info:`, to set the severity level to *info*;

  👉 **Note:** Displayed message does not contain the matched prefix.

- if none of the previous rules match, Oxygen XML Editor plugin sets the 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

Authoring in the Visual Editor

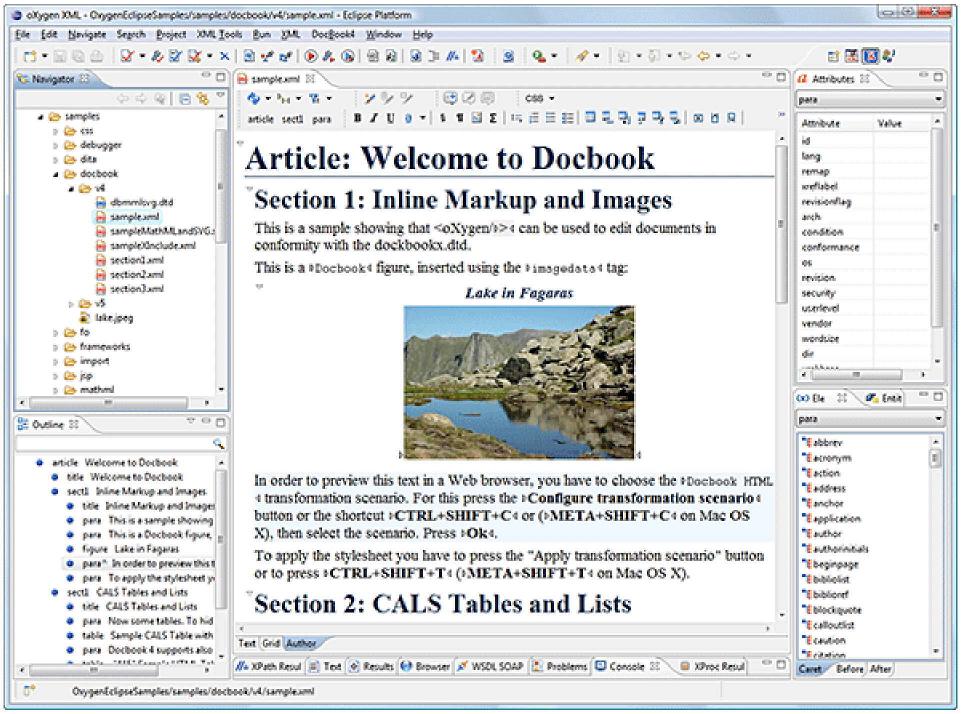
Topics:

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

Authoring XML Documents Without the XML Tags

Once the structure of the XML document and the required restrictions on the elements and attributes are fixed with an XML schema the editing of the document is easier in a WYSIWYG (what-you-see-is-what-you-get) editor in which the XML markup is not visible.

This tagless editor is available as the Author mode of the XML editor. The Author mode is activated by pressing the Author button at the bottom of the editing area where the mode switches of the XML editor are available: Text, Grid, and Author. The Author mode renders the content of the XML document visually based on a CSS stylesheet associated with the document. Many of the actions and features available in Text mode are also available in Author mode.


oXygen XML - OxygenEclipseSamples/samples/docbook/v4/sample.xml - Eclipse Platform
File Edit Navigate Search Project XML Tools Run XML DocBook4 Window Help
Navigator
sample.xml
Attributes
Outline
Text Grid Author
XPath Resul Text Results Browser WSDL SOAP Problems Console XProc Resul
OxygenEclipseSamples/samples/docbook/v4/sample.xml

Article: Welcome to Docbook

Section 1: Inline Markup and Images

This is a sample showing that <oXygen/> can be used to edit documents in conformity with the dockbookx.dtd.

This is a Docbook figure, inserted using the imagedata tag:

Lake in Fagaras

In order to preview this text in a Web browser, you have to choose the Docbook HTML transformation scenario. For this press the **Configure transformation scenario** button or the shortcut **CTRL+SHIFT+C** or (**META+SHIFT+C** on Mac OS X), then select the scenario. Press **Ok**.

To apply the stylesheet you have to press the "Apply transformation scenario" button or to press **CTRL+SHIFT+T** (**META+SHIFT+T** on Mac OS X).

Section 2: CALS Tables and Lists

Figure 113: Author editing mode

The tagless rendering of the XML document in the Author mode is driven by a CSS stylesheet which conforms to the *version 2.1 of the CSS specification* from the W3C consortium. Also some CSS 3 features like namespaces and custom extensions of the CSS specification are supported.

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

The association of such a stylesheet with an XML document is also straightforward: an `xml-stylesheet` XML processing instruction with the attribute `type="text/css"` must be inserted at the beginning of the XML document. If it is an XHTML document, that is the root element is an `html` element, there is a second method for the association of a CSS stylesheet: an element `link` with the `href` and `type` attributes in the `head` child element of the `html` element as *specified in the CSS specification*.

There are two main types of users of the Author mode: *framework developers* and *content authors*. A *framework developer* is a technical person with advanced XML knowledge who defines the framework for authoring XML documents in the tagless editor. Once the framework is created or edited by the developer, it is distributed as a deliverable component ready to plug into the application to the content authors. A *content author* does not need to have advanced knowledge about XML tags or operations like validation of XML documents or applying an XPath expression to an XML document. The author just plugs the framework set-up by the developer into the application and starts editing the content of XML documents without editing the XML tags directly.

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

- the CSS stylesheet which drives the tagless visual rendering of the document;
- the rules for associating an XML schema with the document which is needed for content completion and validation of the document;
- transformation scenarios for the document;
- XML catalogs;
- custom actions available as buttons on the toolbar.

The tagless editor comes with some ready to use predefined document types for XML frameworks largely used today like DocBook, DITA, TEI, XHTML.

General Author Presentation

A content author edits the content of XML documents in tagless mode disregarding the XML tags as they are not visible in the editor. If he edits documents conforming to one of the predefined types he does not need to configure anything as the predefined document types are already configured when the application is installed. Otherwise he must plug the configuration of the document type into the application. This is as easy as unzipping an archive directly in the `[Oxygen-install-folder]/frameworks` folder.

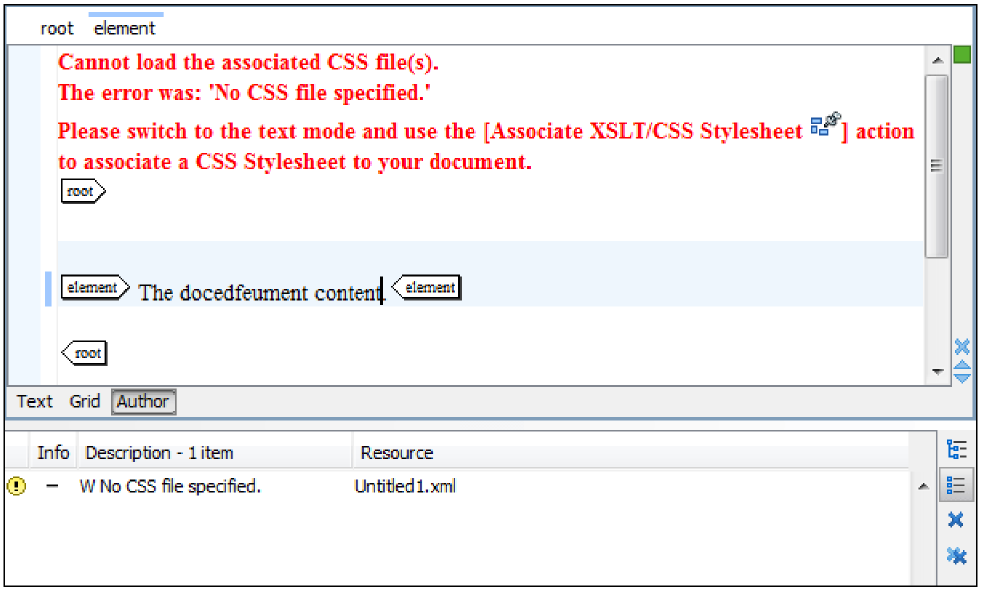
In case the edited XML document does not belong to one of the document types *set up in Preferences* you can specify the CSS stylesheets to be used by inserting an `xml-stylesheet` processing instructions. You can insert the processing instruction by editing the document or by using the **Associate XSLT/CSS stylesheet** action.

The syntax of such a processing instruction is:

```
<?xml-stylesheet type="text/css" media="media type" title="title"
href="URL" alternate="yes|no"?>
```

You can read more about associating a CSS to a document in the section about *customizing the CSS of a document type*.

When the document has no CSS association or the referred stylesheet files cannot be loaded, a default one is used. A warning message is also displayed at the beginning of the document presenting the reason why the CSS cannot be loaded.



root element

Cannot load the associated CSS file(s).
The error was: 'No CSS file specified.'
Please switch to the text mode and use the [Associate XSLT/CSS Stylesheet] action to associate a CSS Stylesheet to your document.

root
element The docedfeument content element
root

Text Grid Author

	Info	Description - 1 item	Resource
!	–	W No CSS file specified.	Untitled1.xml

Figure 114: Document with no CSS association default rendering

Author Views

The content author is supported by special views which are automatically synchronized with the current editing context of the editor panel. The views present additional information about this context thus helping the author to see quickly the current location in the overall document structure and the available editing options.

Outline View

The **Outline** view offers the following functionality:

- *Document Overview*
- *Outline View Specific Actions*
- *Modification Follow-up*
- *Document Structure Change*
- *Document Tag Selection*

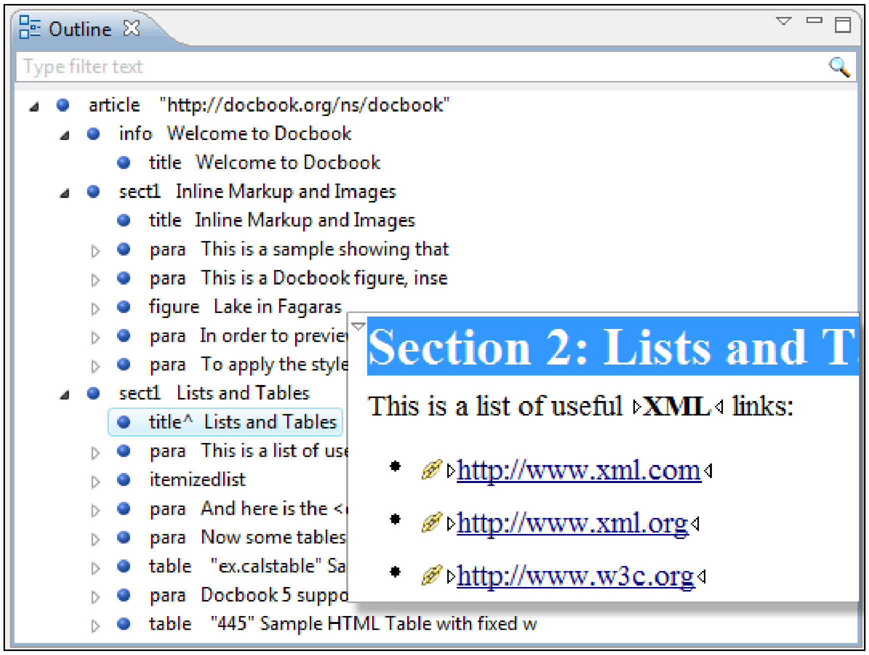


Figure 115: The Outline View

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.

Modification Follow-up

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

Document Structure Change

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

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

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

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

Outline Filters

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

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

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

The Contextual Menu of the Outline Tree

The contextual menu of the **Outline** tree contains the following actions:

- **Edit attributes** - A dialog is presented allowing the user to see and edit the attributes of the selected node.
- The **Append child**, **Insert before** and **Insert after** submenus allow to quickly insert new tags in the document at the place of the element selected in the **Outline** tree. The **Append child** submenu lists the names of all the elements which are allowed by the schema associated with the current document as child of the current element. The effect is the same as typing the '<' character and selecting an element name from the popup menu offered by *the content completion assistant.* The **Insert before** and **Insert after** submenus list the elements which are allowed by the schema associated with the current document as siblings of the current element inserted immediately before respectively after the current element.
- The **Cut**, **Copy** and **Delete** actions execute the same actions as the **Edit** menu items with the same name on the elements currently selected in the **Outline** tree (**Cut**, **Copy**, **Paste**).
- You can insert a well-formed element before, after or as a child of the currently selected element by accessing the **Paste before**, **Paste after** or **Paste as Child** actions.
- The **Toggle Comment** item encloses the currently selected element of the **Outline** tree in an XML comment, if the element is not commented, or removes the comment if it is commented.

- Using the **Rename Element** action the element from the caret position and the elements that have the same name as the current element can be renamed according with the options from the **Rename** dialog.
- The **Expand All / Collapse All** actions expand / collapse the selection and all its children.

Tip: You can copy, cut or delete multiple nodes in the **Outline** by using the contextual menu after selecting multiple nodes in the tree.

Elements View

The **Elements** view presents a list of all defined elements that you can insert in your document. All elements from a sequence are presented but the invalid proposals (which cannot be inserted in the current context) are grayed-out. The upper part of the view features a combo box that contains the current element's ordered ancestors. Selecting a new element in this combo box updates the list of the allowed elements in **Before** and **After** tabs.

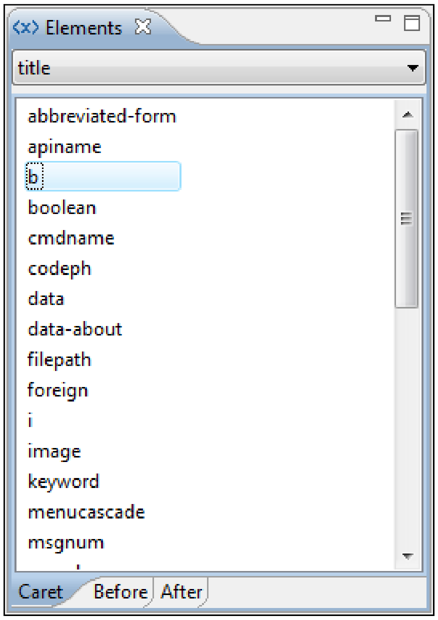


Figure 116: The Elements View

Three tabs present information relative to the caret location:

- **Caret** - Shows a list of all the elements allowed at the current caret location. Double-clicking any of the listed elements inserts that element at the caret position.
- **Before** - Shows a list of all elements that can be inserted before the element selected in the combo box. Double-clicking any of the listed elements inserts that element before the element at the caret position.
- **After** - Shows a list of all elements that can be inserted after the element selected in the combo box. Double-clicking any of the listed elements inserts that element after the element at the caret position.

Double clicking an element name in the list surrounds the current selection in the editor panel with the start tags and end tags of the element. If there is no selection, just an empty element is inserted in the editor panel at the cursor position.

Attributes View

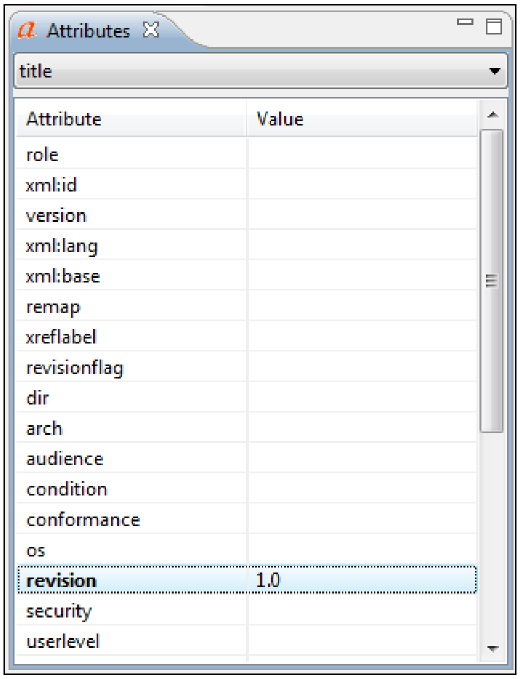
The **Attributes** view presents all the possible attributes of the current element allowed by the schema of the document. It allows you to insert attributes in the current element or change the value of the attributes already used in the element. Different renderings are used for marking the attribute states:

- attributes with a specified value are rendered with a bold font;
- default values are rendered with an plain font, painted gray;
- empty values display the text "[empty]", painted gray;
- invalid attributes and values are painted red;

Clicking a cell in the **Value** column starts editing the value of the attribute from the selected row. If the possible values of the attribute are specified as list in the schema associated with the edited document, the **Value** column works as a combo box where you can select one of the possible values to be inserted in the document.

The attributes table is sortable by clicking the **Attribute** column header. Thus the table contents can be sorted:

- by attribute name in ascending order;
- by attribute name in descending order;
- custom order, where the used attributes are displayed at the beginning of the table sorted in ascending order, followed by the rest of the allowed elements sorted in ascending order.



Attribute	Value
role	
xml:id	
version	
xml:lang	
xml:base	
remap	
xreflabel	
revisionflag	
dir	
arch	
audience	
condition	
conformance	
os	
revision	1.0
security	
userlevel	

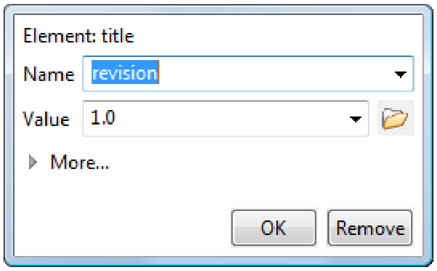
Figure 117: The Attributes View

A combo box located in the upper part of the view allows you to edit the attributes of the ancestors of the current element.

The following actions may be available in the contextual menu:

- **Add** - allows you to insert a new attribute. Adding an attribute that is not in the list of all defined attributes is not possible when the *Allow only insertion of valid elements and attributes* schema aware option is enabled.
- **Set empty value** - Specifies the current attribute value as empty.
- **Remove** - Removes the attribute (action available only if the attribute is specified). You can invoke this action by pressing the **(Delete)** or **(Backspace)** keys.

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



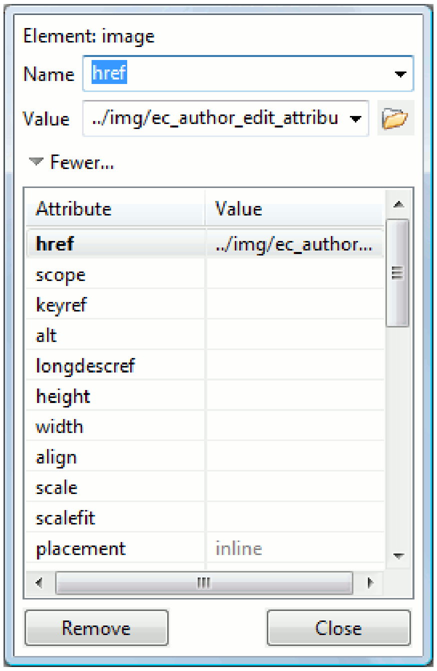
Element: title

Name revision

Value 1.0

Figure 118: Edit attributes in place

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



Element: image

Name: href

Value: ../img/ec_author_edit_attribu

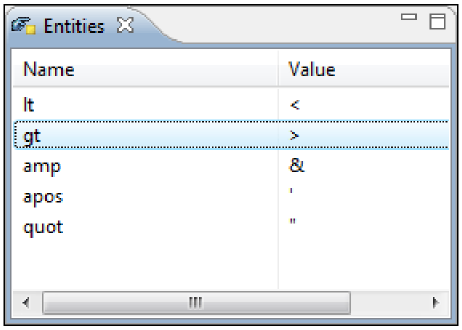
Attribute	Value
href	../img/ec_author...
scope	
keyref	
alt	
longdescref	
height	
width	
align	
scale	
scalefit	
placement	inline

Figure 119: Edit attributes in place - full version

The **Name** field auto-completes the name of the attribute: the complete name of the attribute is suggested based on the prefix already typed in the field as the user types in the field.

Entities View

This view displays a list with all entities declared in the current document as well as built-in ones. Double clicking one of the entities will insert it at the current cursor position.



Name	Value
lt	<
gt	>
amp	&
apos	'
quot	"

Figure 120: The Entities View

The Author Editor

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

Navigating the Document Content

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

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

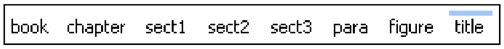


Figure 121: The breadcrumb in Editor view

The last element is also highlighted by a thin light blue bar for easier identification. Clicking one element from the top stripe selects the entire element in the editor view.

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

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

The **Append child**, **Insert before** and **Insert after** submenus of the top stripe popup menu allow you to insert new tags in the document at the place of the selected element. The **Append child** submenu lists the names of all the elements which are allowed by the schema associated with the current document as child of the current element. The effect is the same as typing the '<' character and selecting an element name from the popup menu offered by *the content completion assistant*. The **Insert before** and **Insert after** submenus list the elements which are allowed by the schema associated with the current document as siblings of the current element inserted immediately before respectively after the current element.

The **Cut**, **Copy**, **Paste** and **Delete** items of the popup menu execute the same actions as the **Edit** menu items with the same name on the elements currently selected in the stripe (Cut, Copy, Paste, Delete). The **Cut** and **Copy** operations (like the `display:block` property or the tabular format of the data from a set of table cells) preserve the styles of the copied content. The **Paste before**, **Paste after** and **Paste as Child** actions allow the user to insert an well-formed element before, after or as a child of the currently selected element.

The **Toggle Comment** item of the **Outline** tree popup menu encloses the currently selected element of the top stripe in an XML comment, if the element is not commented, or removes the comment if it is commented.

Using the **Rename Element** action the selected element and the elements that have the same name as the current element can be renamed according with the options from the **Rename** dialog.

When working on a large document, the **folding support** can be used to collapse some elements content leaving in focus only the ones you need to edit. Foldable elements are marked with a small triangle painted in the upper left corner. Hovering with the mouse pointer over that marker, the entire content of the element is highlighted by a dotted border for quick identification of the foldable area. The following actions are available in the contextual menu, **Folding** sub-menu:

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

When working on a suite of documents that refer to one another (references, external entities, XInclude, DITA conref, etc), the **linking support** is useful for navigating between the documents. In the predefined customizations that are bundled with Oxygen XML Editor plugin links are marked with an icon representing a chain link: . When hovering with the mouse pointer over the marker, the mouse pointer changes its shape to indicate that the link can be followed and a tooltip presents the destination location. Click a followable link to open the referred resource in an editor. The same effect can be obtained by using the action **Open file at caret** when the caret is in a followable link element.

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

Displaying the Markup

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

- **Full Tags with Attributes** - Displays full name tags with attributes for both block level as well as in-line level elements.

- **Full Tags** - Displays full name tags without attributes for both block level as well as in-line level elements.
- **Block Tags** - Displays full name tags for block level elements and simple tags without names for in-line level elements.
- **Inline Tags** - Displays full name tags for inline level elements, while block level elements are not displayed.
- **Partial Tags** - Displays simple tags without names for in-line level elements, while block level elements are not displayed.
- **No Tags** - None of the tags is displayed. This is the most compact mode.

The default tags display mode can be configured in the *Author options page*. However, if the document opened in Author editor does not have an associated CSS stylesheet, then the **Full Tags** mode will be used.

Block-level elements are those elements of the source document that are formatted visually as blocks (e. g. paragraphs), while the inline level elements are distributed in lines (e. g. emphasizing pieces of text within a paragraph, inline images, etc). The graphical format of the elements is controlled from the CSS sources via the `display` property.

Bookmarks

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

A bookmark can be placed with:

- one of the menu items available on the menu **Edit** > **Bookmarks** > **Create**
- the menu item **Edit** > **Bookmarks** > **Bookmarks Quick Creation** **(F9)**
- the keyboard shortcuts associated with these menu items and visible on the menu **Edit** > **Bookmarks**

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

Position Information Tooltip

When the caret is positioned inside a new context, a tooltip will be shown for a couple of seconds displaying the position of the caret relative to the current element context.

Here are the common situations that can be encountered:

- The caret is positioned before the first block child of the current node.

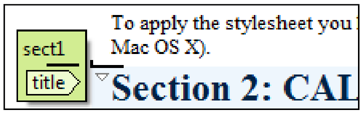


Figure 122: Before first block

- The caret is positioned between two block elements.

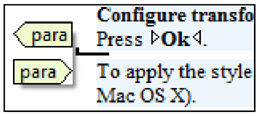


Figure 123: Between two block elements

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

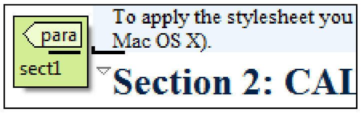
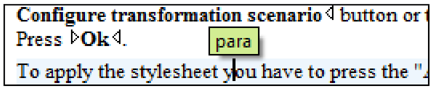


Figure 124: After last block

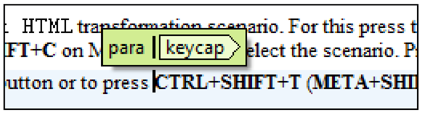
- The caret is positioned inside a node.



Configure transformation scenario button or t
Press **Ok**.
To apply the stylesheet you have to press the "

Figure 125: Inside a node

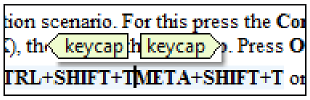
- The caret is positioned inside an element, before an inline child element.



HTML transformation scenario. For this press t
FT+C on M elect the scenario. P
utton or to press **CTRL+SHIFT+T (META+SHI**

Figure 126: Before an inline element

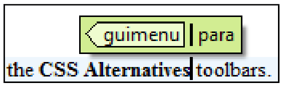
- The caret is positioned between two inline elements.



tion scenario. For this press the Co
), th h . Press O
TRL+SHIFT+T META+SHIFT+T or

Figure 127: Between two inline elements

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



the CSS **Alternatives** toolbars.

Figure 128: After an inline element

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

You can deactivate this feature by unchecking the **Options** > **Preferences** > **Editor / Author** > **Show caret position tooltip** check box. Even if this option is disabled, you can trigger the display of the position tooltip by pressing **Shift+F2**.

Note: The position information tooltip is not displayed if one of the modes *Full Tags with Attributes* or *Full Tags* is selected.

Displaying Referred Content

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

When the referred resource cannot be resolved, an error will be presented inside the element that refers them instead of the content.

If you want to make modifications to the referred content, you must open the referred resource in an editor. The referred resource can be opened quickly by clicking the link (marked with the icon) which is displayed before the referred content or by using the **Edit Reference** action from the contextual menu (in this case the caret is placed at the precise location where the action was invoked in the main file). The referred resource is resolved through the XML Catalog set in **Preferences**.

The referred content is refreshed:

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

Finding and Replacing Text

The **Find / Replace** dialog can be used in the Author mode in the same way as in the Text mode.

Contextual Menu

More powerful support for editing the XML markup is offered via actions included in the contextual menu. Two types of actions are available: **generic actions** (actions that not depends on a specific document type) and **document type actions** (actions that are configured for a specific document type).

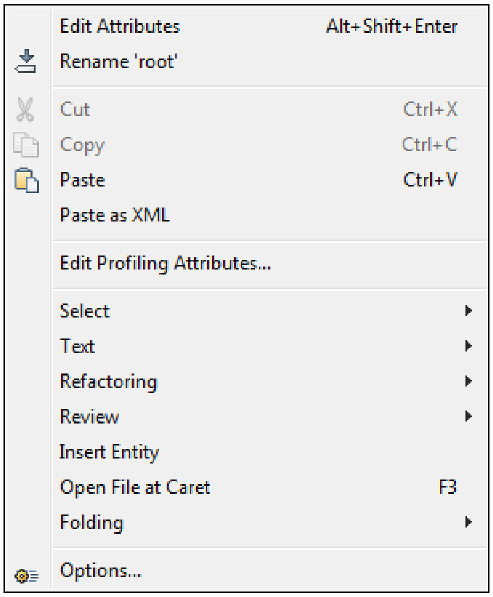


Figure 129: Contextual menu

The generic actions are:

- **Edit Attributes** - A pop-up window is displayed allowing you to manage the element attributes.
- **Rename** - The element from the caret position can be renamed quickly using the content completion window. If the *Allow only insertion of valid elements and attributes* schema aware option is enabled only the proposals from the content completion list are allowed, otherwise a custom element name can also be provided.
- **Cut**, **Copy**, **Paste** - Common edit actions with the same functionality as those found in the text editor.
- **Paste As XML** - Similar to **Paste** operation, except that the clipboard's content is considered to be XML.
- **Edit Profiling Attributes** - Allows you to select the profiling attributes.
- **Select** - Contains the following actions:

  - **Select** > **Select Element** - Selects the entire element at the current caret position.
  - **Select** > **Select Content** - Selects the entire content of the element at the current caret position, excluding the start and end tag. Performing this action repeatedly will result in the selection of the content of the ancestor of the currently selected element content.
  - **Select** > **Select Parent** - Selects the parent of the element at the current caret position.

  Note: You can select an element by triple clicking inside its content. If the element is empty you can select it by double clicking it.

- **Text** - Contains the following actions:

  - **Text** > **To Lower Case** - Converts the selection content to lower case characters.
  - **Text** > **To Upper Case** - Converts the selection content to upper case characters.
  - **Text** > **Capitalize Sentences** - Converts to upper case the first character of every selected sentence.
  - **Text** > **Capitalize Words** - Converts to upper case the first character of every selected word.
  - **Text** > **Count Words** - Counts the number of words and characters (no spaces) in the entire document or in the selection for regular content and read-only content.

    Note: The content marked as deleted with track changes is ignored when counting words.

- **Refactoring** - Contains a series of actions designed to alter the document's structure:
  - **Toggle Comment** - Encloses the currently selected text in an XML comment, or removes the comment if it is commented.
  - **Split Element** - Splits the content of the closest element that contains the caret's position. Thus, if the caret is positioned at the beginning or at the end of the element, the newly created sibling will be empty.
  - **Join Elements** - Joins two adjacent elements that have the same name. The action is available only when the caret position is between the two adjacent elements. Also, joining two elements can be done by pressing the Delete or Backspace keys and the caret is positioned between the boundaries of these two elements.
  - **Surround with Tag...** - Selected text in the editor is marked with the specified start and end tags.
  - **Surround with '<Tag name>'** - Selected text in the editor is marked with start and end tags used by the last '**Surround with Tag...**' action.
  - **Rename Element** - The element from the caret position and the elements that have the same name as the current element can be renamed according with the options from the **Rename** dialog.
  - **Delete Element Tags** - Deletes the tags of the closest element that contains the caret's position. This operation is also executed if the start or end tags of an element are deleted by pressing the **(Delete)** or **(Backspace)** keys.
- **Review** - Provides access to *Track Changes* and Manage Comments actions.
- **Insert Entity** - Allows the user to insert a predefined entity or a character entity. Surrogate character entities (range #x10000 to #x10FFFF) are also accepted. Character entities can be entered in one of the following forms:
  - #<*decimal value*> - e. g. #65
  - &#<*decimal value*>; - e. g. &#65;
  - #x<*hexadecimal value*> - e. g. #x41
  - &#x<*hexadecimal value*>; - e. g. &#x41;
- **Open File at Caret** - Opens in a new editor panel the file with the path under the caret position. If the path represents a directory path, it will be opened in system file browser. If the file does not exist at the specified location, the error dialog that is displayed contains a **Create new file** action which displays the **New** file dialog. This allows you to choose the type or the template for the file. If the action succeeds, the file is created with the referred location and name and is opened in a new editor panel. This is useful when you decide first on the file name and after that you want to create it in the exact location specified at the current cursor position.
- **Options** - Opens the *Author options page*.

Document type actions are specific to some document type. Examples of such actions can be found in section *Predefined document types.*

Editing XML Documents in Author

This section details how to edit the text content and the markup of XML documents in Author mode. It explains also how to edit tables and MathML content in Author mode.

Editing the XML Markup

One of the most useful feature in Author editor is the content completion support. The fastest way to invoke it is to press **(Enter)** or **(Ctrl + Space)** (on *Mac OS X* the shortcut is **(Meta + Space)**) in the editor panel.

Content completion window offers the following types of actions:

- inserting allowed elements for the current context according to the associated schema, if any;
- inserting element values if such values are specified in the schema for the current context;
- inserting new undeclared elements by entering their name in the text field;
- inserting CDATA sections, comments, processing instructions;
- inserting *code templates*.

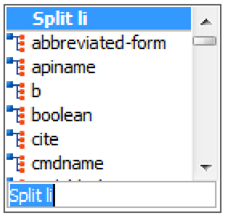


Figure 130: Content completion window

If you press **(Enter)** the displayed content completion window will contain as first entries the **Split <Element name>** items. Usually you can only split the closest block element to the caret position but if it is inside a list item, the list item will also be proposed for split. Selecting **Split <Element name>** splits the content of the specified element around the caret position. Thus, if the caret is positioned at the beginning or at the end of the element, the newly created sibling will be empty.

If the caret is positioned inside a space preserve element the first choice in the content completion window is **Enter** which inserts a new line in the content of the element. If there is a selection in the editor and you invoke content completion, a **Surround with** operation can be performed. The tag used will be the selected item from the content completion window.

By default you are not allowed to insert element names which are not considered by the associated schema as valid proposals in the current context. This can be changed by unchecking the **Allow only insertion of valid elements and attributes** check box from the *Schema aware preferences page*.

Joining two elements - You can choose to join the content of two sibling elements with the same name by using the **contextual menu** > **Join elements** action.

The same action can be triggered also in the next situations:

- The caret is located before the end position of the first element and **(Delete)** key is pressed.
- The caret is located after the end position of the first element and **(Backspace)** key is pressed.
- The caret is located before the start position of the second element and **(Delete)** key is pressed.
- The caret is located after the start position of the second element and **(Backspace)** key is pressed.

In either of the described cases, if the element has no sibling or the sibling element has a different name, **Unwrap** operation will be performed automatically.

Unwrapping the content of an element - You can unwrap the content of an element by deleting its tags using the **Delete element tags** action from the editor contextual menu.

The same action can be triggered in the next situations:

- The caret is located before the start position of the element and **(Delete)** key is pressed.
- The caret is located after the start position of the element and **(Backspace)** key is pressed.
- The caret is located before the end position of the element and **(Delete)** key is pressed.
- The caret is located after the end position of the element and **(Backspace)** key is pressed.

Removing all the markup of an element - You can remove the markup of the current element and keep only the text content with the action **Remove All Markup** available on the submenu **Refactoring** of the contextual menu and on the toolbar **XML Refactoring**.

When you press **(Delete)** or **(Backspace)** in the presented cases the element is unwrapped or it is joined with its sibling. If the current element is empty, the element tags will be deleted.

When you click on a marker representing the start or end tag of an element, the entire element will be selected. The contextual menu displayed when you right-click on the marker representing the start or end tag of an element contains **Append child**, **Insert Before** and **Insert After** submenus as first entries.

Code Templates

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

Oxygen XML Editor 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*:

- **${caret}** - The position where the caret is inserted. This variable can be used in a *code template* , in Author operations, or in a selection plugin.
- **${selection}** - The XML content of the current selection in the editor panel. This variable can be used in a *code template* and Author operations,
- **${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 Editor plugin checks that the URL is valid before passing it to the transformation.
        - `password` - input characters are hidden.
        - `generic` - the input is treated as generic text that requires no special handling.
    - `'default-value'` - optional parameter. Provides a default value in the input text box.

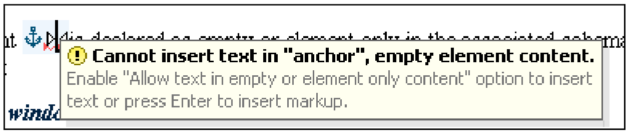
      Examples:

        - `${ask('message')}` - Only the message displayed for the user is specified.
        - `${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 Editor plugin installation folder.
- **${pd}** - Current project folder as file path.
- **${oxygenInstallDir}** - Oxygen XML Editor plugin installation folder as file path.

- **${homeDir}** - The path (as file path) of the user home folder.
- **${pn}** - Current project name.
- **${env(VAR_NAME)}** - Value of the *VAR_NAME* environment variable.
- **${system(var.name)}** - Value of the *var.name* system variable.
- **${date(pattern)}** - Current date. Follows the given pattern. Example: yyyy-MM-dd.

Editing the XML Content

By default you can type only in elements which accept text content. So if the element is declared as empty or element only in the associated schema you are not allowed to insert text in it. This is also available if you try to insert `CDATA` inside an element. Instead a warning message is shown:



Cannot insert text in "anchor", empty element content.
Enable "Allow text in empty or element only content" option to insert text or press Enter to insert markup.

Figure 131: Editing in empty element warning

You can disable this behavior by checking the **Allow Text in empty or element only content** check box in the *Author preferences page*.

Entire sections or chunks of data can be moved or copied by using the drag and drop support. The following situations can be encountered:

- when both the drag and drop sources are Author pages, an well-formed XML fragment is transferred. The section is balanced before dropping it by adding matching tags when needed.
- when the drag source is the Author page but the drop target is a text-based editor only the text inside the selection is transferred as it is.
- the text dropped from another text editor or another application into the Author page is inserted without changes.

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

- **(Ctrl - NumPad+)** or **(Ctrl - +)** or **(Ctrl - mouse wheel)** - Increases font size.
- **(Ctrl - NumPad-)** or **(Ctrl - -)** or **(Ctrl - mouse wheel)** - Decreases font size.
- **(Ctrl - NumPad0)** or **(Ctrl - 0)** - Restores font size to *the size specified in **Preferences***.

Removing the Text Content of the Current Element

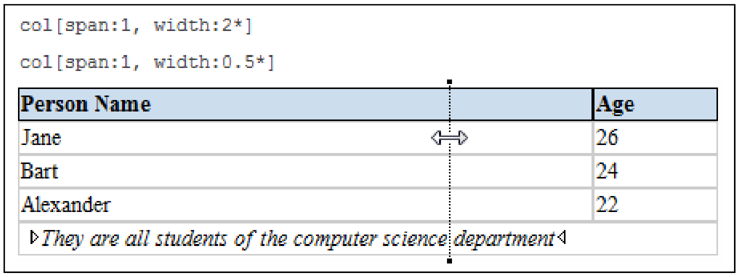
You can remove the text content of the current element and keep only the markup with the action **Remove Text** available on the submenu **Refactoring** of the contextual menu and on the toolbar **XML Refactoring**. This is useful when the markup of an element must be preserved, for example a table structure but the text content must be replaced.

Duplicating Elements with Existing IDs

If the **Auto generate IDs for elements** option (available in the **ID Options** dialog from DITA, Docbook and TEI document types) is turned off and you duplicate elements with existing IDs, the duplicates lose these IDs. If the previously mentioned option is active, when you duplicate content, Oxygen makes sure that if there is an ID attribute set in the XML markup, the newly created duplicate has a new, unique ID attribute value. The option **Remove ID's when copying content in the same document** allows you to control if a pasted element should retain its ID.

Table Layout and Resizing

The support for editing data in tabular form can manage table width and column width specifications from the source document. The specified widths will be considered when rendering the tables and when visually resizing them using mouse drag gestures. These specifications are supported both in fixed and proportional dimensions. The predefined frameworks (DITA, DocBook and XHTML) already implement support for this feature. The layout of the tables from these types of documents takes into account the table width and the column width specifications particular to them. The tables and columns widths can be visually adjusted by dragging with the mouse their edges and the modifications will be committed back into the source document.



```
col[span:1, width:2*]
col[span:1, width:0.5*]
```

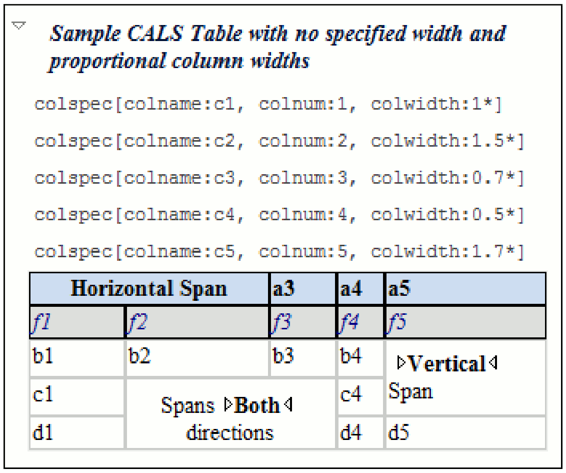
Person Name	Age
Jane	26
Bart	24
Alexander	22
▷*They are all students of the computer science department*◁	

Figure 132: Resizing a column in Oxygen XML Editor plugin Author editor

DocBook Table Layout

The DocBook table layout supports two models: CALS and HTML.

In the CALS model column widths can be specified by using the `colwidth` attribute of the associated `colspec` element. The values can be fixed or proportional.



Sample CALS Table with no specified width and proportional column widths

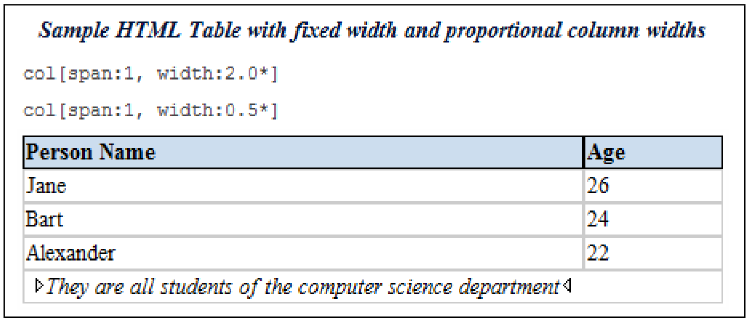
```
colspec[colname:c1, colnum:1, colwidth:1*]
colspec[colname:c2, colnum:2, colwidth:1.5*]
colspec[colname:c3, colnum:3, colwidth:0.7*]
colspec[colname:c4, colnum:4, colwidth:0.5*]
colspec[colname:c5, colnum:5, colwidth:1.7*]
```

Horizontal Span		a3	a4	a5
f1	*f2*	*f3*	*f4*	*f5*
b1	b2	b3	b4	▷**Vertical**◁ Span
c1	Spans ▷**Both**◁ directions		c4	
d1			d4	d5

Figure 133: CALS table

XHTML Table Layout

The HTML table model accepts both table and column widths by using the width attribute of the table element and the col element associated with each column. The values can be represented in fixed units, proportional units or percentages.



Sample HTML Table with fixed width and proportional column widths

```
col[span:1, width:2.0*]
col[span:1, width:0.5*]
```

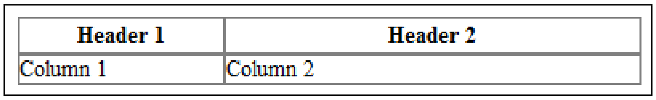
Person Name	Age
Jane	26
Bart	24
Alexander	22
▷*They are all students of the computer science department*◁	

Figure 134: HTML table

DITA Table Layout

The DITA table layout accepts CALS tables and simple tables.

The simple tables accept only relative column width specifications by using the `relcolwidth` attribute of the `simpletable` element.



Header 1	Header 2
Column 1	Column 2

Figure 135: DITA simple table

Image Rendering

The Author editor and the output transformation process might render differently the images referenced in the XML document, since they use different rendering engines. The following image formats are supported by default: GIF, JPEG, PNG, SVG, BMP. Oxygen XML also provides experimental support for *CGM images*.

To allow Oxygen XML to display other formats (like TIFF, JPEG 2000 or WBMP, for example) *install the Java Advanced Imaging (JAI) Image I/O Tools plug-in*.

When an image cannot be rendered, Oxygen XML Editor plugin XML Author displays a warning message that contains the reason why this is happening. Possible causes:

- the image is very large and you need to enable *Show very large images* option;
- the image format is not supported by default. It is recommended to *install the Java Advanced Imaging(JAI) Image I/O Tools plug-in*.

Scaling Images

Image dimension and scaling attributes are taken into account when an image is rendered. The following rules apply:

- if you specify only the width attribute of an image, the height of the image is proportionally applied;
- if you specify only the height attribute of an image, the width of the image is proportionally applied;
- if you specify width and height attributes of an image, both of them controls the rendered image;
- if you want to scale proportionally both the width and height of an image, use the *scale* attribute.

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

Follow this procedure:

1. Start Oxygen XML Editor plugin and open the **Help** > **About** dialog. Open the **Installation Details** dialog, **Configuration** tab and look for *java.runtime.name* and *java.home* properties. Keep their values for later use.
2. *Download* the JAI Image I/O kit corresponding to your operating system and Java distribution (found in the *java.runtime.name* property).

   Please note that the JAI API is not the same thing as JAI Image I/O. Make sure you have installed the latter.
3. Execute the installer. When the installation wizard displays the **Choose Destination Location** page, fill-in the **Destination Folder** field with the value of the *java.home* property. Continue with the installation procedure and follow the on-screen instructions.

Mac OS X Workaround

There is no native implementation of JAI Image I/O for Mac OS X 10.5 and later. However, the JAI Image I/O has a Java implementation fallback which also works on Mac OS X. Some of the image formats are not fully supported in this fallback mode, but at least the TIFF image format is known to be supported.

1. Download a Linux(tar.gz) distribution of JAI Image I/O from: *http://download.java.net/media/jai-imageio/builds/release/1.1/* e.g. jai_imageio-1_1-lib-linux-amd64.tar.gz
2. In the Oxygen/lib directory create a directory named **endorsed** e.g. **Oxygen/lib/endorsed**.
3. Unpack the tar.gz and navigate to the lib directory from the unpacked directory. e.g. **jai_imageio-1_1/lib**. Copy the jar files from there(clibwrapper_jiio.jar and jai_imageio.jar) to the **Oxygen/lib/endorsed** directory.
4. Restart the application and the JAI Image I/O support will be up and running.

Customize Oxygen XML to Render CGM Images (Experimental Support)

Oxygen XML provides experimental support for CGM 1.0 images.

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

Since it is an experimental support, some graphical elements might be missing from the rendered image.

Follow this procedure to enable the rendering of CGM images in Author mode:

1. Download the `CGMPANEL.ZIP` from *http://www.bdaum.de/CGMPANEL.ZIP*.
2. Unpack the ZIP archive and copy the `cgmpanel.jar` into `OXYGEN_INSTALL_DIR\lib` directory.
3. Open `plugin.xml` and add the following configuration line:`<library name="lib/cgmpanel.jar"/>`
4. Restart the application.

Adding an Image

An image can be inserted in an XML document edited in Author mode with the following methods:

- The insert image actions from the predefined document types. The following document types have an insert image action: DocBook 4, DocBook 5, DITA, TEI P4, TEI P5, XHTML.
- Drag an image from other application and drop it in the Author editor. If it is an image file, it is inserted as a reference to the image file. For example, in a DITA topic the path of the image file is inserted as the value of the `href` attribute in an `image` element:

  ```
  <image href="../images/image_file.png"/>
  ```

  If it is only an image part, first it is saved as a file using the file save dialog which is displayed automatically. After saving the image to a file the file path is inserted at the drop position as specified above.
- Copy the image from other application and paste it in your document. The content inserted in the document is similar with that added after dragging and dropping an image.

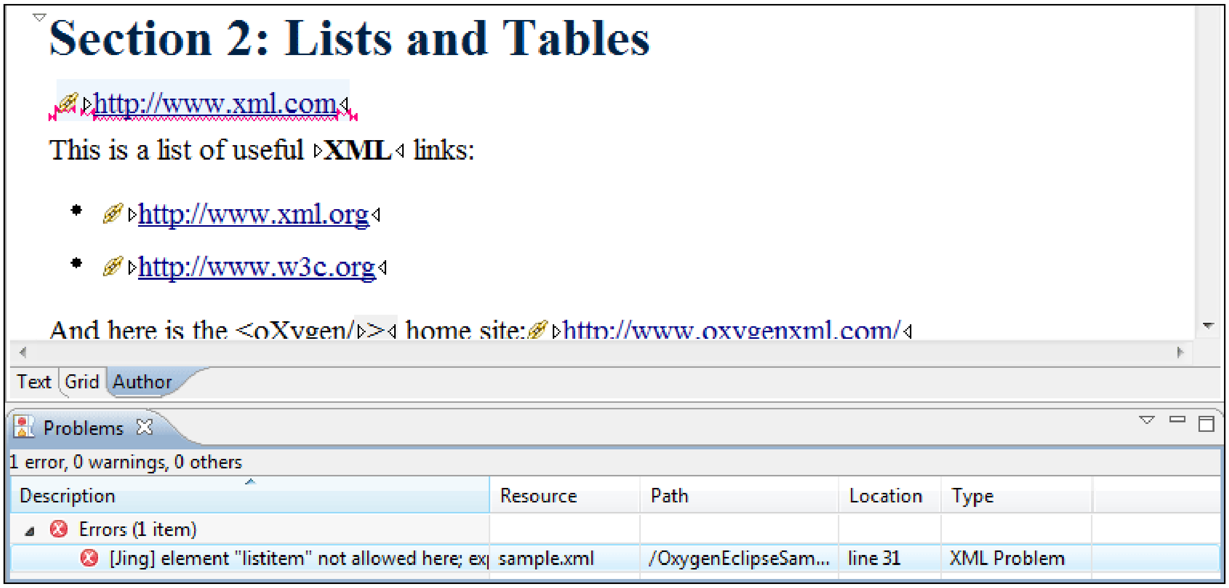
Refreshing the Content

On occasion you may need to reload the content of the document from the disk or reapply the CSS. This can be performed by using the **Reload** action.

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

Validation and Error Presenting

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



Section 2: Lists and Tables

http://www.xml.com

This is a list of useful XML links:

- http://www.xml.org
- http://www.w3c.org

And here is the <oXygen/> home site: http://www.oxygenxml.com/

1 error, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (1 item)				
[Jing] element "listitem" not allowed here; ex	sample.xml	/OxygenEclipseSam...	line 31	XML Problem

Figure 136: Error presenting in Oxygen XML Editor plugin Author editor

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

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

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

Whitespace Handling

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

- when opening documents
- when switching from other editing mode to Author mode
- when saving documents in Author mode
- when switching from Author mode to another one

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

    - Otherwise the white-spaces are ignored.

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

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

  - Before a text node that starts with a white-space.
  - After a text node that ends with a white-space.
  - Before and after CSS `block` nodes.
  - If the current node has an ancestor that is a CSS `table` element.
- **Editing documents** - You can insert space characters in any text nodes. Line breaks are permitted only in space-preserve elements. Tabs are marked in the space-preserve elements with a little marker.

Minimize Differences Between Versions Saved on Different Computers

The number of differences between versions of the same file saved by different content authors on different computers can be minimized by imposing the same set of formatting options when saving the file, for all the content authors. An example for a procedure that minimizes the differences is the following.

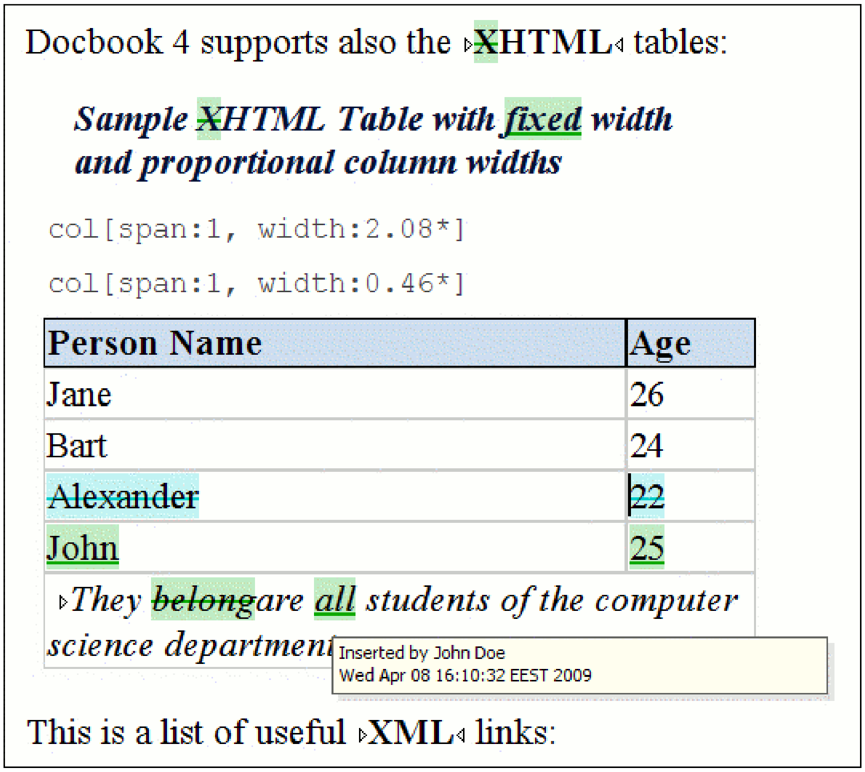
1. Create an Oxygen XML Editor plugin project file that will be shared by all content authors.
2. Set your own preferences in the following panels of the **Preferences** dialog: **Editor / Format** and **Editor / Format / XML**.
3. Save the preferences of these two panels in the Oxygen XML Editor plugin project by selecting the button **Project Options** in these two panels.
4. Save the project and commit the project file to your versioning system so all the content authors can use it.
5. Make sure the project is opened in the **Project** view.
6. Open and save your XML files in the Author mode.
7. Commit the saved XML files to your versioning system.

When other content authors will change the files only the changed lines will be displayed in your diff tool instead of one big change that does not allow to see the changes between two versions of the file.

Review

Tracking Document Changes

Track Changes is a way to keep track of the changes you make to a document. You can activate change tracking for the current document by choosing **Edit** > **Review** > **Track Changes** or by clicking the **Track Changes** button located on the Author toolbar. When *Track Changes* is enabled, your modifications are highlighted using a distinctive color. The name of the author who is currently making changes and the colors can be customized from the *Review* preferences page.



Docbook 4 supports also the ~~X~~HTML tables:

Sample ~~X~~HTML Table with fixed width and proportional column widths

col[span:1, width:2.08*]

col[span:1, width:0.46*]

Person Name	Age
Jane	26
Bart	24
~~Alexander~~	~~22~~
John	25

They ~~belong~~are all students of the computer science department

Inserted by John Doe
Wed Apr 08 16:10:32 EEST 2009

This is a list of useful XML links:

Figure 137: Change Tracking in Author Mode

When hovering a change the tooltip will display information about the author and modification time.

If the selection in the Author contains track changes and you are copying it, the clipboard contains the selection with all the *accepted* changes. This filtering is performed only if the selection is not entirely inside a tracked change.

Tip: The changes are stored in the document as processing instructions and they do not interfere with validating and transforming it. For each change the author name and the modification time are preserved. The following processing instructions are examples of storing *insert* and *delete* changes in the document:

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

Adding Comments into a Document

You can associate a note or a comment to a selected area of text content. Comments can highlight virtually any content from your document, except *read-only* text. The difference between such comments and change tracking is that a comment can be associated to an area of text without modifying or deleting the text.

The actions for managing comments are **Add Comment...**, **Edit Comment...**, **Delete Comment...** and **Manage Comments...** and are available on the **Author Comments** toolbar and on the **Review** submenu of the contextual menu of Author editor.

Tip: The comments are stored in the document as processing instructions containing information about the author name and the comment time:

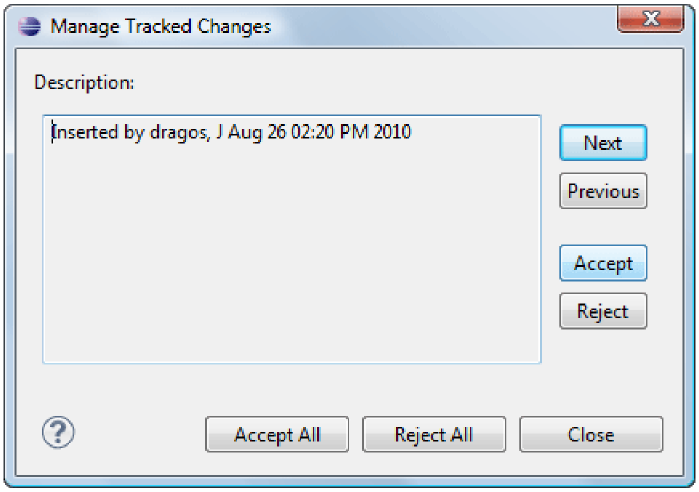
```
<?oxy_comment_start author="John Doe" timestamp="20090508T164459+0300"
comment="Do not change this content"?>
    Important content
<?oxy_comment_end?>
```

Comments are persistent highlights with a colored background. The background color is customizable or can be assigned automatically by the application. This behavior can be controlled from the *Review options page*.

Managing Changes

You can review the changes made by you or other authors and then accept or reject them using the **Track Changes** toolbar buttons or the similar actions from the **Edit** > **Review** menu.

- **Track Changes** - Enables or disables track changes support for the current document.
- **Accept Change(s)** - Accepts the change located at the caret position. If you select a part of a delete or insert change, then only the selected content is accepted. If you select multiple changes, all of them are accepted. For an insert change, it means keeping the inserted text and for a delete change it means removing the content from the document.
- **Reject Change(s)** - Rejects the change located at the caret position. If you select a part of a delete or insert change, then only the selected content is rejected. If you select multiple changes, all of them are rejected. For an insert change, it means removing the inserted text and for a delete change it means preserving the original content from the document.
- **Comment Change** - You can decide to add additional comments to an already existing change. The additional description appears in the tooltip when hovering over the change and in the **Manage Tracked Changes** dialog when navigating changes.
- **Manage Tracked Changes** - Action designed to find and manage all changes in the current document.



Inserted by dragos, J Aug 26 02:20 PM 2010

Figure 138: Manage Tracked Changes

The dialog offers the following actions:

- **Next** - Finds the next change in the document.
- **Previous** - Finds the previous change in the document.
- **Accept** - Accepts the current change. This action is also available on the contextual menu.
- **Reject** - Rejects the current change. This action is also available on the contextual menu.
- **Accept All** - Accepts all changes in the document.
- **Reject All** - Rejects all changes in the document.

The dialog is not modal and it is reconfigured after switching between the dialog and one of the opened editors.

Track Changes Visualization Modes

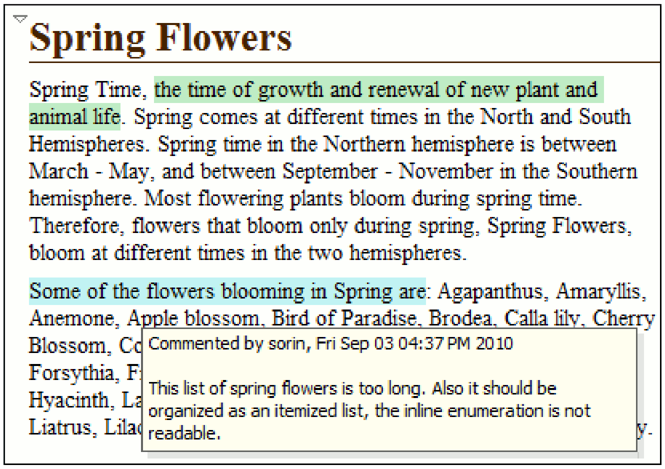
Three specialized actions allow you to switch between the following visualization modes:

- **View All Changes** - default visualization mode, all tracked changes are represented in the Author mode;
- **View Final** - previews the document as if all tracked changes (both inserted and deleted) were accepted;
- **View Original** - previews the document as if all tracked changes (both inserted and deleted) were rejected. You cannot edit the document in this mode. Attempting to do so switches the view mode to **View All Changes**.

Note: All three actions are available only in a drop-down list in the **Track Changes** toolbar.

Managing Comments

A comment is marked in Author editor with a background color which can be configured for each user name.



Spring Flowers

Spring Time, the time of growth and renewal of new plant and animal life. Spring comes at different times in the North and South Hemispheres. Spring time in the Northern hemisphere is between March - May, and between September - November in the Southern hemisphere. Most flowering plants bloom during spring time. Therefore, flowers that bloom only during spring, Spring Flowers, bloom at different times in the two hemispheres.

Some of the flowers blooming in Spring are: Agapanthus, Amaryllis, Anemone, Apple blossom, Bird of Paradise, Brodea, Calla lily, Cherry Blossom, Co Forsythia, F Hyacinth, La Liatrus, Lilac y.

Commented by sorin, Fri Sep 03 04:37 PM 2010

This list of spring flowers is too long. Also it should be organized as an itemized list, the inline enumeration is not readable.

Figure 139: Manage Comments in Author Editor

You can manage comments using the following actions:

- **Add Comment...** - Allows you to insert a comment at the cursor position or on a specific selection of content. The action is available on the Author toolbar.
- **Edit Comment...** - Allows you to change an existing content. The action is available both on the Author toolbar and the contextual menu.
- **Manage Comments...** - Opens a dialog that allows you to manage all comments contained in the current document. You can cycle through comments, edit, and remove individual comments or all comments. The action is available on the Author toolbar.
- **Remove Comment(s)...** - Removes the comment at the cursor position or all comments found in the selected content. The action is available on the Author contextual menu, **Review** sub-menu.

Profiling / Conditional Text

Conditional text is a way to mark blocks of text meant to appear in some renditions of the document, but not in others. It differs from one variant of the document to another, while unconditional text appear in all document versions.

For instance you can mark a section of a document to be included in the manual designated for the *expert* users, other for the *novice* users manual while unmarked sections are included in any rendition.

You can use conditional text when you develop documentation for:

- a series of similar products
- different releases of a product
- various audiences

The benefits of using conditional text include reduced effort for updating and translating your content and an easy way to customize the output for various audiences.

Oxygen XML Editor plugin comes with a preconfigured set of profiling attribute values for some of the most popular document types. These attributes can be redefined to match your specific needs. Also, you can define your own profiling attributes for a custom document type.

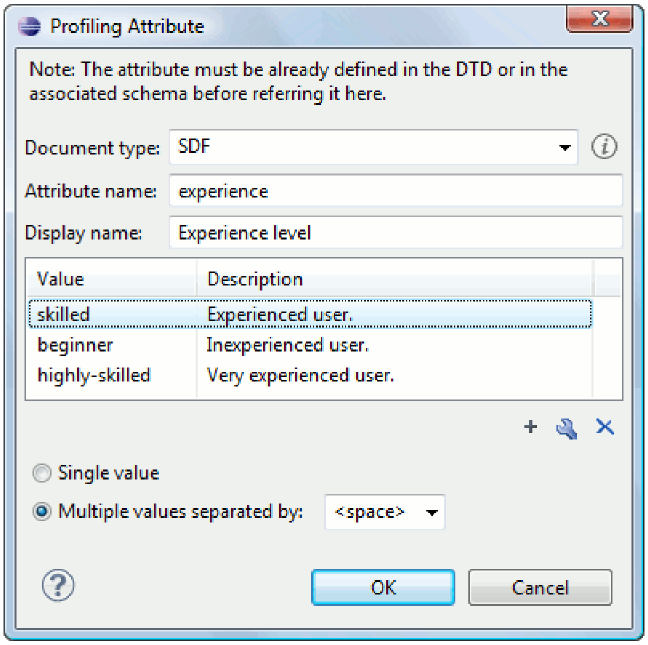
Create Profiling Attributes

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

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

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

   The following dialog is opened:

   Note: The attribute must be already defined in the DTD or in the associated schema before referring it here.

   Document type: SDF

   Attribute name: experience

   Display name: Experience level

Value	Description
skilled	Experienced user.
beginner	Inexperienced user.
highly-skilled	Very experienced user.

   ○ Single value

   ● Multiple values separated by: <space>

3. Fill-in the dialog as follows:
   a) Choose the document type on which the profiling attribute is applied. ***** and **?** are used as wildcards, while **,**(comma character) can be used to specify more patterns. For example use *DITA** to match any document type name that starts with *DITA*.
   b) Set the attribute name.
   c) Set a display name. This field is optional, being used only as a more descriptive rendering in application's profiling dialogs.
   d) Use the **New**, **Edit**, **Delete** buttons to add, edit and delete possible values of the attribute. Each attribute value can have a description.
   e) Choose whether the attribute accepts a single value (**Single value** option checked) or multiple values. Multiple values can be separated by a default delimiter (*space*, *comma*, *semicolon*), or a custom one, that must be supported by the specified document type. For example, the DITA document type only accepts spaces as delimiters for attribute values.
4. Click **OK**.
5. Click **Apply** to save the profiling attribute.

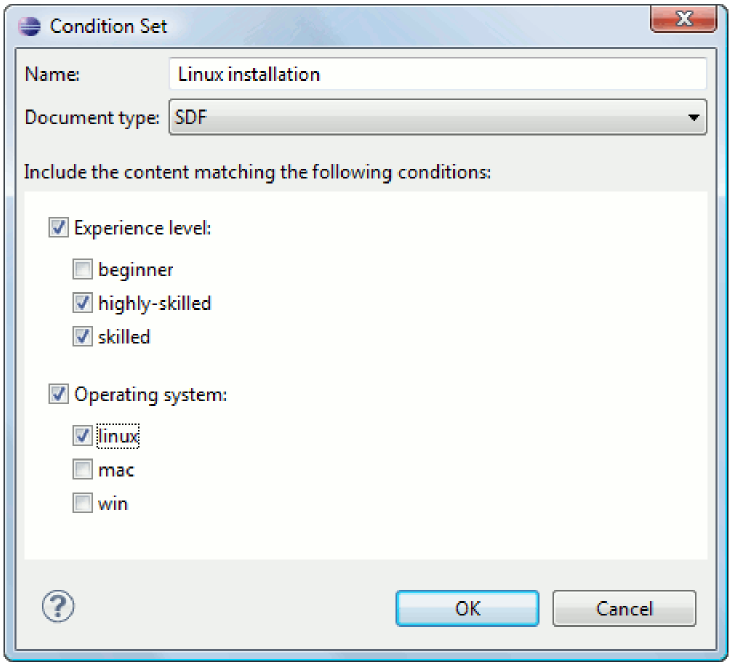
Create Profiling Condition Sets

Several profiling attributes can be aggregated into a profiling condition set that allow you to apply more complex filters on the document content. A Profiling Condition Set is a very powerful and convenient tool used to preview the content that goes into the published output. For example, an installation manual available both in Windows and Linux variants can be profiled to highlight only the Linux procedures for more advanced users.

To create a new profiling condition set:

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

   The following dialog is opened:



Condition Set

Name: Linux installation

Document type: SDF

Include the content matching the following conditions:

☑ Experience level:

- □ beginner
- ☑ highly-skilled
- ☑ skilled

☑ Operating system:

- ☑ linux
- □ mac
- □ win

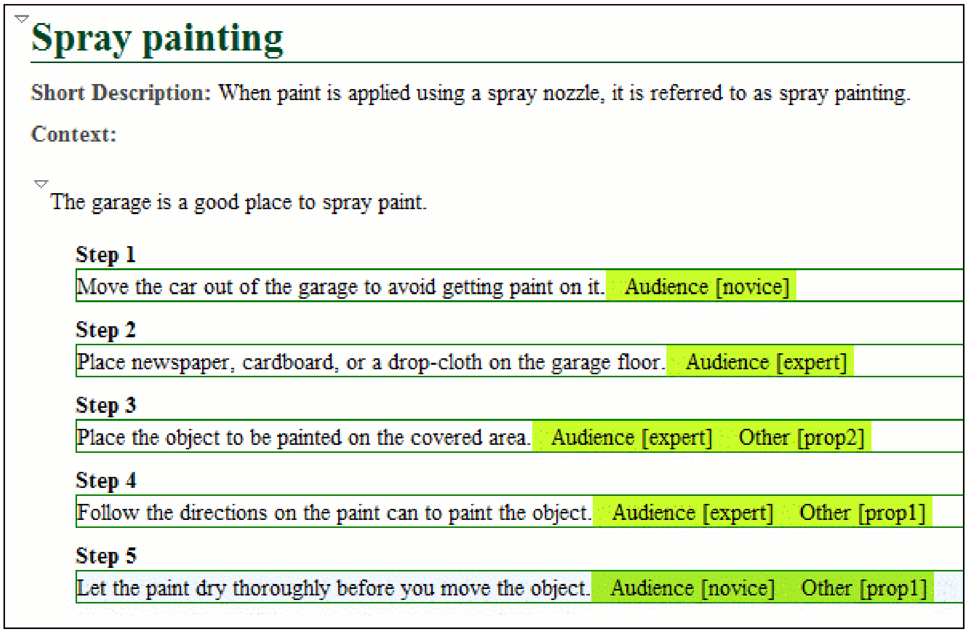
3. Fill-in the dialog as follows:
   a) Type the condition set's name.
   b) Choose the document type for which you have previously defined profiling attributes.
   After choosing a document type, all profiling attributes and their possible values are listed in the central area of the dialog.
   c) Define the combination of attribute values by ticking the appropriate checkboxes.
4. Click **OK**.
5. Click **Apply** to save the condition set. All saved profiling condition sets are available in the ***Profiling / Conditional Text*** *toolbar menu*.

Apply Profiling Condition Sets

All defined Profiling Condition Sets are available as shortcuts in the Profiling / Conditional Text menu. Just click on a menu entry to apply the condition set. The filtered content is grayed-out.

An element is filtered-out when one of its attributes is part of the condition set and its value does not match any of the value covered by the condition set.

As an example, let us suppose that you have the following document:



Spray painting

Short Description: When paint is applied using a spray nozzle, it is referred to as spray painting.

Context:

The garage is a good place to spray paint.

Step 1

Move the car out of the garage to avoid getting paint on it. Audience [novice]

Step 2

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

Step 3

Place the object to be painted on the covered area. Audience [expert] Other [prop2]

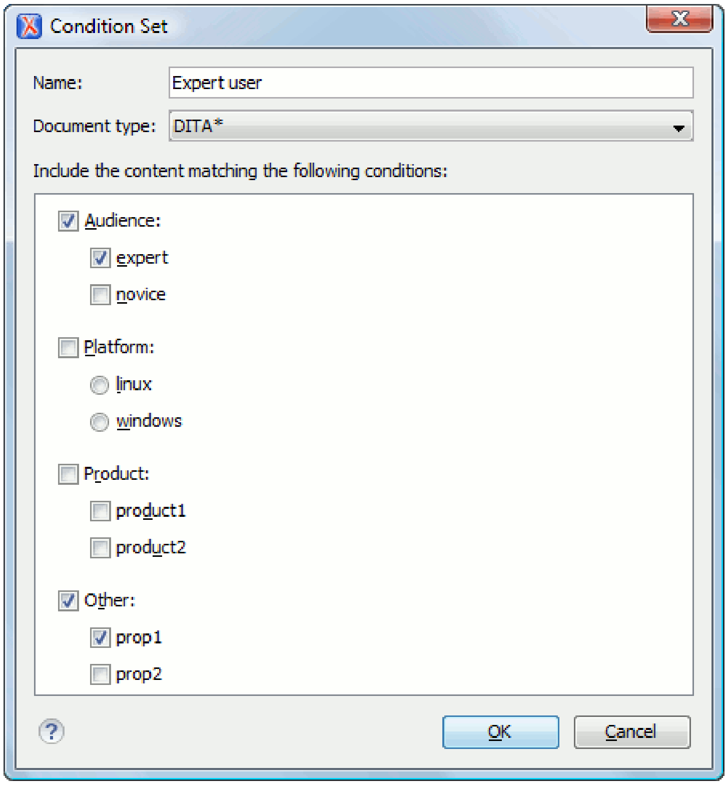
Step 4

Follow the directions on the paint can to paint the object. Audience [expert] Other [prop1]

Step 5

Let the paint dry thoroughly before you move the object. Audience [novice] Other [prop1]

If you apply the following condition set it means that you want to filter-out the content written for non-expert audience and having the *Other* attribute value different than *prop1*.



Condition Set

Name: Expert user

Document type: DITA*

Include the content matching the following conditions:

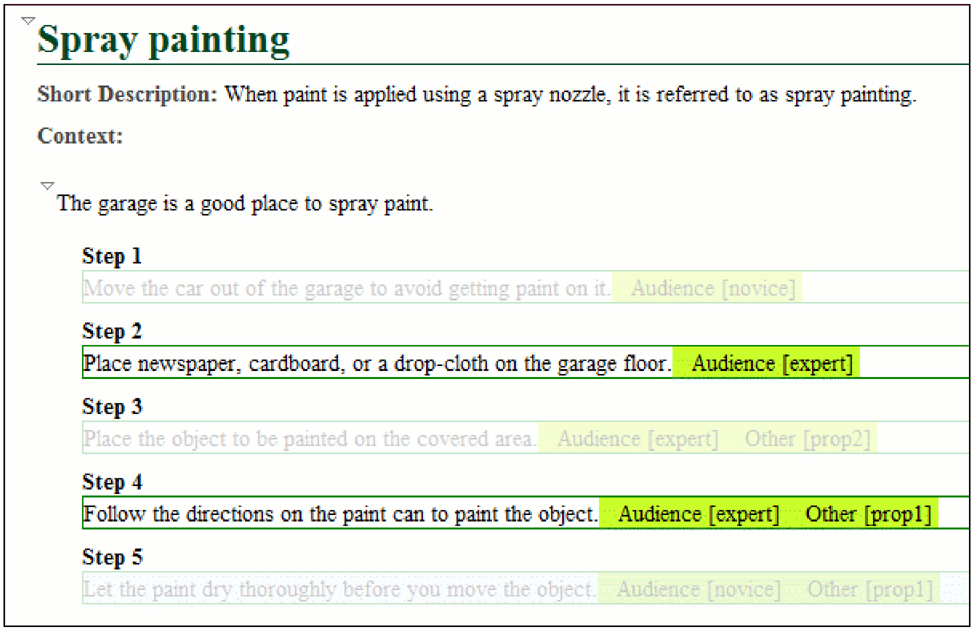
☑ Audience:
- ☑ expert
- ☐ novice

☐ Platform:
- ○ linux
- ○ windows

☐ Product:
- ☐ product1
- ☐ product2

☑ Other:
- ☑ prop1
- ☐ prop2

And this is how the document looks like after you apply the *Expert user* condition set:



Spray painting

Short Description: When paint is applied using a spray nozzle, it is referred to as spray painting.

Context:

The garage is a good place to spray paint.

Step 1

Move the car out of the garage to avoid getting paint on it. Audience [novice]

Step 2

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

Step 3

Place the object to be painted on the covered area. Audience [expert] Other [prop2]

Step 4

Follow the directions on the paint can to paint the object. Audience [expert] Other [prop1]

Step 5

Let the paint dry thoroughly before you move the object. Audience [novice] Other [prop1]

Apply Profiling Attributes

Profiling attributes are applied on element nodes.

Profiling attributes can be applied on a text fragment, a single element or on multiple elements in the same time. If you want to profile a fragment from your document, select the fragment in **Author** mode and follow the next steps.

Note: If there is no selection in your document, the profiling attributes are applied on the element at caret position.

1. Invoke the **Edit Profiling Attributes...** action from the contextual menu.

   The displayed dialog shows all profiling attributes and their values, as defined on the document type of the edited content. The checkboxes corresponding with the values already set in the profiled fragment are checked.

2. In the **Edit Profiling Attributes** dialog, tick the checkboxes corresponding to attribute values you want to apply on the document fragment. The profiling attributes having different values set in the elements of the profiled fragment are marked with a gray background and they are disabled by default. You can change the values of these attributes by choosing the **Change Now** option associated with all attributes.
3. Click **OK** to finish the profiling configuration.

   The attributes and attributes values selected in the **Edit Profiling Attributes** dialog are set on the elements contained in the profiled fragment.

   If you select only a fragment of an element's content, this fragment is wrapped in phrase-type elements on which the profiling attributes are set. Oxygen XML comes with predefined support for DITA and Docbook. For more developer-level customization options, see the *Customize Profiling Conditions* topic.

   If **Show Profiling Attributes** option (available in the ***Profiling / Conditional Text** toolbar menu*) is set, a light green border is painted around profiled text, in the **Author** mode. Also, all profiling attributes set on the current element are listed at the end of the highlighted block and in its tooltip message.

Profiling / Conditional Text Menu

The **Profiling / Conditional Text** toolbar menu groups the following actions:

- **Show Profiling Attributes** - Enable this option to turn on conditional text markers. They are displayed at the end of conditional text block, as a list of attribute name and their currently set values.

- The list of all profiling condition sets that match the current document type. Click on a condition set entry to activate it.
- **Configure Profiling Condition Sets...** - Link to the *Profiling / Conditional Text* preference page, where you can manage profiling attributes and profiling condition sets.

Smart Paste Support

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

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

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

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

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

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

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

- Inserting the converted fragment at the correct location in the document. This part is schema driven

  This capability is controlled by the *Smart paste and drag and drop* option, available in the **Schema Aware** preferences.

Chapter 6

Author for DITA

Topics:

This chapter presents the Author features that are specific for editing DITA XML documents.

Creating DITA Maps and Topics

The basic building block for DITA information is the DITA topic. DITA provides the following topic types:

- *Concept* - For general, conceptual information such as a description of a product or feature.
- *Task* - For procedural information such as how to use a dialog.
- *Reference* - For reference information.

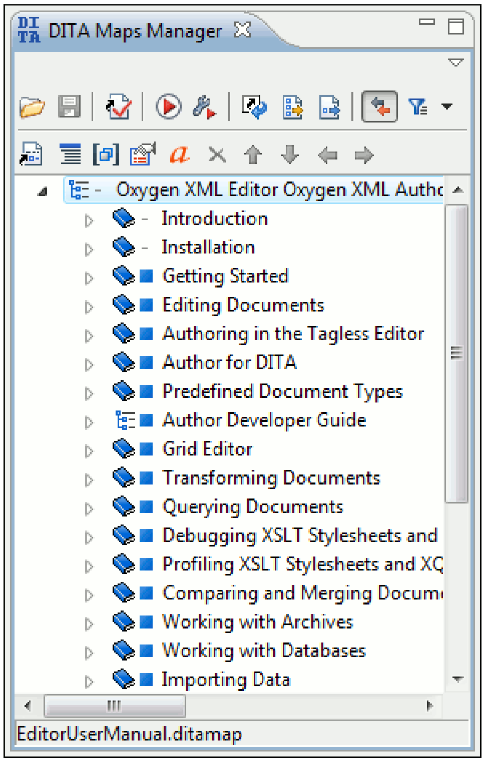
You can organize topics into a DITA map or bookmap. A map is a hierarchy of topics. A bookmap supports also book divisions such as chapters and book lists such as indexes. Maps do not contain the content of topics, but only references to them. These are known as topic references. Usually the maps and bookmaps are saved on disk or in a CMS with the extension '.ditamap'.

Maps can also contain relationship tables that establish relationships between the topics contained within the map. Relationship tables are also used to generate links in your published document.

You can use your map or bookmap to generate a deliverable using an output type such as XHTML, PDF, HTML Help or Eclipse Help.

Editing DITA Maps

Oxygen XML Editor plugin provides a special view for editing DITA maps. The **DITA Maps Manager** view presents a map in a simplified table-of-contents manner allowing the user to navigate easily to the referred topics and maps, make changes and perform transformations to various output formats using the DITA-OT framework bundled with Oxygen XML Editor plugin .



DITA Maps Manager

- Oxygen XML Editor Oxygen XML Authc
    - Introduction
    - Installation
    - Getting Started
    - Editing Documents
    - Authoring in the Tagless Editor
    - Author for DITA
    - Predefined Document Types
    - Author Developer Guide
    - Grid Editor
    - Transforming Documents
    - Querying Documents
    - Debugging XSLT Stylesheets and
    - Profiling XSLT Stylesheets and XQ
    - Comparing and Merging Docum
    - Working with Archives
    - Working with Databases
    - Importing Data

EditorUserManual.ditamap

Figure 140: The DITA Maps Manager View

You can open a map file from **Project** in the **DITA Maps Manager** view by right clicking it and choosing **Open in DITA Maps Manager**. The titles of the referenced resources are resolved and displayed in the view dynamically when navigating the tree. After the map is opened in the view, you can open it in the main editor for further customization using the **Open map in editor** toolbar action.

Drag and drop operations are allowed inside the view. You can move topics inside the same map or between different maps by dragging and dropping them into the desired position. Also, you can copy topics by dragging them while pressing the **Ctrl** (**Meta** on Mac OS) key.

The toolbar includes the actions which are also available on menu **DITA Maps**:

- **Open** - Allows opening the map in the **DITA Maps Manager** view. You can also open a map by dragging it in the **DITA Maps Manager** view from the file system explorer.
- **Open URL** - Allows opening remote maps in the **DITA Maps Manager** view. See *Open URL* for details.
- **Save (Ctrl+S)** - Saves the current DITA map.
- **Validate and Check for Completeness** - *Checks the validity and integrity* of the map.
- **Apply Transformation Scenario** - *Applies the DITA map transformation scenario* that is associated with the current map from the view.
- **Configure Transformation Scenario** - Allows *associating a DITA map transformation* scenario with the current map.
- **Refresh References** - You can use this action to manually trigger a refresh and update of all titles of referred topics. This action is useful when the referred topics are modified externally. When they are modified and saved from the Oxygen XML Editor plugin Author, the DITA Map is updated automatically.
- **Open Map in Editor with Resolved Topics** - Opens the result of expanding all topic references in Author editor.
- **Open Map in Editor** - For complex operations which cannot be performed in the simplified DITA maps view (like editing a relationship table) you can open the map in the main editing area.
- **Link with Editor** - Disables/Enables the synchronization between the file path of the current editor and the selected topic reference in the **DITA Maps Manager** view.
- **Profiling/Conditional Text** menu with the following actions:
  - **Show Profiling Attributes** - Enables/Disables displaying the values of the profiling attributes at the end of the titles of topic references. When enabled, the values of the profiling attributes are displayed both in the **DITA Maps Manager** view and in the **Author** editor.
  - **Configure Profiling Condition Sets ...** - Opens the preferences panel for adding and editing the profiling conditions that can be applied in the **DITA Maps Manager** view and the **Author** editor.

Tip: If your map references other DITA maps they will be shown expanded in the DITA Maps tree and you will also be able to navigate their content. For editing you will have to open each referenced map in a separate editor. You can choose not to expand referenced maps in the **DITA Maps Manager** view or referenced content in the opened editors by unchecking the **Display referred content** checkbox available in the *Author preferences page*.

Tip: The additional edit toolbar can be shown by clicking the "Show/Hide additional toolbar" expand button located on the general toolbar.

The following edit actions can be performed on an opened DITA Map:

- **Insert Reference** - Inserts a reference to a topic file. You can find more details about this action in the *Inserting a Reference, a Key Definition, a Topic Set* topic.
- **Insert Topic Heading** - Inserts a topic heading. You can find more details about this action in the *Inserting a Topic Heading* topic.
- **Insert Topic Group** - Inserts a topic group. You can find more details about this action in the *Inserting a Topic Group* on page 214 topic.
- **Edit Properties** - Edit the properties of a selected node. You can find more details about this action in the *Edit Properties* on page 215 topic.
- **Edit Profiling Attributes** - Allows you to select the profiling attributes.

- **Edit Attributes** - Allows you to edit all the attributes of a selected node. You can find more details about this action in the *Attributes View* on page 180 topic.
- **Delete** - Deletes the selected node.
- **Move Up (Alt + Up)** - Moves the selected nodes in front of their respective previous siblings.
- **Move Down (Alt + Down)** - Moves the selected nodes after their next respective siblings.
- **Promote (Alt + Left)** - Moves the selected nodes after their respective parents as siblings.
- **Demote (Alt + Right)** - Moves the selected nodes as children to their respective previous siblings.

  Note: As an alternative to these actions, you can select one or multiple topics, then drag and drop them to the desired position inside the map.

Editing Actions

Important: References can be made either by using the `href` attribute or by using the new `keyref` attribute to point to a key defined in the map. Oxygen XML Editor plugin tries to resolve both cases. `keyref`s are solved relative to the current map.

The contextual menu contains, in addition to the edit actions described above, the following actions:

- **Open** - Opens in the editor the resources referred by the selected nodes.
- **Append Child/Insert After** - Sub-menus containing the following actions:
  - **Reference** - Appends/Inserts a topic reference as a child/sibling of the selected node.
  - **Reference to the current edited file** - Appends/Inserts a topic reference to the current edited file as a child/sibling of the selected node.
  - **New topic** - Create a new topic from templates, saves it on disk and adds it into the DITA map.
  - **Anchor Reference**, **Key Definition**, **Map Reference**, **Topic Reference**, **Topic Set**, **Topic Set Reference** - Allows you to insert a reference to a topic file, a map file, a topic set, or a key definition.
  - **Topic heading** - Appends/Inserts a topic heading as a child/sibling of the selected node.
  - **Topic group** - Appends/Inserts a topic group as a child/sibling of the selected node.
- **Check Spelling in Files** - Checks the spelling of the files in the scope of the current edited map.
- **Search References** - Finds in the current map all references to the selected topic reference (`topicref` or `mapref` element) and to each element with an ID attribute contained in the selected topic. If the current selection in the **DITA Maps Manager** view is a `keydef` element the action will find any element with an attribute `idref`, `keyref`, `conref` or `conkeyref` that points to the selected `keydef`.
- **Cut, Copy, Paste, Undo, Redo** - Common edit actions with the same functionality as those found in the text editor.
- **Paste Before, Paste After** - Pastes the content of the clipboard before, respectively after, the selected node.

You can also arrange the nodes by dragging and dropping one or more nodes at a time. Drop operations can be performed before, after or as child of the targeted node. The relative location of the drop is indicated while hovering the mouse over a node before releasing the mouse button for the drop.

Drag and drop operations allow you to:

- **Copy** - Select the nodes you want to copy and start dragging them. Before dropping them in the appropriate place, press and hold the **Ctrl** key (**Meta** key on Mac). The mouse pointer changes to indicate that a copy operation is performed.
- **Move** - Select the nodes you want to move and drag and drop them in the appropriate place.
- **Promote Alt+Left Arrow / Demote Alt+Right Arrow** - You can move nodes between child and parent nodes which ensures both **PromoteAlt+Left Arrow** and **DemoteAlt+Right Arrow** operations.

Tip:

You can open and edit linked topics easily by double clicking the references or by right-clicking and choosing **Open in editor**. If the referenced file does not exist you are allowed to create it.

By right clicking the map root element you can open and edit it in the main editor area for more complex operations.

You can decide to open the reference directly in the Author mode and keep this setting as a default.

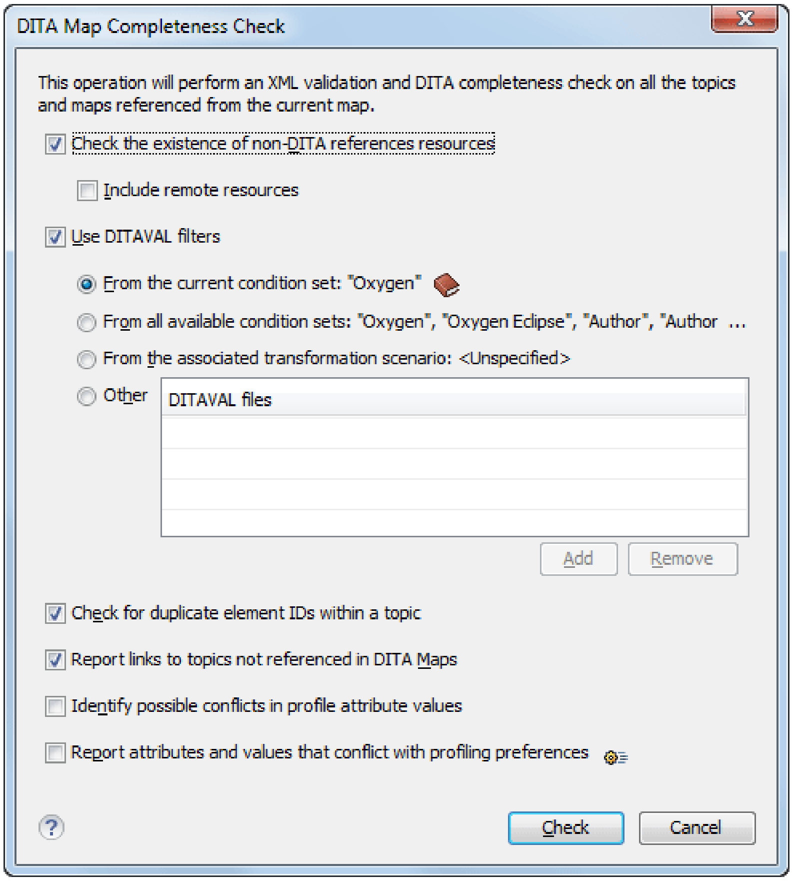
Creating a Map

Here are the steps to create a DITA map are the following:

1. Go to menu **File** > **New** or click on the **New** toolbar button.
2. Select one of the **DITA Map** templates on the tab **From templates** of the **New** dialog.
3. Click the **OK** button.
   A new tab is added in the **DITA Maps Manager** view.
4. Press the **Save** button on the toolbar of the **DITA Maps Manager** view.
5. Select a location and a file name for the map in the **Save As** dialog.

Validating DITA Maps

The validation of DITA maps is done with the action **Validate and Check for Completeness** that is available on *the **DITA Maps Manager** view* toolbar and on the **DITA Maps** menu.



DITA Map Completeness Check

This operation will perform an XML validation and DITA completeness check on all the topics and maps referenced from the current map.

☑ Check the existence of non-DITA references resources

□ Include remote resources

☑ Use DITAVAL filters

● From the current condition set: "Oxygen"

○ From all available condition sets: "Oxygen", "Oxygen Eclipse", "Author", "Author ...

○ From the associated transformation scenario: <Unspecified>

○ Other

DITAVAL files

☑ Check for duplicate element IDs within a topic

☑ Report links to topics not referenced in DITA Maps

□ Identify possible conflicts in profile attribute values

□ Report attributes and values that conflict with profiling preferences

Figure 141: DITA Map Completeness Check

The validation process does the following:

- Checks the file paths of the topic references. If a `href` attribute points to an invalid file path it is reported as a separate error in the **Errors** view.

- Validate each referred topic and map. Each topic file is opened and validated against the appropriate DITA DTD. If other map is referred in the main map, it is checked recursively applying the same algorithm as for the main map.

You can customize the operation setting the following options:

- **Check the existence of non-DITA references resources** - extends the validation of referred resources to non-DITA files. You can also choose to include in the validation also the remote resources;
- **Use DITAVAL filters** - the content of the map is filtered by applying a *profiling condition set* before validation.

  - **From the current condition set** - the map is filtered using the condition set applied currently in the DITA Maps Manager view.
  - **From all available condition sets** - for each available condition set, the map content is filtered using the condition set before validation.
  - **From the associated transformation scenario** - the filtering condition set is specified explicitly as a DITAVAL file in the current transformation scenario associated with the DITA map.
  - **Other DITAVAL files** - for each DITAVAL file, the map content is filtered using the DITAVAL file before validation.

  If a link is invalid in the content that resulted from the filtering process, then it is reported as error.
- **Check for duplicate element IDs within a topic** - if an ID is duplicated after assembling all topics referred in the map, it is reported as error.
- **Report links to topics not referenced in DITA Maps** - checks that all referred topics are linked in the DITA map.
- **Identify possible conflicts in profile attribute values** - when a topic's profiling attributes contain values that are not found in parent topics profiling attributes, the content of the topic is overshadowed when generating profiled output. This option reports such possible conflicts.
- **Report attributes and values that conflict with profiling preferences** - looks for profiling attributes and values not defined in the *Profiling / Conditional Text* preferences page. It also checks if profiling attributes defined as *single-value* have multiple values set in the searched topics.

Create a Topic in a Map

You add a new topic to a DITA map with the following steps:

1. Run the action **Insert Topic Reference** in the view **DITA Maps Manager**.

   The action **Insert Topic Reference** is available on the toolbar and on the contextual menu of the view. The action is available both on the submenu **Append Child** (when you want to insert a topic reference in a map as a child of the current topic reference) and on the submenu **Insert After** (when you want to insert it as a sibling of the current topic reference). The toolbar action is the same as the action from the submenu **Insert After**.
2. Select a topic file in the file system dialog called **Insert Topic Reference**.
3. Press the **Insert** button or the **Insert and close** button in the dialog.
   A reference to the selected topic is added to the current map in the view.
4. If you clicked the **Insert** button you can continue inserting new topic references using the *Insert* button repeatedly in the same file system dialog.
5. Close the dialog using the **Close** button.

Organize Topics in a Map

You can understand better how to organize topics in a DITA map by working with a populated map. You should open the sample map called `flowers.ditamap`, located in the `samples/dita` folder.

1. Open the file `flowers.ditamap`.
2. Select the topic reference *Summer Flowers* and press the ⬇ **Move Down** button to change the order of the topic references *Summer Flowers* and *Autumn Flowers*.
3. Make sure that *Summer Flowers* is selected and press the ➡ **Demote** button. This topic reference and all the nested ones are moved as a unit inside the *Autumn Flowers* topic reference.
4. Close the map without saving.

Create a Bookmap

The procedure for creating a bookmap is similar with that for creating a map.

1. Go to menu **File** > **New** or click on the **New** toolbar button.
   This action will open *the **New** wizard*.
2. Select the **DITA Map - Bookmap** template.
3. Click the **OK** button.
   A new tab with the new bookmap is added in *the **DITA Maps Manager** view*.
4. Press the **Save** button on the toolbar of the **DITA Maps Manager** view.
5. In the **Save As** dialog select a location and a file name for the map.

Create a Subject Scheme

The procedure for creating a DITA subject scheme is similar with that for creating a map.

1. Go to menu **File** > **New** or click on the **New** toolbar button.
   This action will open *the **New** wizard*.
2. Select the **DITA Map - Subject Scheme** template.
3. Click the **OK** button.
   A new tab with the new subject scheme document is added in *the **DITA Maps Manager** view*.
4. Press the **Save** button on the toolbar of the **DITA Maps Manager** view.
5. In the **Save As** dialog select a location and a file name for the map.

Create Relationships Between Topics

The DITA map offers the possibility of grouping different types of links between topics in a relationship table instead of specifying the links of each topic in that topic.

1. Open the DITA map file where you want to create the relationship table.

   Use the action **Open** that is available on the toolbar of the **DITA Maps Manager** view.
2. Place the cursor at the location of the relationship table.
3. Run the action **Insert a DITA reltable**.

   The action is available on the **Author** toolbar, on the menu **DITA** > **Table** and on the **Table** submenu of the contextual menu of the DITA map editor.

   This action displays the **Insert Relationship Table** dialog.
4. Set the parameters of the relationship table that will be created: the number of rows, the number of columns, a table title (optional), a table header (optional).
5. Press **OK** in the **Insert Table** dialog.
6. Set the type of the topics in the header of each column.

   The header of the table (the `relheader` element) already contains a `relcolspec` element for each table column. You should set the value of the attribute `type` of each `relcolspec` element to a value like *concept*, *task*, *reference*. When you click in the header cell of a column (that is a `relcolspec` element) you can see all the attributes of that `relcolspec` element including the `type` attribute in the **Attributes** view. You can edit the attribute type in this view.
7. Place the cursor in a table cell and run the action **Insert Topic Reference** for inserting a topic reference in that cell.

   The action is available on the **Author** toolbar, on the menu **DITA** > **Insert** and on the **Insert** submenu of the contextual menu.

8. Optionally for adding a new row to the table / removing an existing row you should run the action **Insert Row**/ **Delete Row.**

   The actions are available on the **Author** toolbar, on the menu **DITA** > **Table** and on the **Table** submenu of the contextual menu.

9. Optionally for adding a new column to the table / removing an existing column you should run the action **Insert Column**/ **Delete Column**.

   The actions are available on the **Author** toolbar, on the menu **DITA** > **Table** and on the **Table** submenu of the contextual menu.

Advanced Operations

This section explains how to insert references like chapter, topic reference, topic group or topic heading in a DITA map.

Inserting a Reference, a Key Definition, a Topic Set

A DITA map can contain various types of references. The targets of the references can be:

- an anchor
- a map
- a topic
- a topic set

The `topicref` element is a reference to a topic (such as a concept, task, or reference) or other resource. A `topicref` can contain other `topicref` elements, and allows you to express navigation or table-of-contents hierarchies, as well as implying relationships between the containing `topicref` and its children. You can set the collection type of a container `topicref` to determine how its children are related to each other. You can also express relationships among `topicref`'s using group and table structures (using `topicgroup` and `reltable`). Relationships end up expressed as links in the output (with each participant in a relationship having links to the other participants by default).

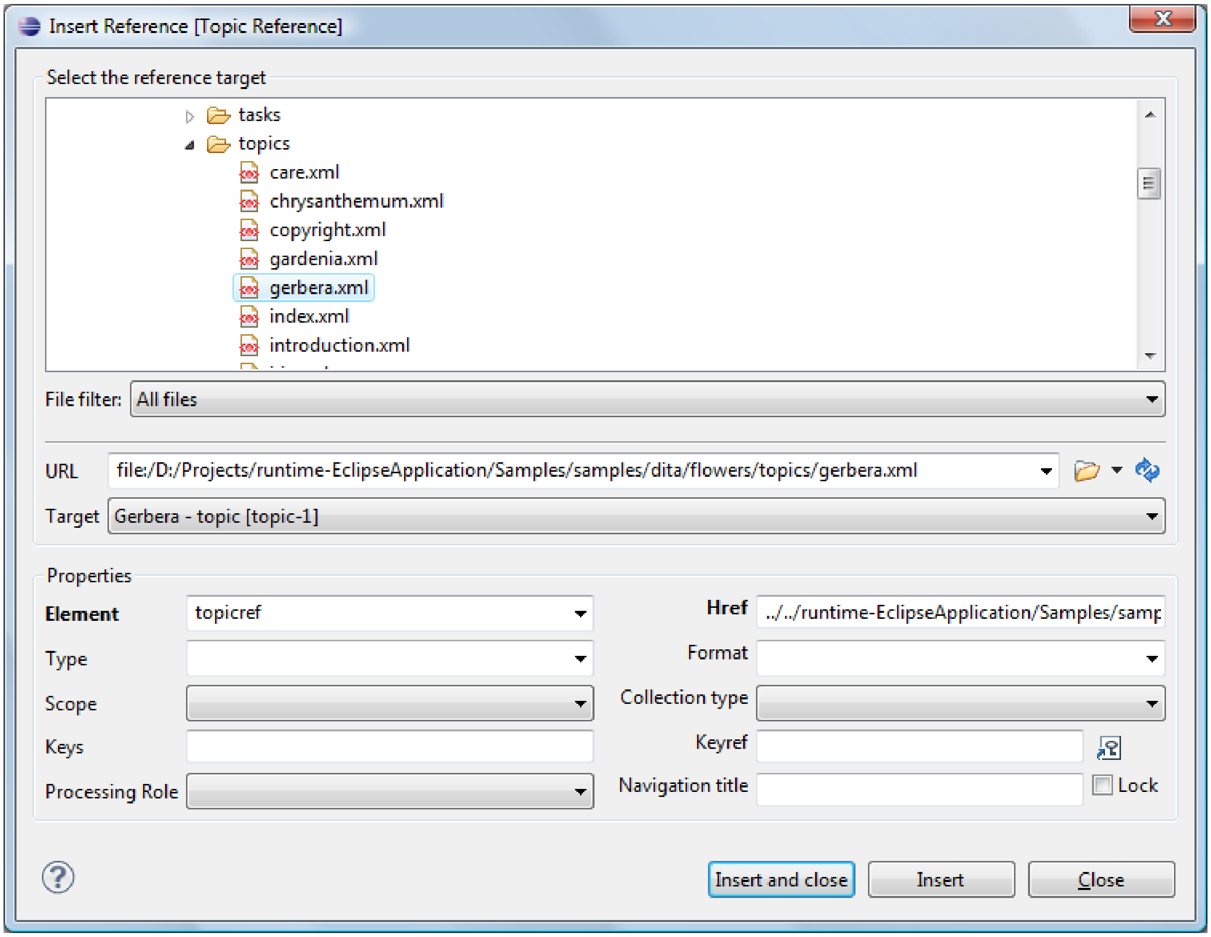
A reference to a topic file, a map file, a topic set, or a key definition may be inserted with the following dialog box which is opened from the actions of the **Append child** and **Insert after** submenus of *the **DITA Maps Manager** view*'s contextual menu. The content of the **Append child** and **Insert after** submenus depend on the selected node of the DITA map tree on which the contextual menu was invoked. For example if the selected node is the `bookmap` root node the possible child nodes are:

- chapter (the `chapter` element),
- part (the `part` element),
- appendix (the `appendix` element),
- appendices (the `appendices` element)

If the selected node is a `topicref` the possible child nodes are:

- anchor reference (the `anchorref` element),
- topic reference (the `topicref` element),
- map reference (the `mapref` element),
- topic set reference (the `topicsetref` element),
- topic set (the `topicset` element),
- key definition (the `keydef` element),
- topic head (the `topichead` element),
- topic group (the `topicgroup` element)

The same dialog box can be used to insert a non-DITA file like a PDF document.



File filter: All files

URL file:/D:/Projects/runtime-EclipseApplication/Samples/samples/dita/flowers/topics/gerbera.xml

Target Gerbera - topic [topic-1]

Element topicref

Type

Scope

Keys

Processing Role

Href ../../runtime-EclipseApplication/Samples/samp

Format

Collection type

Keyref

Navigation title

□ Lock

Figure 142: Insert Topic Reference Dialog

By using the **Insert Topic Reference** dialog you can easily browse for and select the source topic file. The **Target** combo box shows all available topics that can be targeted in the file. Selecting a target modifies the **Href** value to point to it which corresponds to the `href` attribute of the inserted `topicref` element. The **Format** and **Scope** combos are automatically filled based on the selected file and correspond to the `format` and `scope` attributes of the inserted `topicref` element. You can specify and enforce a custom navigation title by checking the **Navigation title** checkbox and entering the desired title.

The file chooser located in the dialog allows you to easily select the desired topic. The selected topic file will be added as a child or sibling of the current selected topic reference, depending on the insert action selected from the contextual menu of the **DITA Maps** view, that is an insert action from the **Append child** submenu or from the **Insert after** one. You can easily insert multiple topic references by keeping the dialog opened and changing the selection in the **DITA Maps Manager** tree. You can also select multiple resources in the file explorer and then insert them all as topic references.

Another easy way to insert a topic reference is to drag files from the **Project** view, file system explorer or **Data Source Explorer** view and drop them into the map tree.

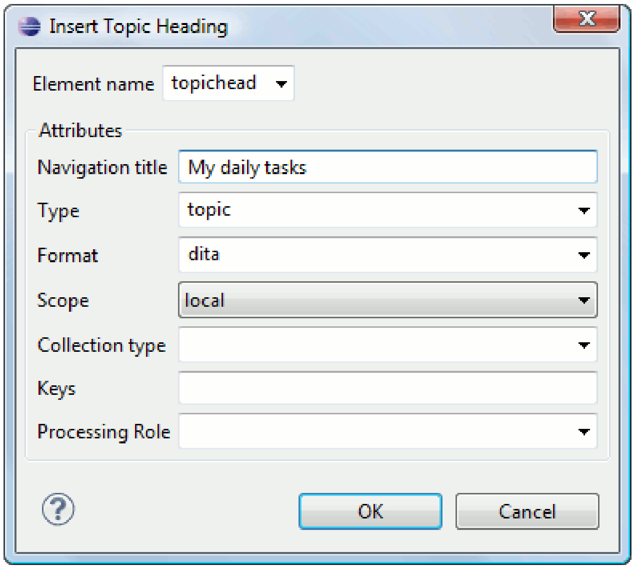
You can also define keys using the **Keys** text field on the inserted `topicref` or `keydef` element. Instead of using the **Href** combo box to point to a location you can reference a key definition using the **Keyref** text field. Use the **Choose key reference** to access the list of keys defined in the currently open DITA map.

The **Processing Role** combo box allows setting the `processing-role` attribute to one of the allowed values for DITA reference elements: `resource-only`, `normal`, `-dita-use-conref-target`.

Inserting a Topic Heading

The `topichead` element provides a title-only entry in a navigation map, as an alternative to the fully-linked title provided by the *topicref* element.

A topic heading can be inserted both from the toolbar action and the contextual node actions.



Insert Topic Heading

Element name: topichead

Attributes

Navigation title: My daily tasks

Type: topic

Format: dita

Scope: local

Collection type:

Keys:

Processing Role:

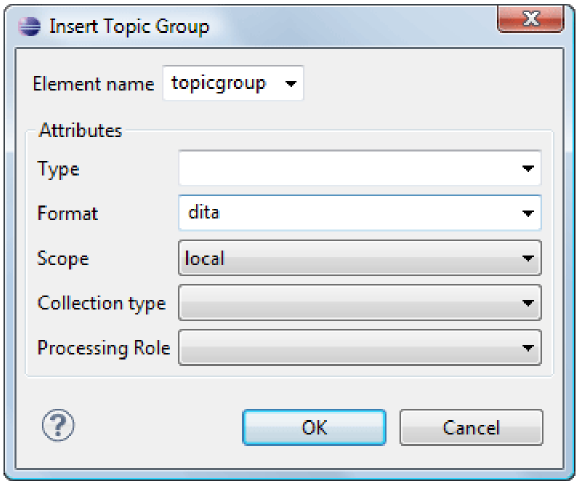
Figure 143: Insert Topic Heading Dialog

By using the **Insert Topic Heading** dialog you can easily insert a *topichead* element. The **Navigation title** is required but other attributes can be specified as well from the dialog.

Inserting a Topic Group

The *topicgroup* element identifies a group of topics (such as a concepts, tasks, or references) or other resources. A *topicgroup* can contain other *topicgroup* elements, allowing you to express navigation or table-of-contents hierarchies, as well as implying relationships between the containing *topicgroup* and its children. You can set the collection-type of a container *topicgroup* to determine how its children are related to each other. Relationships end up expressed as links in the output (with each participant in a relationship having links to the other participants by default).

A topic group may be inserted both from the toolbar action and the contextual node actions.



Insert Topic Group

Element name: topicgroup

Attributes

Type:

Format: dita

Scope: local

Collection type:

Processing Role:

Figure 144: Insert Topic Group Dialog

By using the **Insert Topic Group** dialog you can easily insert a *topicgroup* element. The **Type**, **Format**, **Scope** and **Collection type** attributes can be specified from the dialog.

Edit Properties

The **Edit properties** action, available both on the toolbar and on the contextual menu, is used to edit the properties of the selected node. Depending on the selected node, the action will perform the following tasks:

- If a *topicref* or *chapter* element is selected, the action will show a dialog similar with the ***Insert Topic Reference*** *dialog* allowing the editing of some important attributes.
- If a *topichead* element is selected, the action will show a dialog similar with the ***Insert Topic Heading*** *dialog* allowing the editing of some important attributes.
- If a *topicgroup* element is selected, the action will show a dialog similar with the ***Insert Topic Group*** *dialog* allowing the editing of some important attributes.
- If the map's root element is selected then the user will be able to easily edit the map's title using the **Edit Map title** dialog. By using this dialog you can also specify whether the title will be specified as the *title* attribute to the map or as a *title* element (for DITA-OT 1.1 and 1.2) or specified in both locations.

Transforming DITA Maps and Topics

Oxygen XML Editor plugin uses the DITA Open Toolkit (DITA-OT) to transform DITA maps and topics into an output format. For this purpose both the DITA Open Toolkit 1.5.4 and ANT 1.7 come bundled in Oxygen XML Editor plugin .

More information about the DITA Open Toolkit are available at *http://dita-ot.sourceforge.net/*.

Available Output Formats

You can publish DITA-based documents in any of the following formats:

- **PDF** - DITA to PDF using the DITA OT IDIOM PDF plugin.
- **WebHelp** - DITA to XHTML.
- **XHTML** - DITA to XHTML.
- **Compiled HTML Help (CHM)** - DITA Map to HTML Help. If HTML Help Workshop is installed on your computer, then Oxygen XML Editor plugin detects it and uses it to perform the transformation. When the transformation fails, the hhp (HTML Help Project) file is already generated and it must be compiled to obtain the *.chm* file. Note that HTML Help Workshop fails when the files used for transformation contain diacritics in their names, due to different encodings used when writing the *.hhp* and *.hhc* files.
- **JavaHelp** - DITA Map to JavaHelp.
- **Eclipse Help** - DITA Map to Eclipse Help.
- **Eclipse Content** - DITA Map to Eclipse Content.
- **TocJS** - A JavaScript file that can be included in an HTML file to display in a tree-like manner the table of contents of the transformed DITA map.
- **Open Document Format** - DITA Map to ODF.
- **Docbook** - DITA Map to Docbook.
- **RTF** - DITA Map to Rich Text Format.
- **troff** - DITA Map to Text Processor for Typesetters.
- **Legacy PDF** - DITA Map to PDF using the DITA OT deprecated PDF implementation.

The *TocJS* Transformation

The *TocJS* transformation of a DITA map does not generate all the files needed to display the tree-like table of contents. To get a complete working set of output files you should follow these steps:

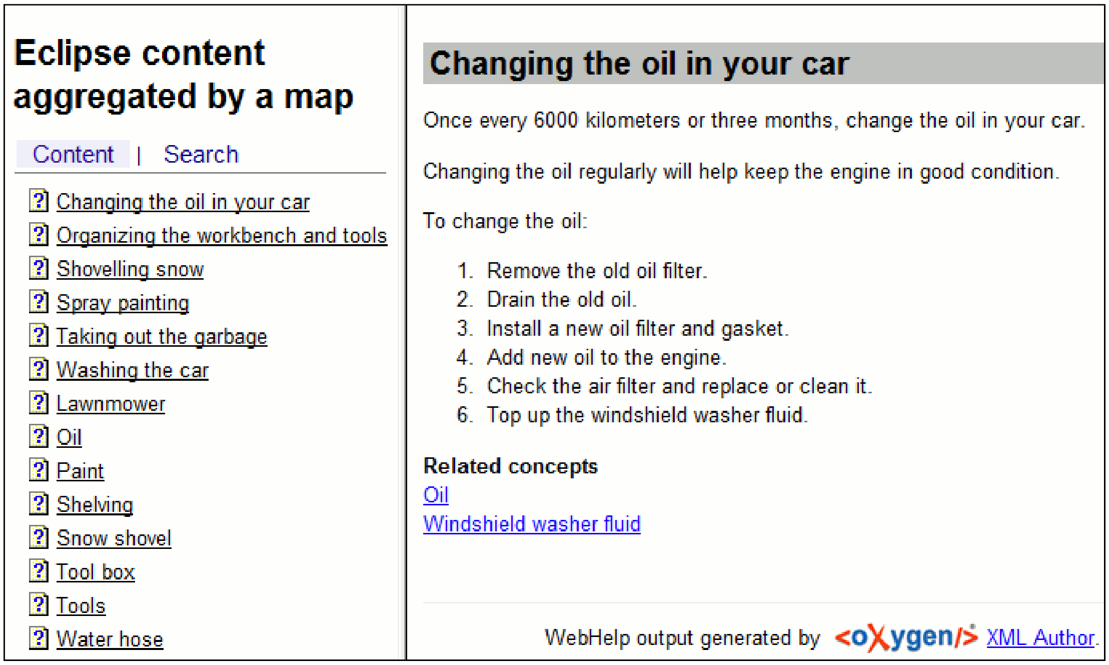
1. Run the *XHTML* transformation on the same DITA map. Make sure the output gets generated in the same output folder as for the *TocJS* transformation.
2. Copy the content of `${frameworks}/dita/DITA-OT/demo/tocjs/basefiles` folder in the transformation's output folder.

3. Copy the `${frameworks}/dita/DITA-OT/demo/tocjs/sample/basefiles/frameset.html` file in the transformation's output folder.
4. Edit `frameset.html` file.
5. Locate element `<frame name="contentwin" src="concepts/about.html">`.
6. Replace `"concepts/about.html"` with `"index.html"`.

WebHelp Output Format

WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include continuous content update and platform independence, since it can be viewed using a regular web browser.

Oxygen XML Editor plugin allows you to publish a DITA Map into a WebHelp format that provides both Table of Contents and advanced search capabilities.



Eclipse content aggregated by a map

Content | Search

- Changing the oil in your car
- Organizing the workbench and tools
- Shovelling snow
- Spray painting
- Taking out the garbage
- Washing the car
- Lawnmower
- Oil
- Paint
- Shelving
- Snow shovel
- Tool box
- Tools
- Water hose

Changing the oil in your car

Once every 6000 kilometers or three months, change the oil in your car.

Changing the oil regularly will help keep the engine in good condition.

To change the oil:

1. Remove the old oil filter.
2. Drain the old oil.
3. Install a new oil filter and gasket.
4. Add new oil to the engine.
5. Check the air filter and replace or clean it.
6. Top up the windshield washer fluid.

Related concepts
Oil
Windshield washer fluid

WebHelp output generated by <oXygen/> XML Author.

Figure 145: WebHelp Output

The layout is composed of two frames:

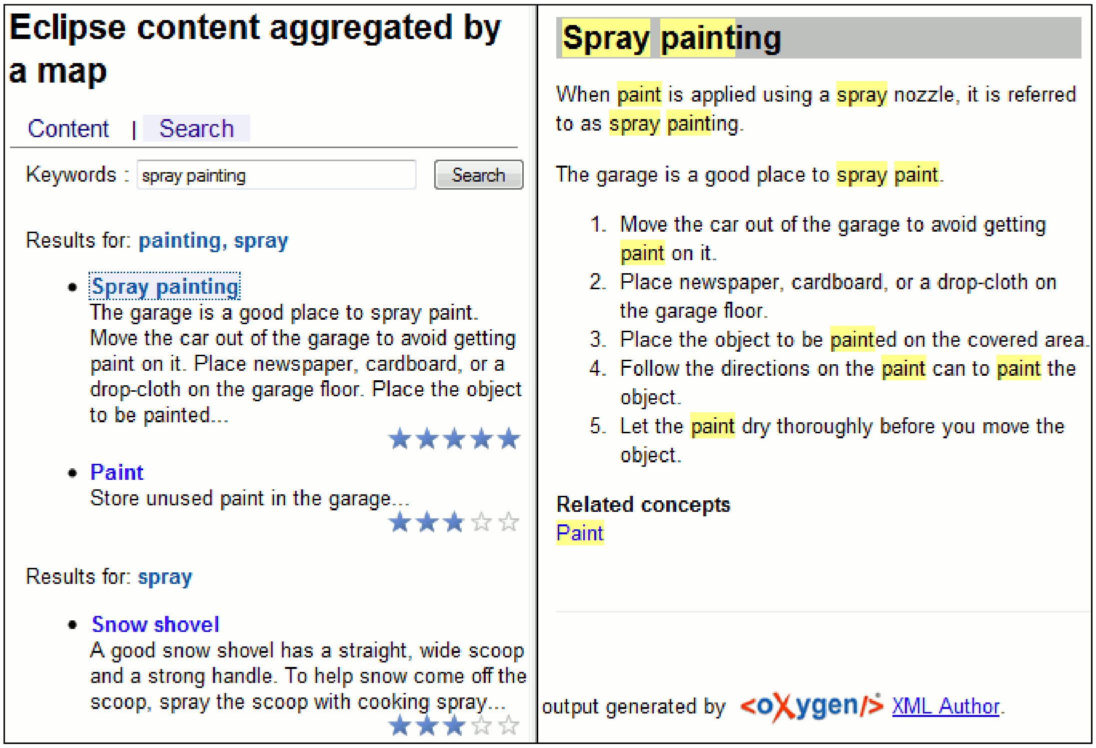
- left frame, containing separate tabs for **Table of Contents** and **Search**;
- central frame where help pages are displayed.

To publish the DITA map to WebHelp, you can use the **DITA Map WebHelp** transformation. You can further customize the out-of-the-box transformation, by editing some of its parameters:

- `args.xhtml.toc` - name of the table of contents file. Default setting is `toc.html`;
- `use.stemming` - controls whether or not you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. Default setting is `false`.
- `clean.output` - deletes all files from the output folder before the transformation is performed. Default setting is `no`.

The **Search** tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on:

- number of keywords found in a single page. The higher the number, the better.
- context - if a word is found in a title or emphasised section of text it scores better than a word found in a unformatted text.



Eclipse content aggregated by a map

Content | Search

Keywords : spray painting Search

Results for: painting, spray

- Spray painting
  The garage is a good place to spray paint. Move the car out of the garage to avoid getting paint on it. Place newspaper, cardboard, or a drop-cloth on the garage floor. Place the object to be painted...
  ★★★★★
- Paint
  Store unused paint in the garage...
  ★★★☆☆

Results for: spray

- Snow shovel
  A good snow shovel has a straight, wide scoop and a strong handle. To help snow come off the scoop, spray the scoop with cooking spray...
  ★★★☆☆

Spray painting

When paint is applied using a spray nozzle, it is referred to as spray painting.

The garage is a good place to spray paint.

1. Move the car out of the garage to avoid getting paint on it.
2. Place newspaper, cardboard, or a drop-cloth on the garage floor.
3. Place the object to be painted on the covered area.
4. Follow the directions on the paint can to paint the object.
5. Let the paint dry thoroughly before you move the object.

Related concepts

Paint

output generated by <oXygen/> XML Author.

Figure 146: WebHelp Search with Stemming Enabled

Rules applied during search:

- keywords are separated by the space character. An expression like *spray painting* counts as two separate keywords: *spray* and *painting*;
- do not use quotes to perform exact search for multiple-word expressions. An expression like *"spray painting"*, returns no results in our case, because it searches for two separate words: *"spray* and *painting"* (note the quote signs attached to each word);
- *indexterm* and *keywords* DITA elements are an effective way to increase the ranking of a page. For example, content inside *keywords* elements weighs twice as much as content inside a *H1* HTML element;
- words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters, count as a single word.

How to Localize WebHelp Output

Static labels used in the WebHelp output are kept in translation files in the `DITA-OT/xsl/common` folder. This folder is found either in the Oxygen XML Editor plugin installation folder, or elsewhere if you are using a different DITA-OT distribution. Translation files have the *strings-lang1-lang2.xml* name format, where *lang1* and *lang2* are ISO language codes. For example, the US English text is kept in the *strings-en-us.xml* file.

Follow these steps to localize the interface of the WebHelp output (for simplicity sake, let us suppose you want to localize the WebHelp interface into Canadian French.):

1. Look for the *strings-fr-ca.xml* file. If it does not exist, create one starting from *strings-en-us.xml*.
2. Translate the labels with the following names: *Content*, *Search*, *Keywords*, *Search no results*, *Output generated by*, *Next topic*, *Previous topic*, and *Parent topic*. Labels are stored in XML elements that have the following format: *<str name="Label name">Caption</str>*.
3. Edit the **DITA Map WebHelp** transformation scenario and set the *args.default.language* parameter to the code of the language you want to localize the interface into (in our case, it is *fr-ca*).
4. Run the transformation scenario to produce the WebHelp output.

Configuring a DITA Transformation

Creating map transformation scenarios is similar to creating scenarios in the main editing area.

The ***Configure Transformation Scenario** dialog* is opened from the toolbar action **Configure Transformation Scenario** of the ***DITA Maps Manager view***. Select as scenario type **DITA OT transformation** then press the **New** button. Next step involves choosing the type of output the DITA-OT ANT scenario will generate:

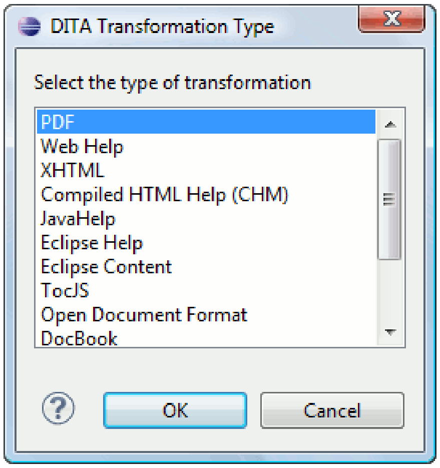


Figure 147: Select DITA Transformation type

Depending on the chosen type of output Oxygen XML Editor plugin generates values for the default ANT parameters so that you can execute the scenario right away without further customization.

Running a DITA Map ANT Transformation

The transformation is run as an external ANT process so you can continue using the application as the transformation unfolds. All output from the process appears in the **DITA Transformation** tab.

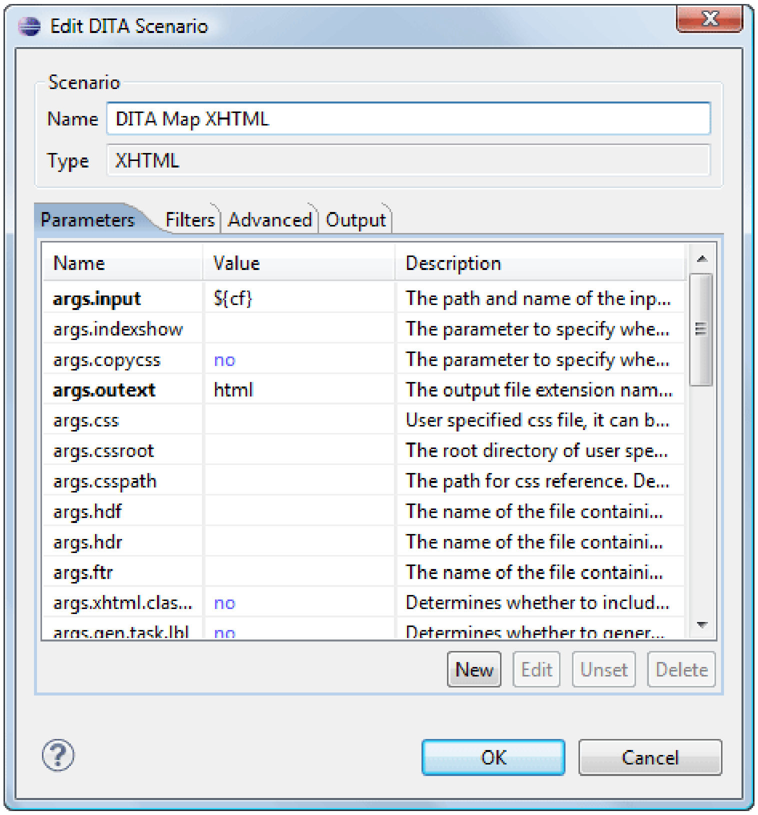
Tip: The HTTP proxy settings from Oxygen XML Editor plugin are also used for the ANT transformation so if the transformation fails because it cannot connect to an external location you can check the *HTTP/Proxy Configuration*.

Customizing a DITA Scenario

This section explains how to edit the parameters of a DITA transformation in the dialog box for configuring such a transformation.

The *Parameters* Tab

In the scenario **Parameters** tab you can customize all the parameters which will be sent to the DITA-OT build file.



Name: DITA Map XHTML

Type: XHTML

Name	Value	Description
args.input	${cf}	The path and name of the inp...
args.indexshow		The parameter to specify whe...
args.copycss	no	The parameter to specify whe...
args.outext	html	The output file extension nam...
args.css		User specified css file, it can b...
args.cssroot		The root directory of user spe...
args.csspath		The path for css reference. De...
args.hdf		The name of the file containi...
args.hdr		The name of the file containi...
args.ftr		The name of the file containi...
args.xhtml.clas...	no	Determines whether to includ...
args.gen.task.lbl	no	Determines whether to gener...

Figure 148: Edit DITA Ant transformation parameters

All the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (eg: XHTML) are listed along with their description. The values for some important parameters are already filled in. You can find more information about each parameter in the *DITA OT Documentation*. You can also add additional parameters to the list.

Using the toolbar buttons you can add, edit or remove a parameter.

Depending on the parameter type the parameter value will be:

- a simple text field for simple parameter values
- a combo box with some predefined values
- a file chooser and an editor variables selector to simplify setting a file path as value to a parameter

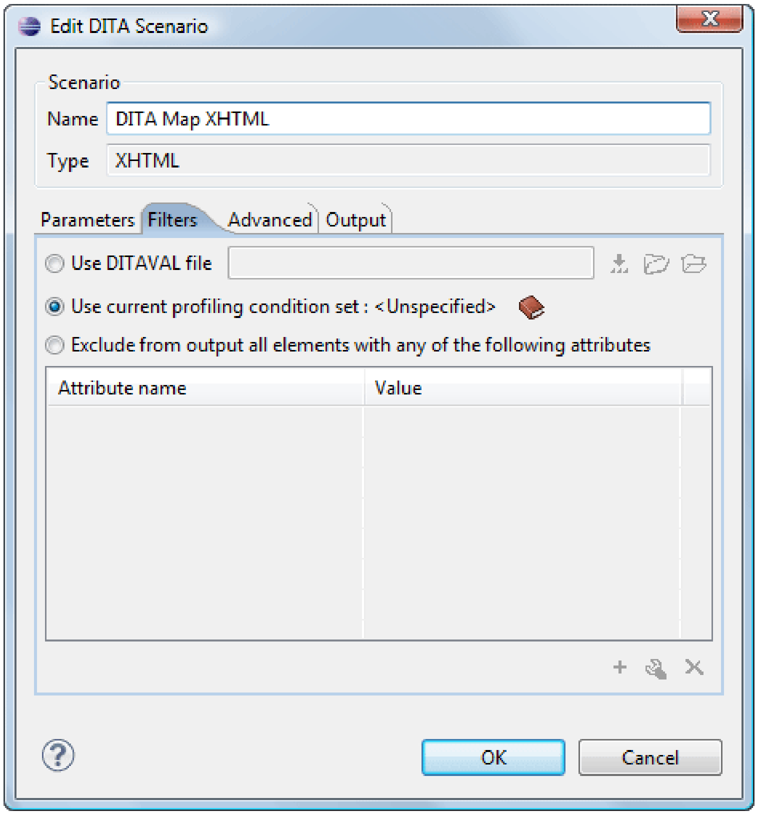
The value of a parameter can be entered at runtime if a value *ask('user-message', param-type, 'default-value' ?)* is used as value of parameter in the Configure parameters dialog.

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'`.

The *Filters* Tab

In the scenario **Filters** tab you can add filters to remove certain content elements from the generated output.



Edit DITA Scenario

Scenario

Name: DITA Map XHTML

Type: XHTML

○ Use DITAVAL file

● Use current profiling condition set : <Unspecified>

○ Exclude from output all elements with any of the following attributes

Attribute name	Value

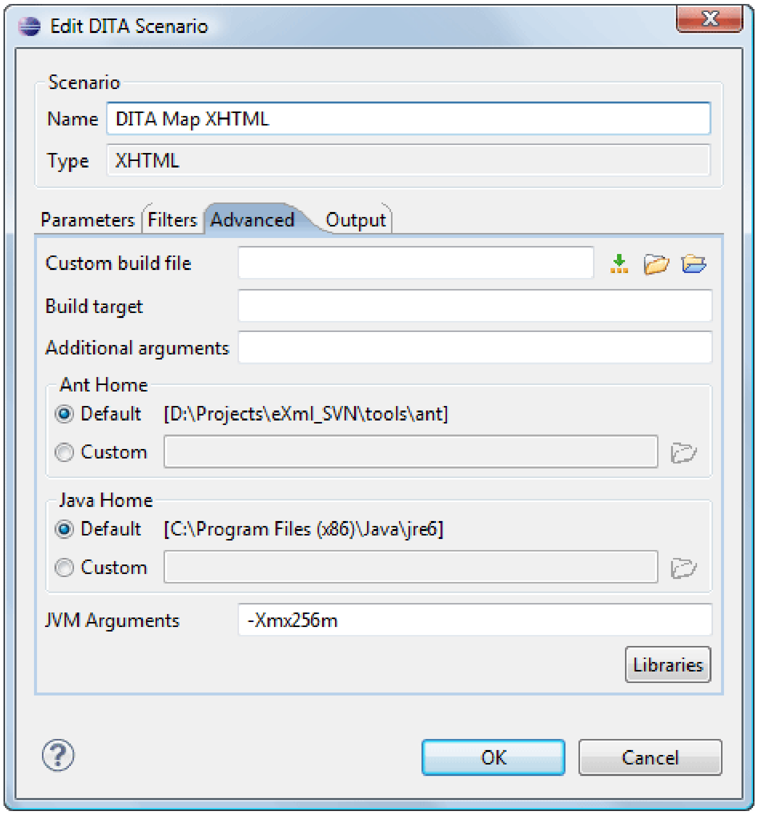
Figure 149: Edit Filters tab

There are three ways to define filters:

- **Use DITAVAL file** - If you already have a DITAVAL file associated with the transformed map you can specify the path to it and it will be used when filtering content. You can find out more about constructing a DITAVAL file in the *DITA OT Documentation* topic.
- **Use current profiling condition set** - Allows you to use the currently active *profiling condition set*.
- **Exclude from output all elements with any of the following attributes** - You can configure a simple list of attribute (name, value) pairs which when present on an element in the input will remove it from output.

The *Advanced* Tab

In the **Advanced** tab, you can specify advanced options for the transformation.



Scenario

Name: DITA Map XHTML

Type: XHTML

Custom build file:

Build target:

Additional arguments:

Ant Home

● Default [D:\Projects\eXml_SVN\tools\ant]

○ Custom

Java Home

● Default [C:\Program Files (x86)\Java\jre6]

○ Custom

JVM Arguments: -Xmx256m

Figure 150: Advanced settings tab

You have several parameters that you can specify here:

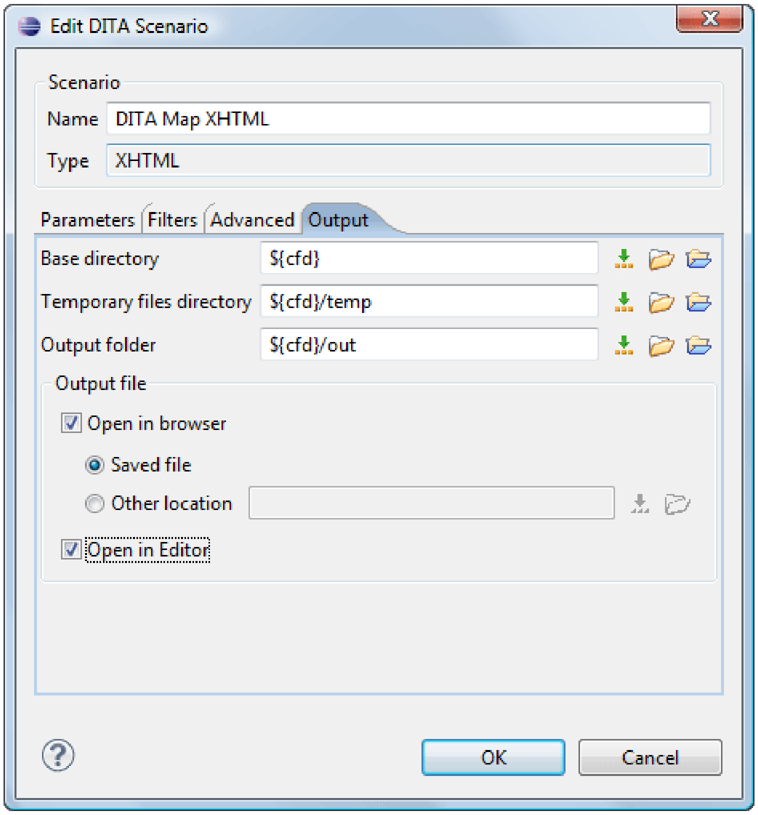
- **Custom build file** - If you use a custom DITA-OT build file you can specify the path to the customized build file. If empty, the `build.xml` file from the `dita.dir` directory configured in the **Parameters** tab is used.
- **Build target** - You can specify a build target to the build file. By default no target is necessary and the default `init` target is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the ANT transformation like -verbose.
- **Ant Home** - You can specify a custom ANT installation to run the DITA Map transformation. By default it is the ANT installation bundled with Oxygen XML Editor plugin .
- **Java Home** - You can specify a custom Java Virtual Machine to run the ANT transformation. By default it is the Java Virtual Machine used by Oxygen XML Editor plugin .
- **JVM Arguments** - This parameter allows you to set specific parameters to the Java Virtual Machine used by ANT. By default it is set to -Xmx256m which means the transformation process is allowed to use 256 megabytes of memory. Sometimes, when performing a large DITA map transformation you may want to increase the memory allocated to the Java Virtual Machine from the default value (256 MB) to a higher value, like 512 MB. In this way, you can avoid the Out of Memory error messages (**OutOfMemoryError**) received from the ANT process.

  Note: If you are publishing DITA to PDF and still experience problems, you should also increase the amount of memory allocated to the FO transformer. To do this, open the *Parameters tab* and increase the value of `maxJavaMemory` parameter (default value is *500*).

- **Libraries** - Oxygen XML Editor plugin adds by default as high priority libraries which are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can specify all the additional libraries (jar files or additional class paths) which are used by the ANT transformer. You can also decide to control all libraries added to the classpath.

The *Output* Tab

In the **Output** tab, you can configure options related to the place where the output is generated.



Edit DITA Scenario

Scenario

Name: DITA Map XHTML

Type: XHTML

Base directory: ${cfd}

Temporary files directory: ${cfd}/temp

Output folder: ${cfd}/out

Output file

☑ Open in browser

● Saved file

○ Other location

☑ Open in Editor

Figure 151: Output settings tab

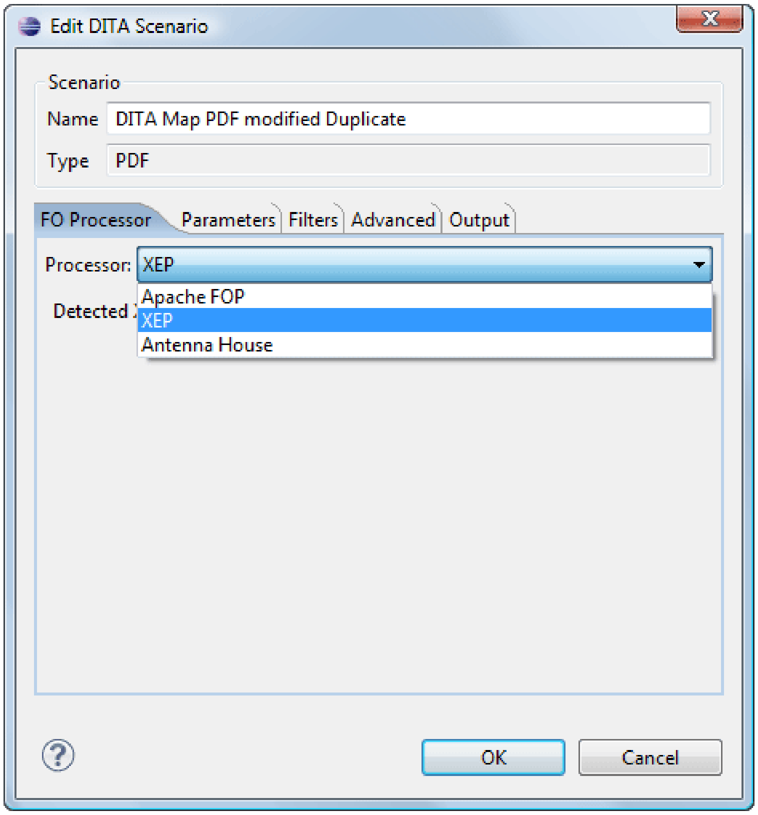
You have several parameters that you can specify here:

- **Base directory** - All the relative paths which appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located.
- **Temporary files directory** - This directory is used to store pre-processed temporary files until the final output is obtained.
- **Output folder** - The folder where the final output content is copied.
- **Output file options** - The transformation output can then be opened in a browser or even in the editor, if specified.

Note: If the DITA Map or topic is opened from a remote location or from a ZIP file, the scenario must specify absolute output, temporary and base file paths.

The *FO Processor* Tab

This tab appears only when selecting to generate PDF output using the IDIOM FO Plugin and allows you to choose the FO Processor.



Edit DITA Scenario

Scenario

Name: DITA Map PDF modified Duplicate

Type: PDF

Processor: XEP

Figure 152: FO Processor configuration tab

You can choose between three processors:

- **Apache FOP** - This processor comes bundled with Oxygen XML Editor plugin .
- **XEP** - The *RenderX* XEP processor.

  If you select **XEP** in the combo and XEP was already installed in Oxygen XML Editor plugin you can see the detected installation path appear under the combo.

  XEP is considered as installed if it was detected from one of the following sources:

  - XEP was configured as an external FO Processor in the ***FO Processors** option page*.
  - The system property *com.oxygenxml.xep.location* was set to point to the XEP executable file for the platform (eg: `xep.bat` on Windows).
  - XEP was installed in the `frameworks/dita/DITA-OT/demo/fo/lib` directory of the Oxygen XML Editor plugin installation directory.
- **Antenna House** - The *Antenna House* AH (v5) or XSL (v4) Formatter processor.

  If you select **Antenna House** in the combo and Antenna House was already installed in Oxygen XML Editor plugin you can see the detected installation path appear under the combo.

  Antenna House is considered as installed if it was detected from one of the following sources:

  - Environment variable set by Antenna House installation (the newest installation version will be used, v5 being preferred over v4).
  - Antenna House was added as an external FO Processor in the Oxygen XML Editor plugin preferences.

Tip: The DITA-OT contributors recommend the use of the IDIOM FO Plugin to transform DITA Maps to PDF as opposed to using the standard PDF target in the DITA-OT framework. As IDIOM is also bundled with Oxygen XML Editor plugin the *PDF2 - IDIOM FO Plugin* output format should be your first choice in transforming your map to PDF. If you do not have a commercial license for XEP or Antenna House you can transform using the Apache FO Processor.

Set a Font for PDF Output Generated with Apache FOP

When a DITA map is transformed to PDF using the Apache FOP processor and it contains some Unicode characters that cannot be rendered by the default PDF fonts then a font that is capable to render these characters must be configured and embedded in the PDF result.

The settings that must be modified for configuring a font for the Apache FOP processor are detailed in *this section*.

DITA-OT Customization

This section explains how to customize specific parameters of a DITA transformation scenario like setting a custom DITA Open Toolkit, a custom build file or a separate installation of the Ant tool.

Support for Transformation Customizations

You can change all DITA transformation parameters to customize your needs. In addition, you can specify a custom build file, parameters to the JVM and many more for the transformation.

Using Your Custom Build File

You can specify a custom build file to be used in DITA-OT transformations by editing the transformation scenario that you are using. In the ***Advanced*** tab you should change the **Custom build file** path to point to the custom build file.

Customizing the Oxygen XML Editor plugin Ant Tool

The Ant 1.7 tool which comes with Oxygen XML Editor plugin is located in the `[Oxygen-install-folder]/tools/ant` directory. Any additional libraries for Ant must be copied to the Oxygen XML Editor plugin Ant `lib` directory.

If you are using Java 1.6 to run Oxygen XML Editor plugin the Ant tool should need no additional libraries to process JavaScript in build files.

Upgrading to a New Version of DITA OT

The DITA OT framework bundled in Oxygen XML Editor plugin is located in the `[Oxygen-install-folder]/frameworks/dita/DITA-OT` folder.

Important: There are a couple of modifications made to the DITA OT framework which will be overwritten if you choose to copy the new DITA-OT version over the bundled one:

- The DTD's in the framework have been enriched with documentation for each element. If you overwrite you will lose the documentation which is usually shown when hovering an element or in the **Model** view.
- Several build files from the IDIOM plugin have been modified to allow transformation using the Oxygen XML Editor plugin Apache built-in FOP libraries and usage of the Oxygen XML Editor plugin classpath while transforming.
- Oxygen XML Editor plugin provides Java patches for some DITA OT problems. These patches are located in the `[Oxygen-install-folder]/frameworks/dita/DITA-OT/lib/dost-patches.jar` library. If the patches library conflicts with the new DITA OT libraries it either has to be removed from disk or removed from the libraries list available in the DITA Map transformation scenario.

Increasing the Memory for the Ant Process

For details about setting custom JVM arguments to the ANT process please see *this section*.

Resolving Topic References Through an XML Catalog

There are situations where you want to resolve URIs with an XML catalog:

- you customized your DITA map to refer topics using URI's instead of local paths

- you have URI content references in your DITA topic files and you want to map them to local files when the map is transformed

In such situations you have to *add the catalog to Oxygen XML Editor plugin* . The **DITA Maps Manager** view will solve the displayed topic refs through the added XML catalog and also the DITA map transformations (for PDF output, for XHTML output, etc) will solve the URI references through the added XML catalog.

DITA Specialization Support

This section explains how you can integrate and edit a DITA specialization in Oxygen XML Editor plugin .

Integration of a DITA Specialization

A DITA specialization includes:

- DTD definitions for new elements as extensions of existing DITA elements
- optionally specialized processing, that is new XSLT template rules that match the extension part of the `class` attribute values of the new elements and thus extend the default processing available in DITA Open Toolkit

A specialization can be integrated in Oxygen XML Editor plugin XML Author with minimum effort.

If the DTD's that define the extension elements are located in a folder outside the DITA Open Toolkit folder you should add new rules to the DITA OT catalog file for resolving the DTD references from the DITA files that use the specialized elements to that folder. This allows correct resolution of DTD references to your local DTD files and is needed for both validation and transformation of the DITA maps or topics. The DITA OT catalog file is called `catalog-dita.xml` and is located in the root folder of the DITA Open Toolkit.

If there is specialized processing provided by XSLT stylesheets that override the default stylesheets from DITA OT these new stylesheets must be called from the Ant build scripts of DITA OT.

Important: If you are using DITA specialization elements in your DITA files it is recommended that you activate the **Enable DTD processing in document type detection** checkbox in the ***Document Type Association*** *page*.

Editing DITA Map Specializations

In addition to recognizing the default DITA map formats: `map` and `bookmap` the **DITA Maps Manager** view can also be used to open and edit specializations of DITA Maps.

All advanced edit actions available for the map like insertion of topic refs, heads, properties editing, allow the user to specify the element in an editable combo box. Moreover the elements which appear initially in the combo are all the elements which are allowed to appear at the insert position for the given specialization.

The topic titles rendered in the **DITA Maps Manager** view are collected from the target files by matching the `class` attribute and not a specific element name.

When editing DITA specializations of maps in the main editor the insertions of topic reference, topic heading, topic group and conref actions should work without modification. For the table actions you have to modify each action by hand to insert the correct element name at caret position. You can go to the **DITA Map** document type from the ***Document Type Association*** *page* and edit the table actions to insert the element names as specified in your specialization. See *this section* for more details.

Editing DITA Topic Specializations

In addition to recognizing the default DITA topic formats: `topic`, `task`, `concept`, `reference` and `composite`, topic specializations can also be edited in the Author page.

The content completion should work without additional modifications and you can choose the tags which are allowed at the caret position.

The CSS styles in which the elements are rendered should also work on the specialized topics without additional modifications.

The toolbar/menu actions should be customized to insert the correct element names if this is the case. You can go to the `DITA` document type from the ***Document Type Association** page* and edit the actions to insert the element names as specified in your specialization. See *this section* for more details.

Use a New DITA Open Toolkit in Oxygen XML Editor plugin

Oxygen XML Editor plugin comes bundled with a DITA Open Toolkit, located in the `[Oxygen-install-folder]/frameworks/dita/DITA-OT` directory. To use a new DITA Open Toolkit, follow these steps:

1. Edit your transformation scenarios and in the **Parameters** tab change the value for the `dita.dir` directory to point to the new directory.
2. If you want to make changes in the libraries that come with the new DITA Open Toolkit and are used by the ANT process you can go to the **Advanced** tab, click the **Libraries** button, uncheck the checkbox **Allow Oxygen to add high priority libraries to classpath**.
3. If there are also changes in the DTD's and you want to use the new versions for content completion and validation, go to the Oxygen XML Editor plugin preferences in the **Document Type Association** page, edit the **DITA** and **DITA Map** document types and modify the catalog entry in the **Catalogs** tab to point to the custom catalog file `catalog-dita.xml`.

If the transformation fails after completing these steps please also take a look at this note about *upgrading the DITA OT which comes bundled with Oxygen XML Editor plugin* .

Reusing Content

The DITA framework allows reusing content from other DITA files with a content reference in the following ways:

- You can select content in a topic, create a reusable component from it and reference the component in other locations using the actions **Create Reusable Component** and **Insert Reusable Component**. A reusable component is a file, usually shorter than a topic. You also have the option of replacing the selection with the component that you are in the process of creating.
- You can add, edit, and remove a content reference (`conref`) attribute to/from an existing element. The actions **Add/Edit Content Reference** and **Remove Content Reference** are available on the contextual menu of the Author editor and on the DITA menu. When a content reference is added or an existing content reference is edited, you can select any topic ID or interval of topic IDs (set also the `conrefend` field in the dialog for adding/editing the content reference) from a target DITA topic file.
- You can insert an element with a content reference (`conref` or `conkeyref`) attribute using one of the actions **Insert Content Reference** and **Insert Content Key Reference** that are available on the DITA menu, the Author custom actions toolbar and the contextual menu of the Author editor.

DITA makes the distinction between local content, that is the text and graphics that are actually present in the element, and referenced content that is referred by the element but is located in a different file. You have the option of displaying referenced content by setting the option **Display referred content** that is available from menu **Options** > **Preferences** > **Editor** > **Edit modes** > **Author**.

Working with Content References

The DITA feature called `conref` (short for *content reference*) enables a piece of content to be included by reference in multiple contexts. When you need to update that content, you do it in only one place. Typical uses of content references are for product names, warnings, definitions or process steps.

You can use either or both of the following strategies for managing content references:

- *Reusable components* - With this strategy, you create a new file for each piece of content that you want to reuse.

- *Arbitrary content references* - You may prefer to keep many pieces of reusable content in one file. For example, you might want one file to consist of a list of product names, with each product name in a `phrase` (`<ph>` element) within the file. Then, wherever you need to display a product name, you can insert a content reference that points to the appropriate `<ph>` element in this file.

  This strategy requires more setup than reusable components, but makes easier centrally managing the reused content.

Oxygen XML Editor plugin creates a reference to the external content by adding a `conref` attribute to an element in the local document. The `conref` attribute defines a link to the referenced content, made up of a path to the file and the topic ID within the file. The path may also reference a specific element ID within the topic. Referenced content is not physically copied to the referencing file, but Oxygen XML Editor plugin displays it as if it is there in the referencing file. You can also choose to view local content instead of referenced content, to edit the attributes or contents of the referencing element.

How to Work with Reusable Components

When you need to reuse a part of a DITA topic in different places (in the same topic or in different topics) it is recommended to create a separate component and insert only a reference to the new component in all places. Below are the steps for extracting a reusable component, inserting a reference to the component and quickly editing the content inside the component.

1. Select with the mouse the content that you want to reuse in the DITA file opened in Author mode.
2. Start the action **Create Reusable Component** that is available on the DITA menu, the Author framework actions toolbar and the contextual menu of the Author editor.
3. In the combo box **Reuse Content** select the DITA element with the content that you want to extract in a separate component. The combo box contains the current DITA element where the cursor is located (for example a `p` element - a paragraph - or a `step` or a `taskbody` or a `conbody` etc.) and also all the ancestor elements of the current element.
4. In the **Description** area enter a textual description for quick identification by other users of the component.
5. If you want to replace the extracted content with a reference to the new component you should leave the checkbox **Replace selection with content reference** with the default value (selected).
6. Press the **Save** button which will open a file system dialog where you have to select the folder and enter the name of the file that will store the reusable component.
7. Press the **Save** button in the file system dialog to save the reusable component in a file. If the checkbox was selected in the **Create Reusable Component** dialog the `conref` attribute will be added to the element that was extracted as a separate component. In Author mode the content that is referenced by the `conref` attribute is displayed with grey background and is read-only because it is stored in other file.
8. Optionally, to insert a reference to the same component in other location just place the cursor at the insert location and run the action **Insert Reusable Component** that is available on the DITA menu, the Author framework actions toolbar and the contextual menu of the Author editor. Just select in the file system dialog the file that stores the component and press the **OK** button. The action will add a `conref` attribute to the DITA element at the insert location. The referenced content will be displayed in Author mode with grey background to indicate that it is not editable.
9. Optionally, to edit the content inside the component just click on the open icon at the start of the grey background area which will open the component in a separate editor.

Insert a Direct Content Reference

You should follow these steps for inserting an element with a content reference (`conref`) attribute:

1. Start one of the actions **Insert a DITA Content Reference** and **Insert a DITA Content Key Reference**.
2. In the dialog **Insert Content Reference** select the file with the referenced content in the **URL** field.
3. In the tree that presents the DITA elements of the specified file that have an `id` attribute you have to select the element or the interval of elements that you want to reference. The `conref` field will be filled automatically with

the `id` value of the selected element. If you select an interval of elements the `conrefend` field will be filled with the `id` value of the element that ends the selected interval.

4. Press the **OK** button to insert in the current DITA file an element with the same name and with the same `conref` attribute value (and optionally with the same `conrefend` attribute value) as the element(s) selected in the dialog.

DITA Profiling / Conditional Text

Conditional text is a way to mark blocks of text meant to appear in some renditions of the document, but not in others. It differs from one variant of the document to another, while unconditional text appear in all document versions.

For instance you can mark a section of a document to be included in the manual designated for the *expert* users, other for the *novice* users manual while unmarked sections are included in any rendition.

You can use conditional text when you develop documentation for:

- a series of similar products
- different releases of a product
- various audiences

The benefits of using conditional text include reduced effort for updating and translating your content and an easy way to customize the output for various audiences.

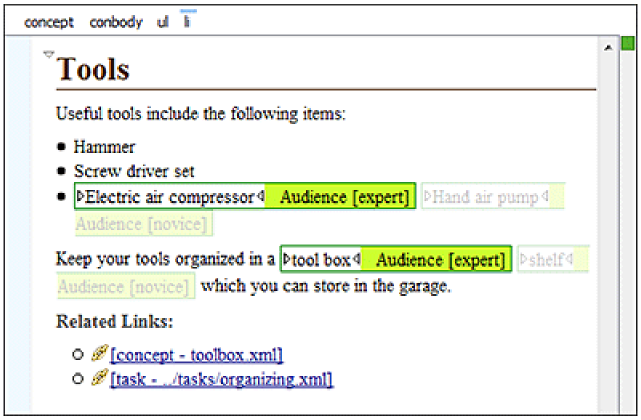
Oxygen XML Editor plugin offers full support for DITA conditional text processing: profiling attributes can be easily managed to filter content in the published output. You can toggle between different profile sets to see how the edited content looks like before publishing.

DITA offers support for profiling/conditional text by using profiling attributes. With Oxygen XML Editor plugin you can define values for the DITA profiling attributes. The profiling configuration can be shared between content authors through the project file. There is no need for coding or editing configuration files.

Several profiling attributes can be aggregated into a profiling condition set that allow you to apply more complex filters on the document content. A Profiling Condition Set is a very powerful and convenient tool used to preview the content that goes into the published output. For example, an installation manual available both in Windows and Linux variants can be profiled to highlight only the Linux procedures for more advanced users.

Profiling / Conditional Text Markers

If **Show Profiling Attributes** option (available in the ***Profiling / Conditional Text** toolbar menu*) is set all profiling attributes set on the current element are listed at the end of the highlighted block. Profiled text is marked in the **Author** mode with a light green border.



Tools

Useful tools include the following items:

- Hammer
- Screw driver set
- Electric air compressor Audience [expert] Hand air pump Audience [novice]

Keep your tools organized in a tool box Audience [expert] shelf Audience [novice] which you can store in the garage.

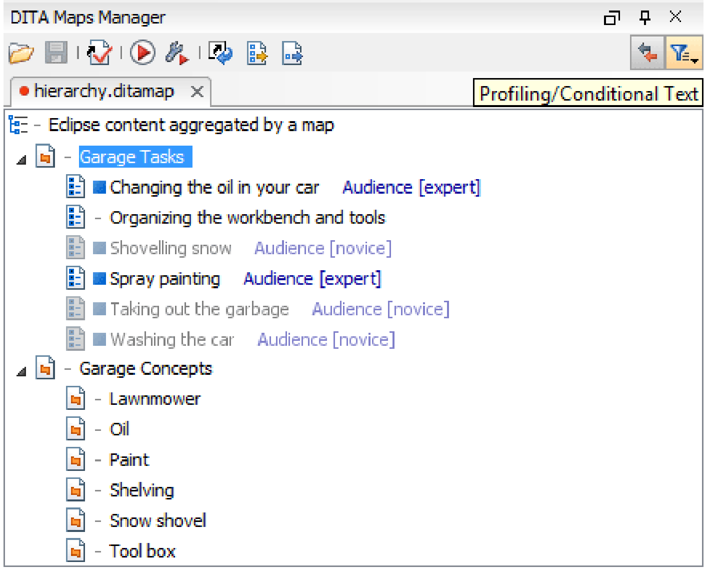
Related Links:

- [concept - toolbox.xml]
- [task - ../tasks/organizing.xml]

Figure 153: Profiling in Author

In the ***DITA Maps Manager View*** view different decorators are user to mark profiled and non-profiled topics:

- ■ - the topic contains profiling attributes;
- □ - the topic inherits profiling attribute from its ancestors;
- ◪ - the topic contains and inherits profiling attributes;
- - (dash) - the topic neither contains, nor inherits profiling attributes.



- Eclipse content aggregated by a map
  - Garage Tasks
    - ■ Changing the oil in your car Audience [expert]
    - - Organizing the workbench and tools
    - ■ Shovelling snow Audience [novice]
    - ■ Spray painting Audience [expert]
    - ■ Taking out the garbage Audience [novice]
    - ■ Washing the car Audience [novice]
  - Garage Concepts
    - - Lawnmower
    - - Oil
    - - Paint
    - - Shelving
    - - Snow shovel
    - - Tool box

Figure 154: Profiling in DITA Maps Manager

The profiled content that does not match the rules imposed by the current condition sets is grayed-out, meaning that it will not be included in the published output.

Profiling with a Subject Scheme Map

A subject scheme map enables DITA users to create custom profiling values and to manage the profiling attribute values used in the DITA topics without having to write a DITA specialization.

Subject scheme maps use key definitions to define a collection of profiling values instead of a collection of topics. The highest level of map that uses the set of profiling values must reference the subject scheme map in which the profiling values are defined, for example:

```
<topicref href="test.ditamap" format="ditamap" type="subjectScheme"/>
```

A profiled value is a short and readable keyword that identifies a metadata attribute. For example, the `audience` metadata attribute may take a value that identifies the user group associated with a particular content unit. Typical user values for a medical-equipment product line might include `therapist`, `oncologist`, `physicist`, `radiologist`, and so on. A subject scheme map can define a list of these audience values:

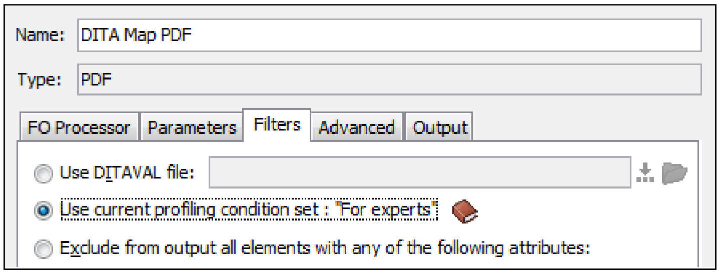
```
<subjectScheme>
    <!-- Pull in a scheme that defines audience user values -->
    <subjectdef keys="users">
        <subjectdef keys="therapist">
        <subjectdef keys="oncologist">
        <subjectdef keys="physicist">
        <subjectdef keys="radiologist">
    </subjectdef>

    <!-- Define an enumeration of the audience attribute, equal to
       each value in the users subject. This makes the following values
       valid for the audience attribute: therapist, oncologist, physicist,
radiologist -->
    <enumerationdef>
```

```
        <attributedef name="audience"/>
        <subjectdef keyref="users"/>
    </enumerationdef>
</subjectScheme>
```

When editing a DITA topic file in Text mode or in Author mode, the Oxygen application collects all the profiling values from the subject scheme map that is referenced in the current DITA map. In our example the values `therapist`, `oncologist`, `physicist`, `radiologist` and displays them in *the content completion window* as values for the `audience` attribute.

Publish Profiled Text

Oxygen XML Editor plugin comes with preconfigured transformation scenarios for DITA. All these scenarios take into account the current profiling condition set.



Name: DITA Map PDF

Type: PDF

○ Use DITAVAL file:

● Use current profiling condition set : "For experts"

○ Exclude from output all elements with any of the following attributes:

How to Profile DITA Content

1. Go to **Preferences** > **Editor** > **Edit modes** > **Author** > **Profiling / Conditional Text** page and edit the **Profiling Attributes** table.
2. For DITA there are already default entries for `audience`, `platform`, `product` and `otherprops`. You can customize the attributes and their values.

   This is a one-time operation. Once you save the customized attributes and values, you can use them to profile any DITA project.
3. To use the profiling attributes set in the previous step, do one of the following:
   a) Right-click (Ctrl + click on MacOS) a topic reference in **DITA Maps Manager** and choose **Edit Profiling Attributes** from the contextual menu.
   b) In the **Author** editing mode, right-click (Ctrl + click on MacOS) an XML element and choose **Edit Profiling Attributes** from the contextual menu.
   c) Use the **Attributes** view to set profiling attributes.

   Turn on the **Show Profiling Attributes** option to display the profiling markup in the **Author** editing mode.

Working with MathML

You can add MathML equations in a DITA document using one of the following methods:

- embed MathML directly into a DITA topic. You can start with the **Framework templates / DITA / topic / Composite with MathML** document template, available in the **New** file action wizard.
- reference an external MathML file as an image, using the **Insert Image Reference** toolbar action.

Note that MathML equations contained in DITA topics can only be published out-of-the-box in PDF using the **DITA PDF** transformation scenario. For other publishing formats users must employ additional customizations for handling MathML content.

Chapter 7

Predefined Document Types

Topics:

The following are the short presentations of some document types that come bundled with Oxygen XML Editor plugin . For each document type there are presented built-in transformation scenarios, document templates and Author extension actions.

Document Type

A *document type* or *framework* is associated to an XML file according to a set of rules. It includes also many settings that improve editing in the Tagless editor for the category of XML files it applies for. These settings include:

- a default grammar used for validation and content completion in both Author mode and Text mode
- CSS stylesheet(s) for rendering XML documents in Author mode
- user actions invoked from toolbar or menu in Author mode
- predefined scenarios used for transformation of the class of XML documents defined by the document type
- XML catalogs
- directories with file templates
- user defined extensions for customizing the interaction with the content author in Author mode

The tagless editor comes with some predefined document types already configured when the application is installed on the computer. These document types describe well-known XML frameworks largely used today for authoring XML documents. Editing a document which conforms to one of these types is as easy as opening it or creating it from one of the predefined document templates which also come with the application.

The DocBook 4 Document Type

DocBook is a very popular set of tags for describing books, articles, and other prose documents, particularly technical documentation.

A file is considered to be a *DocBook 4* document when one of the following conditions are true:

- root element name is `book` or `article`
- the PUBLIC ID of the document contains the string `-//OASIS//DTD DocBook XML`

The schema of *DocBook 4* documents is `${frameworks}/docbook/dtd/docbookx.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering DocBook content is located in `${frameworks}/docbook/css/docbook.css`.

The XML catalog is stored in `${frameworks}/docbook/catalog.xml`.

Author Extensions

Specific actions for DocBook documents are:

- **Bold emphasized text** - Emphasizes the selected text by surrounding it with `<emphasis role="bold"/>` tag.
- **Italic emphasized text** - Emphasizes the selected text by surrounding it with `<emphasis role="italic"/>` tag.
- **Underline emphasized text** - Emphasizes the selected text by surrounding it with `<emphasis role="italic"/>` tag.

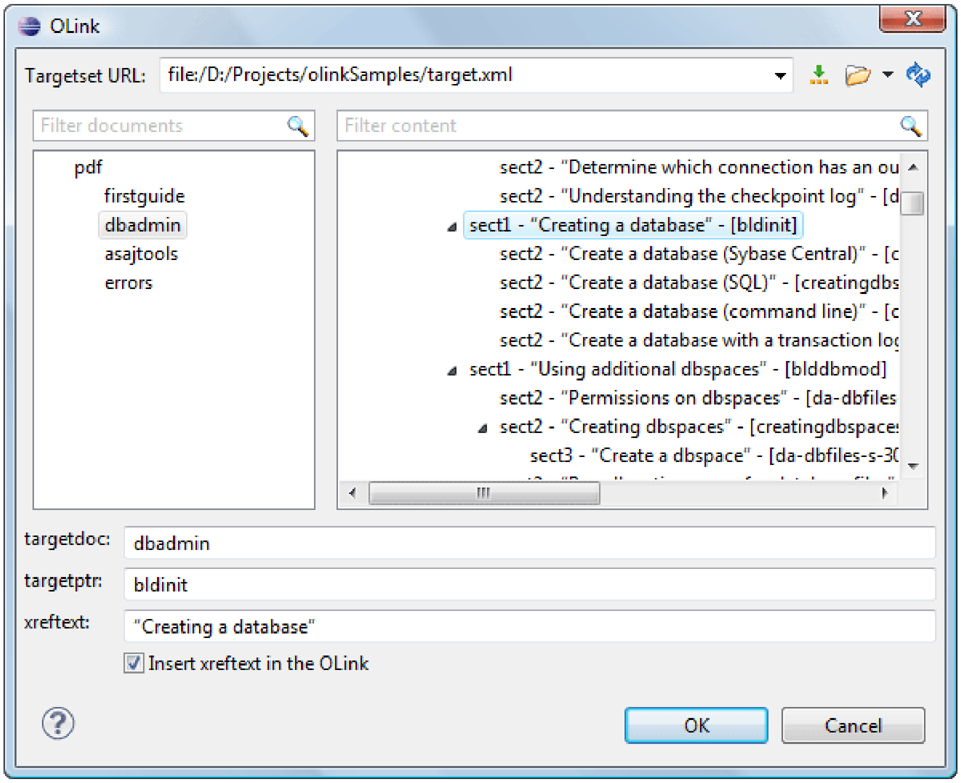
  Note:

  Bold, **Italic**, and **Underline** are toggle actions.

  For all of the above actions, if there is no selection, then a new `emphasis` tag with specific role is inserted. These actions are available in any document context and are grouped under the **Emphasize** toolbar actions group.

- **Browse reference manual** - Opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position.
- **link** - Inserts a hypertext link.

- **ulink** - Inserts a link that addresses its target with an URL (Universal Resource Locator).
- **olink** - Inserts a link that addresses its target indirectly, using the `targetdoc` and `targetptr` values which are present in a ***Targetset*** file.



Targetset URL: file:/D:/Projects/olinkSamples/target.xml

pdf
firstguide
dbadmin
asajtools
errors

sect2 - "Determine which connection has an ou
sect2 - "Understanding the checkpoint log" - [d
sect1 - "Creating a database" - [bldinit]
sect2 - "Create a database (Sybase Central)" - [c
sect2 - "Create a database (SQL)" - [creatingdbs
sect2 - "Create a database (command line)" - [c
sect2 - "Create a database with a transaction lo
sect1 - "Using additional dbspaces" - [blddbmod]
sect2 - "Permissions on dbspaces" - [da-dbfiles
sect2 - "Creating dbspaces" - [creatingdbspace
sect3 - "Create a dbspace" - [da-dbfiles-s-30

targetdoc: dbadmin

targetptr: bldinit

xreftext: "Creating a database"

☑ Insert xreftext in the OLink

Figure 155: Insert OLink Dialog

After you choose the **Targetset** URL, the structure of the target documents is presented. For each target document (`targetdoc`), the content is displayed allowing for easy identification of the `targetptr` for the `olink` element which will be inserted. You can use the search fields to quickly identify a target. If you already know the values for the `targetdoc` and `targetptr`, you can insert them directly in the corresponding fields. You also have the possibility to edit an `olink` using the action **Edit OLink** available on the contextual menu. The last used **Targetset** URL will be used to identify the edited target.

- **URI** - Inserts an URI element. The URI identifies a Uniform Resource Identifier (URI) in content.
- **xref** - Inserts a cross reference to another part of the document.

Note: These actions are grouped under the **Link** toolbar actions group.

- § **Insert Section** - Inserts a new section / subsection in the document, depending on the current context. For example if the current context is `sect1` then a `sect2` is inserted, and so on.
- ¶ **Insert Paragraph** - Inserts a new paragraph depending on the current context. If current context is a paragraph context (one of the ancestors of the element at caret position is `para`) then a new paragraph will be inserted after the paragraph at caret. Otherwise a new paragraph is inserted at caret position.
- **Insert Graphic** - Inserts a graphic object at the caret position. This is done by inserting either `<figure>` or `<inlinegraphic>` element depending on the current context. The following graphical formats are supported: GIF, JPG, JPEG, BMP, PNG, SVG.
- **Insert List Item** - Inserts a new list item in any of the above three list types.
- **Insert Ordered List** - Inserts an ordered list. A child list item is also inserted automatically by default.
- **Insert Itemized List** - Inserts an itemized list. A child list item is also inserted automatically by default.
- **Insert Variable List** - Inserts a DocBook variable list. A child list item is also inserted automatically by default.

- **Insert Procedure** - Inserts a DocBook `procedure` element. A step `child` item is also inserted automatically.
- **Insert Table** - Opens a dialog that allows you to configure the table to be inserted. The dialog allows the user to configure the number of rows and columns of the table, if the header and footer will be generated and how the table will be framed. **CALS** or **HTML** table model can be selected.

  Note: If the **Title** checkbox is unchecked, an `informaltable` element is inserted.

- **Insert Row** - Inserts a new table row with empty cells. The action is available when the caret position is inside a table.
- **Insert Column** - Inserts a new table column with empty cells after the current column. The action is available when the caret position is inside a table.
- **Insert Cell** - Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, a new one is inserted at caret position. If the caret is inside a cell, then the new one will be created after the current cell.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.
- **Join Row Cells** - Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.
- **Join Cell Above** - Joins the content of cell from current caret position with that of the cell above it. This action works only if both cells have the same column span.
- **Join Cell Below** - Joins the content of cell from current caret position with that of the cell below it. This action works only if both cells have the same column span.
- **Split Cell To The Left** - Splits the cell from current caret position in two, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column. The column span of the source cell is decreased with one.
- **Split Cell To The Right** - Splits the cell from current caret position in two, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column. The column span of the source cell is decreased with one.
- **Split Cell Above** - Splits the cell from current caret position in two, inserting a new empty table cell above it. This action works only if the current cell spans over more than one row. The row span of the source cell is decreased with one.
- **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below it. This action works only if the current cell spans over more than one row. The row span of the source cell is decreased with one.

Note: DocBook 4 supports only CALS table model. HTML table model is supported only in DocBook 5.

Caution: Column specifications are required for table actions to work properly.

- **Generate IDs** - Available in the contextual menu only, this action allows you to generate an unique ID for the element at caret position. If the element already has an id set, it is preserved. Further options are offered in the **ID Options** dialog, available in the **DocBook4** main menu. Here you can specify the elements for which Oxygen XML Editor plugin generates an ID if the **Auto generate ID's for elements** is enabled. If you want to keep already set element id's when copying content in the same document, make sure the **Remove ID's when copying content in the same document** option is not checked.

All actions described above are available in the contextual menu, the **DocBook4** submenu of the main menu or in the **Author custom actions** toolbar.

Dragging a file from *the **Project** view* or from *the **DITA Maps Manager** view* and dropping it into a DocBook 4 document that is edited in Author mode creates a link to the dragged file (the `ulink` DocBook element) at the drop location. Dragging an image file from the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) and dropping it into a DocBook 4 document inserts an image element (the `inlinegraphic` DocBook element with the `fileref` attribute) with the location of the dragged file at the drop location (similar with the **Insert Graphic** toolbar action).

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 agtargets SYSTEM "file:///doc/adminguide/target.db">
<!ENTITY reftargets SYSTEM "file:///doc/man/target.db">
]>
<targetset>
  <targetsetinfo>
    Description of this target database document,
    which is for the examples in olink doc.
  </targetsetinfo>

  <!-- Site map for generating relative paths between documents -->
  <sitemap>
    <dir name="documentation">
      <dir name="guides">
        <dir name="mailuser">
          <document targetdoc="MailUserGuide"
                    baseuri="userguide.html">
            &ugtargets;
          </document>
        </dir>
        <dir name="mailadmin">
          <document targetdoc="MailAdminGuide">
            &agtargets;
          </document>
        </dir>
      </dir>
      <dir name="reference">
        <dir name="mailref">
          <document targetdoc="MailReference">
            &reftargets;
          </document>
        </dir>
      </dir>
    </dir>
  </sitemap>
</targetset>
```

   An example of a `target.db` file:

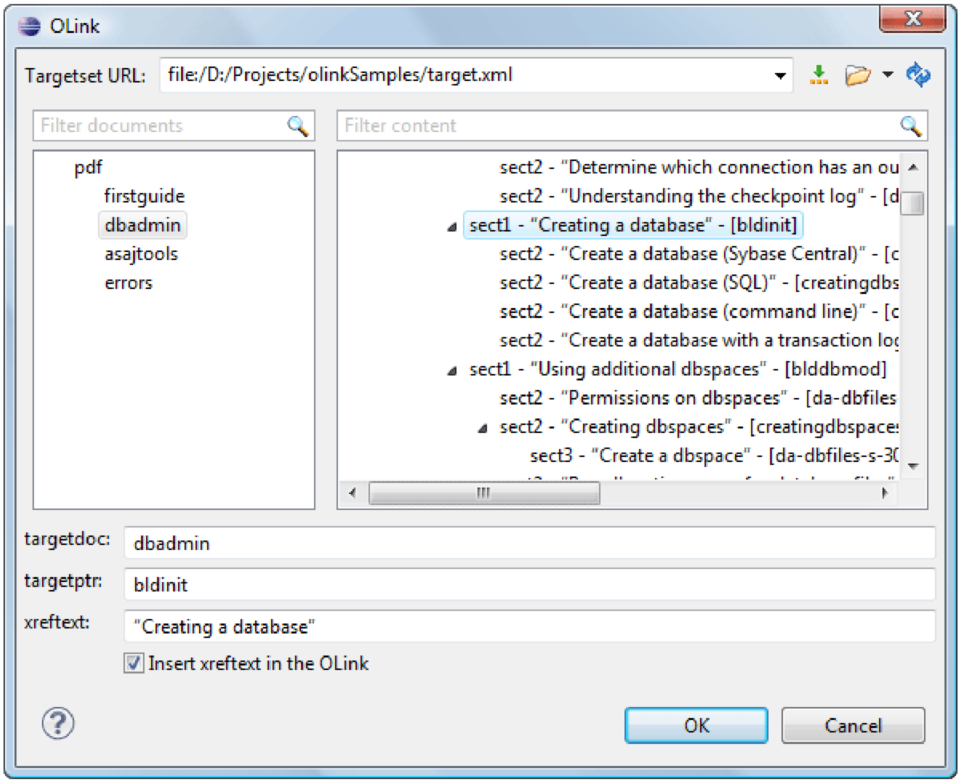
```
<!DOCTYPE div
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">

  <ttl>Administering User Accounts</ttl>
  <xreftext>How to administer user accounts</xreftext>
  <div element="part" href="#d5e4" number="I">
    <ttl>First Part</ttl>
    <xreftext>Part I, “First Part”</xreftext>
    <div element="chapter" href="#d5e6" number="1">
      <ttl>Chapter Title</ttl>
      <xreftext>Chapter 1, Chapter Title</xreftext>
      <div element="sect1" href="#src_chapter" number="1"
targetptr="src_chapter">
        <ttl>Section1 Title</ttl>
        <xreftext>xreflabel_here</xreftext>
      </div>
    </div>
  </div>
</div>
```

4. Generate the target data files.

These files are the `target.db` files from the above example of target database document. They are created with the same DocBook transformation scenario as the HTML or XHTML output. The XSLT parameter called `collect.xref.targets` must be set to the value `yes`. The default name of a target data file is `target.db` but it can be changed by setting an absolute file path in the XSLT parameter `targets.filename`.

5. Insert `olink` elements in the DocBook XML documents.

When a DocBook XML document is edited in Author mode 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`.



Targetset URL: file:/D:/Projects/olinkSamples/target.xml

pdf
firstguide
dbadmin
asajtools
errors

sect2 - "Determine which connection has an ou
sect2 - "Understanding the checkpoint log" - [d
sect1 - "Creating a database" - [bldinit]
sect2 - "Create a database (Sybase Central)" - [c
sect2 - "Create a database (SQL)" - [creatingdbs
sect2 - "Create a database (command line)" - [c
sect2 - "Create a database with a transaction lo
sect1 - "Using additional dbspaces" - [blddbmod]
sect2 - "Permissions on dbspaces" - [da-dbfiles
sect2 - "Creating dbspaces" - [creatingdbspace
sect3 - "Create a dbspace" - [da-dbfiles-s-30

targetdoc: dbadmin

targetptr: bldinit

xreftext: "Creating a database"

☑ Insert xreftext in the OLink

Figure 156: Insert OLink Dialog

6. Process each document for output.

That is done using a DocBook transformation scenario in which the URL of the target database document is set in the `target.database.document` parameter. The DocBook XSL stylesheets know how to resolve `olinks` in the output files using the value of this parameter.

The DocBook 5 Document Type

A file is considered to be a DocBook 5 document when the namespace is *http://docbook.org/ns/docbook*.

DocBook 5 documents use a Relax NG and Schematron schema located in `${frameworks}/docbook/5.0/rng/docbookxi.rng`, where *${frameworks}* is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering DocBook content is located in `${frameworks}/docbook/css/docbook.css`.

The XML catalog is stored in `${frameworks}/docbook/5.0/catalog.xml`.

Author Extensions

The DocBook 5 extensions are the same as the *DocBook 4 extensions*. In addition, the table actions work also for HTML tables.

A drag and drop with a file from *the **Project** view* or from *the **DITA Maps Manager** view* to a DocBook 5 document that is edited in Author mode will create a link to the dragged file (the `link` DocBook element) at the drop location. A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `inlinemediaobject` DocBook element with an `imagedata` child element) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

Transformation Scenarios

Default transformation scenarios allow you to transform DocBook 5 documents to HTML, HTML Chunk, PDF, XHTML, XHTML Chunk, WebHelp (experimental) and EPUB.

DocBook to EPUB Transformation

The EPUB specification recommends the use of *OpenType* fonts (recognized by their `.otf` file extension) when possible. To use a specific font:

- first you need to declare it in your CSS file, like:

```
@font-face {
font-family: "MyFont";
font-weight: bold;
font-style: normal;
src: url(fonts/MyFont.otf);
}
```

- tell the CSS where this font is used. To set it as default for `h1` elements, use the `font-family` rule as in the following example:

```
h1 {
font-size:20pt;
margin-bottom:20px;
font-weight: bold;
font-family: "MyFont";
text-align: center;
}
```

- in your DocBook to EPUB transformation, set the `epub.embedded.fonts` parameter to `fonts/MyFont.otf`. If you need to provide more files, use comma to separate their file paths.

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 agtargets SYSTEM "file:///doc/adminguide/target.db">
<!ENTITY reftargets SYSTEM "file:///doc/man/target.db">
]>
<targetset>
  <targetsetinfo>
    Description of this target database document,
    which is for the examples in olink doc.
  </targetsetinfo>

  <!-- Site map for generating relative paths between documents -->
  <sitemap>
    <dir name="documentation">
      <dir name="guides">
        <dir name="mailuser">
          <document targetdoc="MailUserGuide"
                    baseuri="userguide.html">
            &ugtargets;
          </document>
        </dir>
        <dir name="mailadmin">
          <document targetdoc="MailAdminGuide">
            &agtargets;
          </document>
        </dir>
      </dir>
      <dir name="reference">
        <dir name="mailref">
          <document targetdoc="MailReference">
            &reftargets;
          </document>
        </dir>
      </dir>
    </dir>
  </sitemap>
</targetset>
```

An example of a `target.db` file:

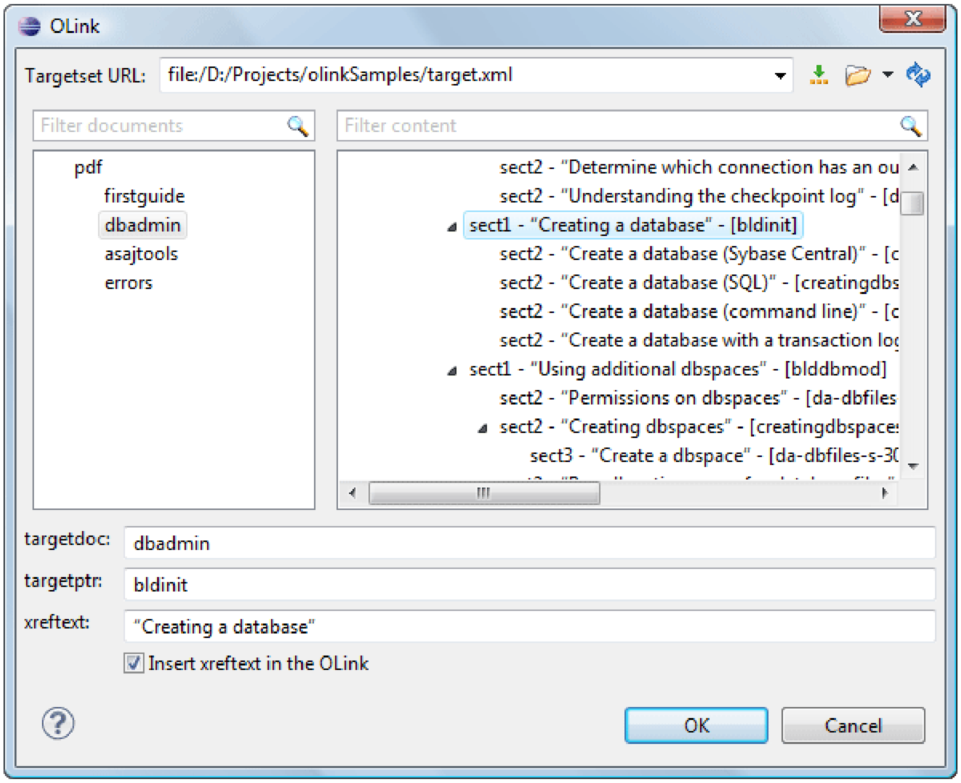
```
<!DOCTYPE div
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">

  <ttl>Administering User Accounts</ttl>
  <xreftext>How to administer user accounts</xreftext>
  <div element="part" href="#d5e4" number="I">
    <ttl>First Part</ttl>
    <xreftext>Part I, “First Part”</xreftext>
    <div element="chapter" href="#d5e6" number="1">
      <ttl>Chapter Title</ttl>
      <xreftext>Chapter 1, Chapter Title</xreftext>
      <div element="sect1" href="#src_chapter" number="1"
targetptr="src_chapter">
        <ttl>Section1 Title</ttl>
        <xreftext>xreflabel_here</xreftext>
      </div>
    </div>
  </div>
</div>
```

4. Generate the target data files.

   These files are the `target.db` files from the above example of target database document. They are created with the same DocBook transformation scenario as the HTML or XHTML output. The XSLT parameter called `collect.xref.targets` must be set to the value `yes`. The default name of a target data file is `target.db` but it can be changed by setting an absolute file path in the XSLT parameter `targets.filename`.

5. Insert `olink` elements in the DocBook XML documents.

   When a DocBook XML document is edited in Author mode 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`.



Targetset URL: file:/D:/Projects/olinkSamples/target.xml

pdf
firstguide
dbadmin
asajtools
errors

sect2 - "Determine which connection has an ou
sect2 - "Understanding the checkpoint log" - [d
sect1 - "Creating a database" - [bldinit]
sect2 - "Create a database (Sybase Central)" - [c
sect2 - "Create a database (SQL)" - [creatingdbs
sect2 - "Create a database (command line)" - [c
sect2 - "Create a database with a transaction lo
sect1 - "Using additional dbspaces" - [blddbmod]
sect2 - "Permissions on dbspaces" - [da-dbfiles
sect2 - "Creating dbspaces" - [creatingdbspace
sect3 - "Create a dbspace" - [da-dbfiles-s-30

targetdoc: dbadmin

targetptr: bldinit

xreftext: "Creating a database"

☑ Insert xreftext in the OLink

Figure 157: Insert OLink Dialog

6. Process each document for output.

   That is done using a DocBook transformation scenario in which the URL of the target database document is set in the `target.database.document` parameter. The DocBook XSL stylesheets know how to resolve `olinks` in the output files using the value of this parameter.

The DocBook Targetset Document Type

DocBook *Targetset* documents are used to resolve cross references with DocBook `olink`'s.

A file is considered to be a *Targetset* when the root name is `targetset`.

This type of documents use a DTD and schema located in `${frameworks}/docbook/xsl/common/targetdatabase.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

Templates

There is a default template for *Targetset* documents in the `${frameworks}/docbook/templates/Targetset` folder. It is available when creating *new documents from templates*.

- **Docbook Targetset - Map** - New Targetset Map.

The DITA Topics Document Type

The Darwin Information Typing Architecture (DITA) is an XML-based architecture for authoring, producing, and delivering technical information. It divides content into small, self-contained topics that can be reused in different deliverables. The extensibility of DITA permits organizations to define specific information structures and still use standard tools to work with them.

A file is considered to be a DITA topic document when either of the following occurs:

- the root element name is one of the following: `concept`, `task`, `reference`, `dita`, `topic`
- PUBLIC ID of the document is one of the PUBLIC ID's for the elements above
- the root element of the file has an attribute named `DITAArchVersion` attribute from the "http://dita.oasis-open.org/architecture/2005/" namespace. This enhanced case of matching is only applied when the **Enable DTD processing** option from the ***Document Type Detection*** *option page* is enabled.

The default schema used for DITA topic documents is located in `${frameworks}/dita/dtd/ditabase.dtd`, where *${frameworks}* is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering DITA content in Author mode is `${frameworks}/dita/css/dita.css`.

The default XML catalog is `${frameworks}/dita/catalog.xml`.

Author Extensions

The specific actions for DITA topic documents are:

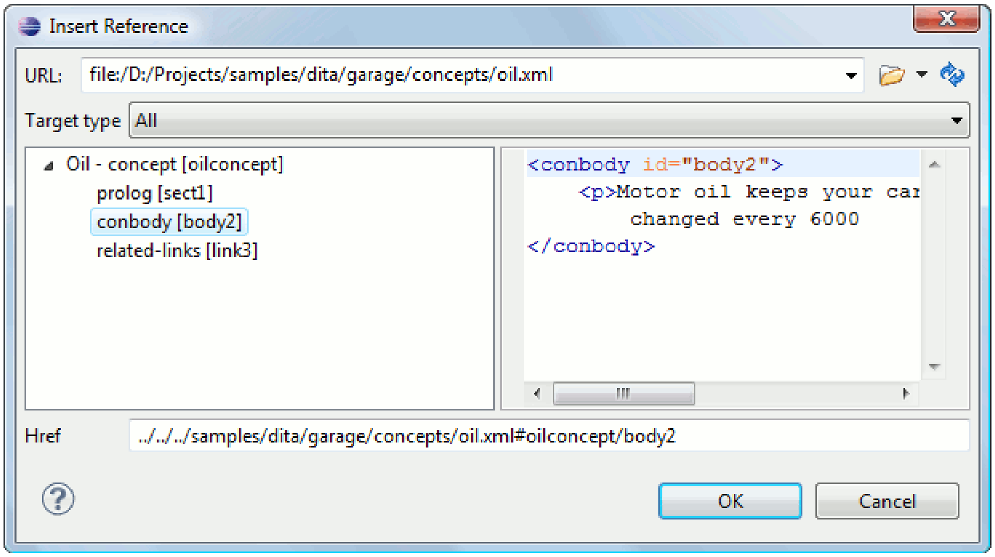
- **B** **Bold** - Surrounds the selected text with a `b` tag.
- *I* **Italic** - Surrounds the selected text with an `i` tag.
- U **Underline** - Surrounds the selected text with a `u` tag.

  Note:

  Bold, **Italic**, and **Underline** are toggle actions.

  For all of the above actions, if there is no selection in the document, then a new specific tag is inserted. These actions are available in any document context.

- **Browse reference manual** - Opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position.
- **Cross Reference** - Inserts an `xref` element with the value of attribute `format` set to `dita`. The target of the `xref` is selected in a dialog which lists all the IDs available in a file selected by the user.



URL: file:/D:/Projects/samples/dita/garage/concepts/oil.xml

Target type: All

- Oil - concept [oilconcept]
  - prolog [sect1]
  - conbody [body2]
  - related-links [link3]

```
<conbody id="body2">
    <p>Motor oil keeps your car
        changed every 6000
</conbody>
```

Href: ../../../samples/dita/garage/concepts/oil.xml#oilconcept/body2

Figure 158: Insert a cross reference in a DITA document

- **Key Reference** - Inserts a user specified element with the value of attribute `keyref` attribute set to a specific key name. As stated in the DITA 1.2 specification keys can be defined at map level which can be then referenced. The target of the `keyref` is selected in a dialog which lists all the keys available in the current opened map from the DITA Maps Manager.

  You can also reference elements at sub-topic level by pressing the **Sub-topic** button and choosing the target.

  Important: All keys which are presented in the dialog are gathered from the current opened DITA map. Elements which have the `keyref` attribute set are displayed as links. The current opened DITA map is also used to resolve references when navigating `keyref` links in the Author mode. Image elements which use key references are rendered as images.

- **File Reference** - Inserts an `xref` element with the value of attribute `format` set to `xml`.
- **Web Link** - Inserts an `xref` element with the value of attribute `format` set to `html`, and `scope` set to `external`.
- **Related Link to Topic** - Inserts a `link` element inside a `related-links` parent.
- **Related Link to File** - Inserts a `link` element with the `format` attribute set to `xml` inside a `related-links` parent.
- **Related Link to Web Page** - Inserts a `link` element with the attribute `format` set to `html` and `scope` set to `external` inside a `related-links` parent.

  Note: The actions for inserting references described above are grouped inside **link** toolbar actions group.

- **Paste as Link** (available on the contextual menu of Author editor for any topic file) - Inserts a `link` element or an `xref` one (depending on the location of the paste) that points to the DITA XML element from the clipboard. An entire DITA XML element with an ID attribute must be present in the clipboard when the action is invoked. The `href` attribute of `link/href` will point to this ID value.
- **Paste as Content Reference** (available on the contextual menu of Author editor for any topic file) - Inserts a content reference (a DITA element with a `conref` attribute) to the DITA XML element from the clipboard. An entire DITA XML element with an ID attribute must be present in the clipboard when the action is invoked. The `conref` attribute will point to this ID value.
- § **Insert Section / Step** - Inserts a new section / step in the document, depending on the current context. A new section will be inserted in either one of the following contexts:
  - section context, when the value of `class` attribute of the current element or one of its ancestors contains `topic` or `section`.
  - topic's body context, when the value of `class` attribute of the current element contains `topic/body`.

A new step will be inserted in either one of the following contexts:

- task step context, when the value of `class` attribute of the current element or one of its ancestors contains `task/step`.
- task steps context, when the value of `class` attribute of the current element contains `task/steps`.

- ¶ **Insert Paragraph** - Inserts a new paragraph depending on the current context. If current context is a paragraph context (the value of `class` attribute of the current element or one of its ancestors contains `topic/p`) then a new paragraph will be inserted after this paragraph. Otherwise a new paragraph is inserted at caret position.
- **Insert Concept** - Inserts a new concept. Concepts provide background information that users must know before they can successfully work with a product or interface. This action is available in one of the following contexts:

  - concept context, one of the current element ancestors is a `concept`. In this case an empty `concept` will be inserted after the current `concept`.
  - concept or DITA context, current element is a `concept` or `dita`. In this case an empty `concept` will be inserted at current caret position.
  - DITA topic context, current element is a `topic` child of a `dita` element. In this case an empty `concept` will be inserted at current caret position.
  - DITA topic context, one of the current element ancestors is a DITA `topic`. In this case an empty `concept` will be inserted after the first `topic` ancestor.

- **Insert Task** - Inserts a new task. Tasks are the main building blocks for task-oriented user assistance. They generally provide step-by-step instructions that will enable a user to perform a task. This action is available in one of the following contexts:

  - task context, one of the current element ancestors is a `task`. In this case an empty `task` will be inserted after the last child of the first `concept`'s ancestor.
  - task context, the current element is a `task`. In this case an empty `task` will be inserted at current caret position.
  - topic context, the current element is a `dita topic`. An empty `task` will be inserted at current caret position.
  - topic context, one of the current element ancestors is a `dita topic`. An empty `task` will be inserted after the last child of the first ancestor that is a `topic`.

- **Insert Reference** - Inserts a new reference in the document. A reference is a top-level container for a reference topic. This action is available in one of the following contexts:

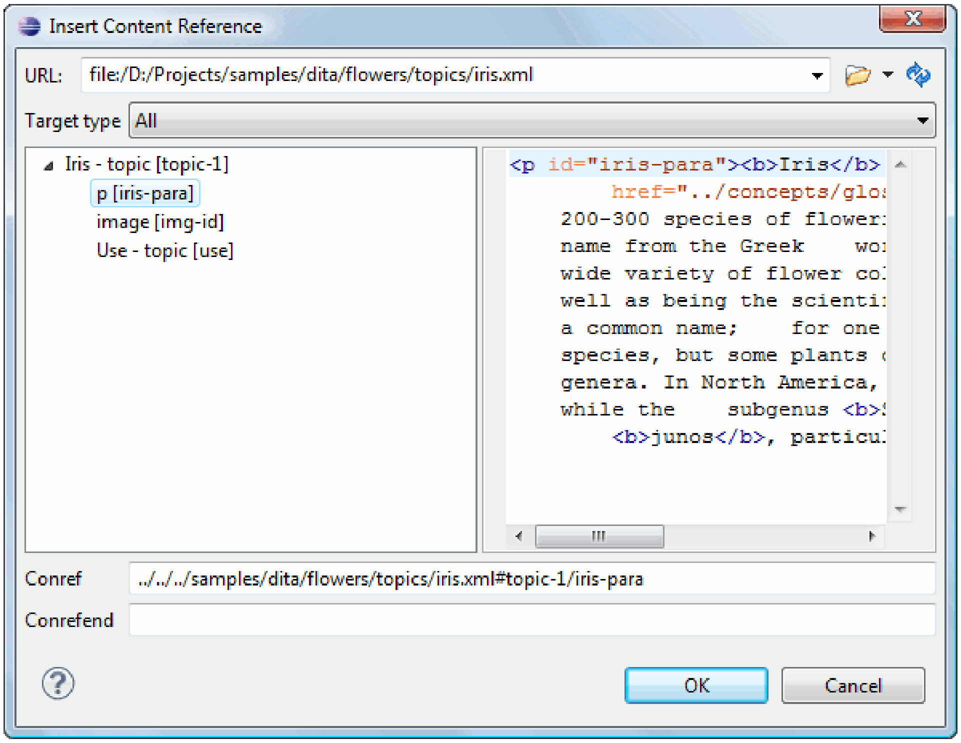
  - reference context - one of the current element ancestors is a `reference`. In this case an empty `reference` will be inserted after the last child of the first ancestor that is a `reference`.
  - `reference` or `dita` context - the current element is either a `dita` or a `reference`. An empty `reference` will be inserted at caret position.
  - topic context - the current element is `topic` descendant of `dita` element. An empty `reference` will be inserted at caret position.
  - topic context - the current element is descendant of `dita` element and descendant of `topic` element. An empty `reference` will be inserted after the last child of the first ancestor that is a `topic`.

- **Insert Image Reference** - Inserts a graphic object at the caret position. Depending on the current context, an image-type DITA element is inserted. The following graphical formats are supported: GIF, JPG, JPEG, BMP, PNG, SVG. Also you can use this action to *refer MathML files*.
- **Insert Content Reference** - Inserts a content reference at the caret position.

  The DITA `conref` attribute provides a mechanism for reuse of content fragments. The conref attribute stores a reference to another element and is processed to replace the referencing element with the referenced element. The element containing the content reference acts as a placeholder for the referenced element. The identifier for the referenced element must be either absolute or resolvable in the context of the referencing element. See *here* for more details.

  Oxygen XML Editor plugin will *display the referred content* of a DITA `conref` if it can resolve it to a valid resource. If you have URI's instead of local paths in the XML documents and your DITA OT transformation needs an XML catalog to map the URI's to local paths you have *add the catalog to Oxygen XML Editor plugin* . If the URI's can be resolved the referred content will be displayed in Author mode and in the transformation output.

A content reference is inserted with the action **Insert a DITA Content Reference** available on the toolbar **Author custom actions** and on the menu **DITA** > **Insert**.



URL: file:/D:/Projects/samples/dita/flowers/topics/iris.xml

Target type All

- Iris - topic [topic-1]
  - p [iris-para]
  - image [img-id]
  - Use - topic [use]

```
<p id="iris-para"><b>Iris</b>
      href="../concepts/glo:
  200-300 species of flower:
  name from the Greek    wo:
  wide variety of flower co:
  well as being the scienti:
  a common name;    for one
  species, but some plants :
  genera. In North America,
  while the    subgenus <b>:
      <b>junos</b>, particu:
```

Conref ../../../samples/dita/flowers/topics/iris.xml#topic-1/iris-para

Conrefend

Figure 159: Insert Content Reference Dialog

In the URL chooser you set the URL of the file from which you want to reuse content. Depending on the **Target type** filter you will see a tree of elements which can be referred (which have ID's). For each element the XML content is shown in the preview area. The **Conref** value is computed automatically for the selected tree element. After pressing **OK** an element with the same name as the target element and having the attribute `conref` with the value specified in the **Conref** value field will be inserted at caret position.

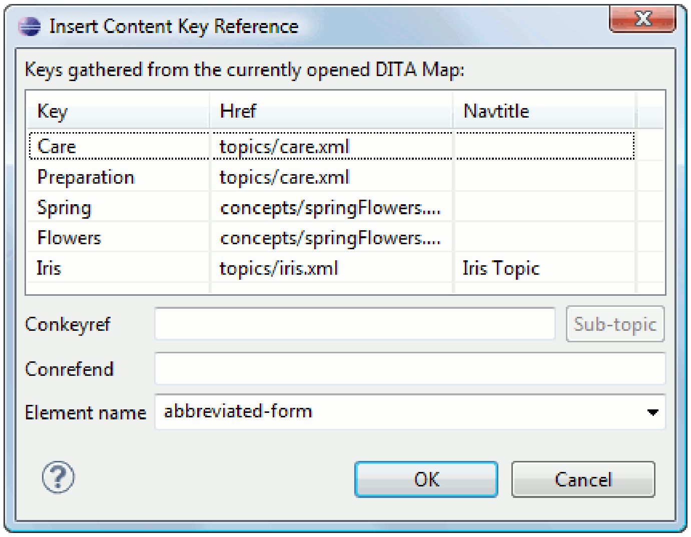
According to the DITA 1.2 specification the `conrefend` attribute can be used to specify content reference ranges. This is a very useful feature when referencing multiple consecutive steps or list items. If you use multiple contiguous sibling selection the `conrefend` value will also be set to the value of the last selected ID path. Oxygen XML Editor plugin will present the entire referenced range as read-only content.

- **Insert Content Key Reference** - Inserts a content key reference at the caret position.

  As stated in the DITA 1.2 specification the `conkeyref` attribute provides a mechanism for reuse of content fragments similar with the `conref` mechanism. Keys are defined at map level which can be referenced using `conkeyref`. The `conkeyref` attribute contains a key reference to another element and is processed to replace the referencing element with the referenced element. The element containing the content key reference acts as a placeholder for the referenced element. The identifier for the referenced element must be either absolute or resolvable in the context of the referencing element.

  Oxygen XML Editor plugin will *display the key referred content* of a DITA `conkeyref` if it can resolve it to a valid resource in the context of the current opened DITA map.

  A content key reference is inserted with the action **Insert a DITA Content Key Reference** available on the toolbar **Author custom actions** and on the menu **DITA** > **Insert**.



Insert Content Key Reference

Keys gathered from the currently opened DITA Map:

Key	Href	Navtitle
Care	topics/care.xml	
Preparation	topics/care.xml	
Spring	concepts/springFlowers....	
Flowers	concepts/springFlowers....	
Iris	topics/iris.xml	Iris Topic

Conkeyref

Conrefend

Element name: abbreviated-form

Figure 160: Insert Content Key Reference Dialog

To reference target elements at sub-topic level just press the **Sub-topic** button and choose the target.

According to the DITA 1.2 specification the `conrefend` attribute can be used to specify content reference ranges. This is a very useful feature when referencing multiple consecutive steps or list items. If you use multiple contiguous sibling selection for IDs at sub-topic level the `conrefend` value will also be set to the value of the last selected ID path. Oxygen XML Editor plugin will present the entire referenced range as read-only content.

> **Important:** All keys which are presented in the dialog are gathered from the current opened DITA map. Elements which have the `conkeyref` attribute set are displayed by default with the target content expanded. The current opened DITA map is also used to resolve references when navigating `conkeyref` links in the Author mode.

- **Replace conref / conkeyref reference with content** - Replaces the content reference fragment or the `conkeyref` at caret position with the referenced content. This action is useful when you want to make changes to the content but decide to keep the referenced fragment unchanged.
- **Insert Equation** - Allows you to insert an MathML equation. .
- **Create Reusable Component** - Creates a reusable component from a selected fragment of text. For more information, see *Reusing Content*.
- **Insert Reusable Component** - Inserts a reusable component at cursor location. For more information, see *Reusing Content*.
- **Remove Content Reference** - Removes the `conref` attribute of an element. For more information, see *Reusing Content*.
- **Add/Edit Content Reference** - Add or edit the `conref` attribute of an element. For more information, see *Reusing Content*.
- **Paste as Content Reference** - Pastes the content of the clipboard as a content reference. Note that the copied element must have the `id` attribute set.
- **Paste as Link** - Pastes the content of the clipboard as a link. Note that the copied element must have the `id` attribute set.
- **Insert Ordered List** - Inserts an ordered list with one list item.
- **Insert Unordered List** - Inserts an unordered list with one list item.
- **Insert List Item** - Inserts a new list item for in any of the above two list types.
- **Insert Table** - Opens a dialog that allows you to configure the table to be inserted. The dialog allows the user to configure:
  - the number of rows and columns of the table

  - if the header will be generated
  - if the title will be added
  - how the table will be framed

- **Insert Row** - Inserts a new table row with empty cells. The action is available when the caret position is inside a table.
- **Insert Column** - Inserts a new table column with empty cells after the current column. The action is available when the caret position is inside a table.
- **Insert Cell** - Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, a new one will be inserted at caret's position. If the caret is inside a cell, then the new one will be created after the current cell.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.
- **Join Row Cells** - Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.
- **Join Cell Above** - Joins the content of cell from current caret position with that of the cell above it. Note that this action works only if both cells have the same column span.
- **Join Cell Below** - Joins the content of cell from current caret position with that of the cell below it. Note that this action works only if both cells have the same column span.
- **Split Cell To The Left** - Splits the cell from current caret position in two, inserting a new empty table cell to the left. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell To The Right** - Splits the cell from current caret position in two, inserting a new empty table cell to the right. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell Above** - Splits the cell from current caret position in two, inserting a new empty table cell above. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.
- **Split Cell Below** - splits the cell from current caret position in two, inserting a new empty table cell below. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.

Note: DITA supports the CALS table model similar with DocBook document type in addition to the `simpletable` element specific for DITA.

Caution: Column specifications are required for table actions to work properly.

- **Generate IDs** - Available in the contextual menu only, this action allows you to generate an unique ID for the element at caret position. If the element already has an id set, it is preserved. Further options are offered in the **ID Options** dialog, available in the **DITA** main menu. Here you can specify the elements for which Oxygen XML Editor plugin generates an ID if the **Auto generate ID's for elements** is enabled. If you want to keep already set element id's when copying content in the same document, make sure the **Remove ID's when copying content in the same document** option is not checked.
- **Search References** - Finds the references to the `id` attribute value of the selected element in all the topics from the current DITA map (opened in the **DITA Maps Manager** view). The default shortcut of the action is **Ctrl+Shift+G** and can be changed in the **DITA Topic** document type.

All actions described above are available in the contextual menu, the **DITA** submenu of the main menu or in the **Author custom actions** toolbar.

A drag and drop with a file from *the **Project** view* or from *the **DITA Maps Manager** view* to a DITA topic document that is edited in Author mode will create a link to the dragged file (the `xref` DITA element with the **href** attribute) at the drop location. A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `image` DITA element with the `href` attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

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 Editor plugin install directory.

The CSS file used for rendering DITA content is located in `${frameworks}/dita/css/dita.css`.

The default XML catalog is stored in `${frameworks}/dita/catalog.xml`.

Author Extensions

Specific actions for DITA map documents are:

- **Insert Topic Reference** - Inserts a reference to a topic.
- **Insert Content Reference** - Inserts a content reference at the caret position.
- **Insert Content Key Reference** - Inserts a content reference at the caret position.
- **Insert Table** - Opens a dialog that allows you to configure the relationship table to be inserted. The dialog allows the user to configure the number of rows and columns of the relationship table, if the header will be generated and if the title will be added.
- **Insert Row** - Inserts a new table row with empty cells. The action is available when the caret position is inside a table.
- **Insert Column** - Inserts a new table column with empty cells after the current column. The action is available when the caret position is inside a table.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.

All actions described above are available in the contextual menu, the **DITA** submenu of the main menu and in the **Author custom actions** toolbar.

A drag and drop with a file from *the **Project** view* or from *the **DITA Maps Manager** view* to a DITA map document that is edited in Author mode will create a link to the dragged file (a `topicref` element, a `chapter` one, a `part` one, etc.) at the drop location.

Transformation Scenarios

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 `bookmap` files.

Here are some of the DITA Map templates available when creating *new documents from templates*:

- **DITA Map - Bookmap** - New DITA Bookmap
- **DITA Map - Map** - New DITA Map

- **DITA Map - Learning Map** - New DITA learning and training content specialization map
- **DITA Map - Learning Bookmap** - New DITA learning and training content specialization bookmap
- **DITA Map - Eclipse Map** - New DITA learning and training content specialization bookmap

The XHTML Document Type

The Extensible HyperText Markup Language, or XHTML, is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax.

A file is considered to be a XHTML document when the root element name is a `html`.

The schema used for these documents is located in `${frameworks}/xhtml/dtd/xhtml1-strict.dtd`, where *${frameworks}* is a subdirectory of the Oxygen XML Editor plugin install directory.

The default CSS options for the XHTML document type are set to merge the CSSs specified in the document with the CSSs defined in the XHTML document type.

The CSS file used for rendering XHTML content is located in `${frameworks}/xhtml/css/xhtml.css`.

There are three default catalogs for XHTML document type:

- `${frameworks}/xhtml/dtd/xhtmlcatalog.xml`
- `${frameworks}/xhtml11/dtd/xhtmlcatalog.xml`
- `${frameworks}/xhtml11/schema/xhtmlcatalog.xml`

Author Extensions

Specific actions are:

- **B** **Bold** - Changes the style of the selected text to `bold` by surrounding it with `b` tag.
- *I* **Italic** - Changes the style of the selected text to `italic` by surrounding it with `i` tag.
- U **Underline** - Changes the style of the selected text to `underline` by surrounding it with `u` tag.

  Note:

  Bold, **Italic**, and **Underline** are toggle actions.

  For all of the above actions, if there is no selection, then a new specific tag will be inserted. These actions are available in any document context.

- H **Headings** - Groups actions for inserting `h1`, `h2`, `h3`, `h4`, `h5`, `h6` elements.
- ¶ **Insert Paragraph** - Inserts a new paragraph depending on the current context. If current context is a paragraph context (one of the ancestors of the element at caret position is `p`) then a new paragraph will be inserted after the paragraph at caret. Otherwise a new paragraph is inserted at caret position.
- **Insert Graphic** - Inserts a graphic object at the caret position. This is done by inserting an `img` element regardless of the current context. The following graphical formats are supported: GIF, JPG, JPEG, BMP, PNG, SVG.
- **Insert Ordered List** - Inserts an ordered list (`ol` element) with one list item (`li` child element).
- **Insert Unordered List** - Inserts an unordered list (`ul` element) with one list item (`li` child element).
- **Insert Definition List** - Inserts a definition list (`dl` element) with one list item (a `dt` child element and a `dd` child element).
- **Insert List Item** - Inserts a new list item for in any of the above two list types.
- **Insert Table** - Opens a dialog that allows you to configure the table to be inserted. The dialog allows the user to configure the number of rows and columns of the table, if the header and footer will be generated and how the table will be framed.

- **Insert Row** - Inserts a new table row with empty cells. The action is available when the caret position is inside a table.
- **Insert Column** - Inserts a new table column with empty cells after the current column. The action is available when the caret position is inside a table.
- **Insert Cell** - Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, a new one will be inserted at caret's position. If the caret is inside a cell, then the new one will be created after the current cell.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.
- **Join Row Cells** - Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.
- **Join Cell Above** - Joins the content of cell from current caret position with that of the cell above it. Note that this action works only if both cells have the same column span.
- **Join Cell Below** - Joins the content of cell from current caret position with that of the cell below it. Note that this action works only if both cells have the same column span.
- **Split Cell To The Left** - Splits the cell from current caret position in two, inserting a new empty table cell to the left. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell To The Right** - Splits the cell from current caret position in two, inserting a new empty table cell to the right. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell Above** - Splits the cell from current caret position in two, inserting a new empty table cell above. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.
- **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.

All actions described above are available in the contextual menu, the **XHTML** submenu of the main menu and in the **Author custom actions** toolbar.

A drag and drop with a file from *the **Project** view* or from *the **DITA Maps Manager** view* to a DITA topic document that is edited in Author mode will create a link to the dragged file (the `xref` DITA element with the **href** attribute) at the drop location. A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `image` DITA element with the `href` attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

Transformation Scenarios

The following default transformation scenarios are available for XHTML:

- **XHTML to DITA concept** - Converts an XHTML document to a DITA concept document
- **XHTML to DITA reference** - Converts an XHTML document to a DITA reference document
- **XHTML to DITA task** - Converts an XHTML document to a DITA task document
- **XHTML to DITA topic** - Converts an XHTML document to a DITA topic document

Templates

Default templates are available for XHTML. They are stored in `${frameworksDir}/xhtml/templates` folder and they can be used for easily creating basic XHTML documents.

Here are some of the XHTML templates available when creating *new documents from templates*.

- **XHTML - 1.0 Strict** - New Strict XHTML 1.0
- **XHTML - 1.0 Transitional** - New Transitional XHTML 1.0
- **XHTML - 1.1 DTD Based** - New DTD based XHTML 1.1
- **XHTML - 1.1 DTD Based + MathML 2.0 + SVG 1.1** - New XHTML 1.1 with MathML and SVG insertions
- **XHTML - 1.1 Schema based** - New XHTML 1.1 XML Schema based

The TEI ODD Document Type

The **Text Encoding Initiative - One Document Does it all (*TEI ODD*)** is a TEI XML-conformant specification format that allows creating a custom TEI P5 schema in a literate programming fashion. A system of XSLT stylesheets called *Roma* was created by the TEI Consortium for manipulating the ODD files.

A file is considered to be a TEI ODD document when either of the following occurs:

- the file extension is `.odd`
- the document's namespace is *http://www.tei-c.org/ns/1.0*

The schema used for these documents is located in `${frameworks}/tei/xml/tei/custom/schema/relaxng/brown_odds.rng`, where `${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering TEI ODD content is located in `${frameworks}/tei/xml/tei/css/tei_oxygen_odd.css`.

There are two default catalogs for TEI ODD document type:

- `${frameworks}/tei/xml/tei/custom/schema/catalog.xml`
- `${frameworks}/tei/xml/tei/schema/catalog.xml`

Author Extensions

The specific actions for TEI ODD documents are:

- **Bold** - Changes the style of the selected text to `bold` by surrounding it with `hi` tag and setting the `rend` attribute to `bold`.
- **Italic** - Changes the style of the selected text to `italic` by surrounding it with `hi` tag and setting the `rend` attribute to `italic`.
- **Underline** - Changes the style of the selected text to `underline` by surrounding it with `hi` tag and setting the `rend` attribute to `ul`.

  Note:

  Bold, **Italic**, and **Underline** are toggle actions.

  For all of the above actions, if there is no selection, then a new specific tag will be inserted. These actions are available in any document context.

- **Insert Section** - Inserts a new section / subsection, depending on the current context. For example if the current context is `div1` then a `div2` will be inserted and so on.
- **Insert Paragraph** - Inserts a new paragraph depending on the current context. If current context is a paragraph context (one of the ancestors of the element at caret position is `p`) then a new paragraph will be inserted after the paragraph at caret. Otherwise a new paragraph is inserted at caret position.
- **Insert Image** - Inserts a graphic object at the caret position. The following dialog is displayed allowing the user to specify the `entity` that refers the image itself.
- **Insert Ordered List** - Inserts an ordered list (`list` element with `type` attribute set to `ordered`) with one list item (`item` element).

- **Insert Itemized List** - Inserts an unordered list (`list` element with `type` attribute set to `bulleted`) with one list item (`item` element).
- **Insert List Item** - Inserts a new list item for in any of the above two list types.
- **Insert Table** - Opens a dialog that allows you to configure the table to be inserted. The dialog allows the user to configure the number of rows and columns of the table and if the header will be generated.
- **Insert Row** - Inserts a new table row with empty cells. The action is available when the caret position is inside a table.
- **Insert Column** - Inserts a new table column with empty cells after the current column. The action is available when the caret position is inside a table.
- **Insert Cell** - Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, a new one will be inserted at caret's position. If the caret is inside a cell, then the new one will be created after the current cell.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.
- **Join Row Cells** - Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.
- **Join Cell Above** - Joins the content of cell from current caret position with that of the cell above it. Note that this action works only if both cells have the same column span.
- **Join Cell Below** - Joins the content of cell from current caret position with that of the cell below it. Note that this action works only if both cells have the same column span.
- **Split Cell To The Left** - Splits the cell from current caret position in two, inserting a new empty table cell to the left. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell To The Right** - Splits the cell from current caret position in two, inserting a new empty table cell to the right. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell Above** - Splits the cell from current caret position in two, inserting a new empty table cell above. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.
- **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.
- **Generate IDs** - Available in the contextual menu only, this action allows you to generate an unique ID for the element at caret position. If the element already has an id set, it is preserved. Further options are offered in the **ID Options** dialog, available in the **TEI ODD** main menu. Here you can specify the elements for which Oxygen XML Editor plugin generates an ID if the **Auto generate ID's for elements** is enabled. If you want to keep already set element id's when copying content in the same document, make sure the **Remove ID's when copying content in the same document** option is not checked.

All actions described above are available in the contextual menu, the **TEI ODD** submenu of the main menu or in the **Author custom actions** toolbar.

A drag and drop with a file from *the **Project** view* or from *the **DITA Maps Manager** view* to a **TEI ODD** document that is edited in Author mode will create a link to the dragged file (the `ptr` element with the `target` attribute) at the drop location.

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 Editor plugin install directory.

The CSS file used for rendering TEI P4 content is located in `${frameworks}/tei/xml/tei/css/tei_oxygen.css`.

There are two default catalogs for TEI P4 document type:

- `${frameworks}/tei/xml/teip4/schema/dtd/catalog.xml`
- `${frameworks}/tei/xml/teip4/custom/schema/dtd/catalog.xml`

Author Extensions

The specific actions for TEI P4 documents are:

- B **Bold** - Changes the style of the selected text to `bold` by surrounding it with `hi` tag and setting the `rend` attribute to `bold`.
- I **Italic** - Changes the style of the selected text to `italic` by surrounding it with `hi` tag and setting the `rend` attribute to `italic`.
- U **Underline** - Changes the style of the selected text to `underline` by surrounding it with `hi` tag and setting the `rend` attribute to `ul`.

  Note:

  Bold, **Italic**, and **Underline** are toggle actions.

  For all of the above actions, if there is no selection, then a new specific tag will be inserted. These actions are available in any document context.

- **Browse reference manual** - Opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position.
- § **Insert Section** - Inserts a new section / subsection, depending on the current context. For example if the current context is `div1` then a `div2` will be inserted and so on.
- ¶ **Insert Paragraph** - Inserts a new paragraph depending on the current context. If current context is a paragraph context (one of the ancestors of the element at caret position is `p`) then a new paragraph will be inserted after the paragraph at caret. Otherwise a new paragraph is inserted at caret position.
- **Insert Image** - Inserts a graphic object at the caret position. The following dialog is displayed allowing the user to specify the `entity` that refers the image itself.
- **Insert Ordered List** - Inserts an ordered list (`list` element with `type` attribute set to `ordered`) with one list item (`item` element).
- **Insert Itemized List** - Inserts an unordered list (`list` element with `type` attribute set to `bulleted`) with one list item (`item` element).
- **Insert List Item** - Inserts a new list item for in any of the above two list types.
- **Insert Table** - Opens a dialog that allows you to configure the table to be inserted. The dialog allows the user to configure the number of rows and columns of the table and if the header will be generated.
- **Insert Row** - Inserts a new table row with empty cells. The action is available when the caret position is inside a table.
- **Insert Column** - Inserts a new table column with empty cells after the current column. The action is available when the caret position is inside a table.
- **Insert Cell** - Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, a new one will be inserted at caret's position. If the caret is inside a cell, then the new one will be created after the current cell.
- **Delete Column** - Deletes the table column where the caret is located.
- **Delete Row** - Deletes the table row where the caret is located.
- **Join Row Cells** - Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.
- **Join Cell Above** - Joins the content of cell from current caret position with that of the cell above it. Note that this action works only if both cells have the same column span.
- **Join Cell Below** - Joins the content of cell from current caret position with that of the cell below it. Note that this action works only if both cells have the same column span.
- **Split Cell To The Left** - Splits the cell from current caret position in two, inserting a new empty table cell to the left. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell To The Right** - Splits the cell from current caret position in two, inserting a new empty table cell to the right. Note that this action works only if the current cell spans over more than one column. The column span of the source cell will be decreased with one.
- **Split Cell Above** - Splits the cell from current caret position in two, inserting a new empty table cell above. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.
- **Split Cell Below** - Splits the cell from current caret position in two, inserting a new empty table cell below. Note that this action works only if the current cell spans over more than one row. The row span of the source cell will be decreased with one.
- **Generate IDs** - Available in the contextual menu only, this action allows you to generate an unique ID for the element at caret position. If the element already has an id set, it is preserved. Further options are offered in the **ID Options** dialog, available in the **TEI P4** main menu. Here you can specify the elements for which Oxygen XML Editor plugin

generates an ID if the **Auto generate ID's for elements** is enabled. If you want to keep already set element id's when copying content in the same document, make sure the **Remove ID's when copying content in the same document** option is not checked.

All actions described above are available in the contextual menu, the **TEI P4** submenu of the main menu or in the **Author custom actions** toolbar.

A drag and drop with a file from *the **Project** view* or from *the **DITA Maps Manager** view* to a TEI P4 document that is edited in Author mode will create a link to the dragged file (the `ptr` element with the `target` attribute) at the drop location.

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

Customization of TEI Frameworks

The *TEI P4 framework* and *TEI P5 one* are available as a public project at the following SVN repository:

```
https://oxygen-tei.googlecode.com/svn/trunk/
```

This project is the base for customizing a TEI framework.

1. Check out the project on a local computer from the SVN repository.

   This action is done with an SVN client application that creates a working copy of the SVN repository on a local computer.

2. Customize the TEI framework in Oxygen.

   a) Set the Oxygen `frameworks` folder to the folder of the SVN working copy.

      Go to menu **Options** > **Preferences** > **Global** and set the path of the SVN working copy in the option **Use custom frameworks**.

   b) Edit the TEI framework in the **Preferences** dialog.

      Go to menu **Options** > **Preferences** > **Document Type Association** and edit the TEI framework.

   c) Close the **Preferences** dialog with the **OK** button.

      If the dialog is closed with the **Cancel** button, the modifications are not saved on disk.

3. Build a jar file with the TEI framework.

   The SVN project includes a `build.xml` file that can be used for building a jar file using the Ant tool. The command that should be used:

   ```
   ant -f build.xml
   ```

4. Distribute the jar file to the users that need the customized TEI framework.

   The command from the above step creates a file `tei.zip` in the `dist` subfolder of the SVN project. Each user that needs the customized TEI framework will receive the `tei.zip` file and will unzip it in the `frameworks` folder of the Oxygen install folder. After restarting the Oxygen application the new TEI framework will appear in the **Options** > **Preferences** > **Document Type Association**.

The TEI P5 Document Type

The TEI P5 document type is 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 Editor plugin install directory.
- A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `graphic` DITA element with the `url` attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

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

Customization of TEI Frameworks

The *TEI P4 framework* and *TEI P5 one* are available as a public project at the following SVN repository:

```
https://oxygen-tei.googlecode.com/svn/trunk/
```

This project is the base for customizing a TEI framework.

1. Check out the project on a local computer from the SVN repository.

   This action is done with an SVN client application that creates a working copy of the SVN repository on a local computer.

2. Customize the TEI framework in Oxygen.

   a) Set the Oxygen `frameworks` folder to the folder of the SVN working copy.

   Go to menu **Options** > **Preferences** > **Global** and set the path of the SVN working copy in the option **Use custom frameworks**.

   b) Edit the TEI framework in the **Preferences** dialog.

   Go to menu **Options** > **Preferences** > **Document Type Association** and edit the TEI framework.

   c) Close the **Preferences** dialog with the **OK** button.

If the dialog is closed with the **Cancel** button, the modifications are not saved on disk.

3. Build a jar file with the TEI framework.

   The SVN project includes a `build.xml` file that can be used for building a jar file using the Ant tool. The command that should be used:

   ```
   ant -f build.xml
   ```

4. Distribute the jar file to the users that need the customized TEI framework.

   The command from the above step creates a file `tei.zip` in the `dist` subfolder of the SVN project. Each user that needs the customized TEI framework will receive the `tei.zip` file and will unzip it in the `frameworks` folder of the Oxygen install folder. After restarting the Oxygen application the new TEI framework will appear in the **Options** > **Preferences** > **Document Type Association**.

The EPUB Document Type

Three distinct frameworks support the EPUB document type:

- **NCX** - A declarative global navigation definition.
- **OCF** - The Open Container Format(OCF) defines a mechanism by which all components of an Open Publication Structure(OPS) can be combined into a single file-system entity.
- **OPF**: - The Open Packaging Format(OPF) defines the mechanism by which all components of a published work conforming to the Open Publication Structure(OPS) standard including metadata, reading order and navigational information are packaged into an OPS Publication.

Chapter 8

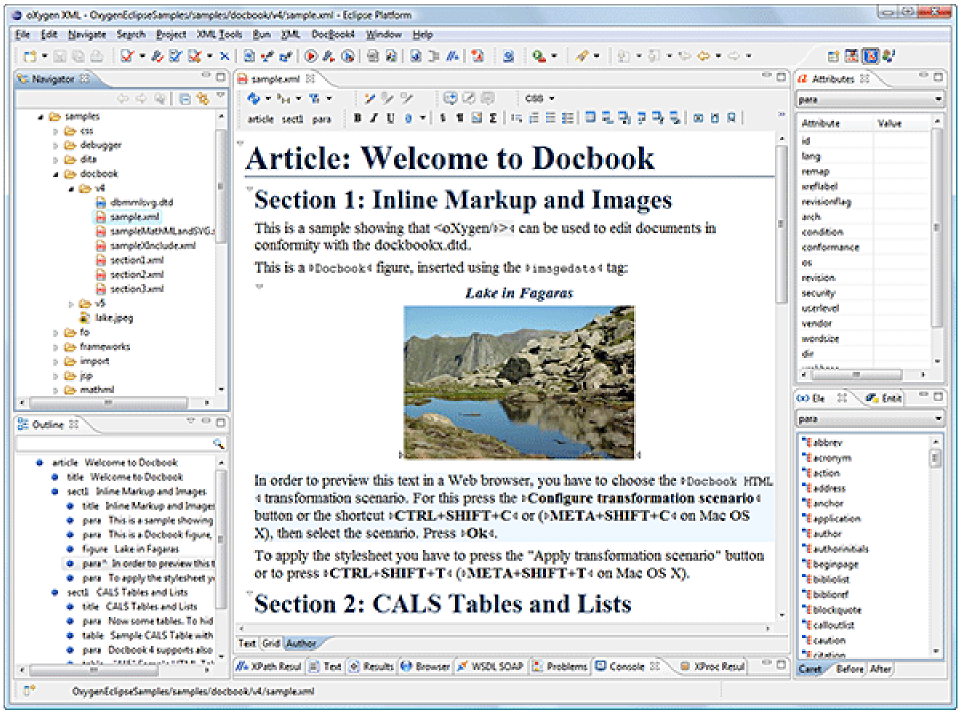
Author Developer Guide

Topics:

- *Simple Customization Tutorial*
- *Advanced Customization Tutorial - Document Type Associations*
- *CSS Support in Author*
- *Example Files Listings - The Simple Documentation Framework Files*
- *Author Component*

The Author editor of Oxygen XML Editor plugin was designed to bridge the gap between the XML source editing and a friendly user interface. The main achievement is the fact that the Author combines the power of source editing with the intuitive interface of a text editor.

This guide is targeted at advanced authors who want to customize the Author editing environment and is included both as a chapter in the Oxygen XML Editor plugin user manual and as a separate document in *the Author SDK*.



Article: Welcome to Docbook

Section 1: Inline Markup and Images

This is a sample showing that <oXygen/> can be used to edit documents in conformity with the dockbookx.dtd.

This is a Docbook figure, inserted using the imagedata tag:

Lake in Fagaras

In order to preview this text in a Web browser, you have to choose the Docbook HTML transformation scenario. For this press the **Configure transformation scenario** button or the shortcut **CTRL+SHIFT+C** or (**META+SHIFT+C** on Mac OS X), then select the scenario. Press **Ok**.

To apply the stylesheet you have to press the "Apply transformation scenario" button or to press **CTRL+SHIFT+T** (**META+SHIFT+T** on Mac OS X).

Section 2: CALS Tables and Lists

Figure 161: Oxygen XML Author Visual Editor

Although Oxygen XML Editor plugin comes with already configured frameworks for DocBook, DITA, TEI, XHTML, you might need to create a customization of the editor to handle other types of documents. The common use case is when your organization holds a collection of XML document types used to define the structure of internal documents and they need to be visually edited by people with no experience in working with XML files.

There are several ways to customize the editor:

1. Create a CSS file defining styles for the XML elements the user will work with, and create XML files that refer the CSS through an `xml-stylesheet` processing instruction.

2. Fully configure a document type association. This involves putting together the CSSs, the XML schemes, actions, menus, etc, bundling them and distributing an archive. The CSS and the GUI elements are settings of the Oxygen XML Author. The other settings like the templates, catalogs, transformation scenarios are general settings and are enabled whenever the association is active, no matter the editing mode (Text, Grid or Author).

Both approaches will be discussed in the following sections.

Simple Customization Tutorial

The most important elements of a document type customization are represented by an XML Schema to define the XML structure, the CSS to render the information and the XML instance template which links the first two together.

XML Schema

Let's consider the following XML Schema, `test_report.xsd` defining a report with results of a testing session. The report consists of a title, few lines describing the test suite that was run and a list of test results, each with a name and a boolean value indicating if the test passed or failed.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="report">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="title"/>
                <xs:element ref="description"/>
                <xs:element ref="results"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="description">
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
                <xs:element name="line">
                    <xs:complexType mixed="true">
                        <xs:sequence minOccurs="0"
                            maxOccurs="unbounded">
                            <xs:element name="important"
                              type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="results">
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
                <xs:element name="entry">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="test_name"
                              type="xs:string"/>
                            <xs:element name="passed"
                               type="xs:boolean"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

The use-case is that several users are testing a system and must send report results to a content management system. The Author customization should provide a visual editor for this kind of documents.

CSS Stylesheet

A set of rules must be defined for describing how the XML document is to be rendered into the Author. This is done using Cascading Style Sheets or CSS on short. CSS is a language used to describe how an HTML or XML document should be formatted by a browser. CSS is widely used in the majority of websites.

The elements from an XML document are displayed in the layout as a series of boxes. Some of the boxes contain text and may flow one after the other, from left to right. These are called in-line boxes. There are also other type of boxes that flow one below the other, like paragraphs. These are called block boxes.

For example consider the way a traditional text editor arranges the text. A paragraph is a block, because it contains a vertical list of lines. The lines are also blocks. But any block that contains inline boxes is arranging its children in a horizontal flow. That is why the paragraph lines are also blocks, but the traditional "bold" and "italic" sections are represented as inline boxes.

The CSS allows us to specify that some elements are displayed as tables. In CSS a table is a complex structure and consists of rows and cells. The "table" element must have children that have "table-row" style. Similarly, the "row" elements must contain elements with "table-cell" style.

To make it easy to understand, the following section describes the way each element from the above schema is formatted using a CSS file. Please note that this is just one from an infinite number of possibilities of formatting the content.

report

This element is the root element of the report document. It should be rendered as a box that contains all other elements. To achieve this the display type is set to **block**. Additionally some margins are set for it. The CSS rule that matches this element is:

```
report{
    display:block;
    margin:1em;
}
```

title

The title of the report. Usually titles have a larger font. The **block** display should also be used - the next elements will be placed below it, and change its font to double the size of the normal text.

```
title {
    display:block;
    font-size:2em;
}
```

description

This element contains several lines of text describing the report. The lines of text are displayed one below the other, so the description will have the same **block** display. To make it standout the background color is changed.

```
description {
    display:block;
    background-color:#EEEEFF;
    color:black;
}
```

line

A line of text in the description. A specific aspect is not defined for it, just indicate that the display should be **block**.

```
line {
    display:block;
}
```

important

The `important` element defines important text from the description. Because it can be mixed with text, its display property must be set to **inline**. To make it easier to spot, the text will be emphasized.

```
important {
    display:inline;
    font-weight:bold;
}
```

results

The `results` element shows the list of test_names and the result for each one. To make it easier to read, it is displayed as a **table** with a green border and margins.

```
results{
    display:table;
    margin:2em;
    border:1px solid green;
}
```

entry

An item in the results element. The results are displayed as a table so the entry is a row in the table. Thus, the display is **table-row**.

```
entry {
    display:table-row;
}
```

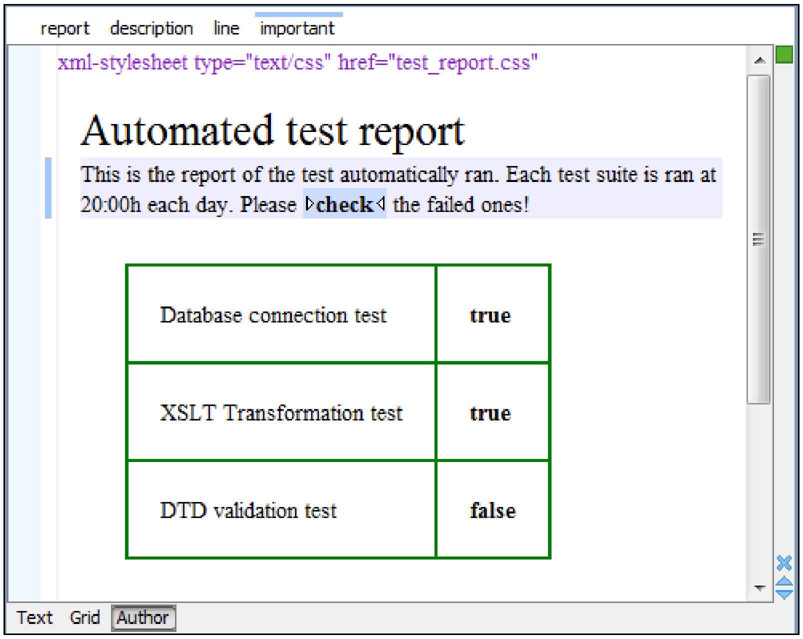
test_name, passed

The name of the individual test, and its result. They are cells in the results table with display set to **table-cell**. Padding and a border are added to emphasize the table grid.

```

test_name, passed{
    display:table-cell;
    border:1px solid green;
    padding:20px;
}

passed{
    font-weight:bold;
}
```

The full content of the CSS file `test_report.css` is:

```

report {
    display:block;
    margin:1em;
}

description {
    display:block;
    background-color:#EEEEFF;
    color:black;
}

line {
    display:block;
}

important {
    display:inline;
    font-weight:bold;
}

title {
    display:block;
    font-size:2em;
}

results{
    display:table;
    margin:2em;
    border:1px solid green;
}

entry {
    display:table-row;
}

test_name, passed{
    display:table-cell;
    border:1px solid green;
    padding:20px;
}

passed{
    font-weight:bold;
}
```



xml-stylesheet type="text/css" href="test_report.css"

Automated test report

This is the report of the test automatically ran. Each test suite is ran at 20:00h each day. Please **check** the failed ones!

Database connection test	**true**
XSLT Transformation test	**true**
DTD validation test	**false**

Figure 162: A report opened in the Author

The XML Instance Template

Based on the XML Schema and the CSS file the Author can help the content author in loading, editing and validating the test reports. An XML file template must be created, a kind of skeleton, that the users can use as a starting point for creating new test reports. The template must be generic enough and refer the XML Schema file and the CSS stylesheet.

This is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="test_report.css"?>
<report xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="test_report.xsd">
  <title>Automated test report</title>
  <description>
    <line>This is the report of the test automatically ran. Each test suite
is ran at 20:00h each
      day. Please <important>check</important> the failed ones!</line>
  </description>
  <results>
    <entry>
      <test_name>Database connection test</test_name>
      <passed>true</passed>
    </entry>
    <entry>
      <test_name>XSLT Transformation test</test_name>
      <passed>true</passed>
    </entry>
    <entry>
      <test_name>DTD validation test</test_name>
      <passed>false</passed>
    </entry>
  </results>
</report>
```

The processing instruction `xml-stylesheet` associates the CSS stylesheet to the XML file. The href pseudo attribute contains the URI reference to the stylesheet file. In our case the CSS is in the same directory as the XML file.

The next step is to place the XSD file and the CSS file on a web server and modify the template to use the HTTP URLs, like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
        href="http://www.mysite.com/reports/test_report.css"?>
<report xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation=
        "http://www.mysite.com/reports/test_report.xsd">
    <title>Test report title</title>
    <description>
.......
```

The alternative is to create an archive containing the `test_report.xml`, `test_report.css` and `test_report.xsd` and send it to the content authors.

Advanced Customization Tutorial - Document Type Associations

Oxygen XML is highly customizable. Practically you can associate an entire class of documents (grouped logically by some common features like namespace, root element name or filename) to a bundle consisting of CSS stylesheets, validation schemas, catalog files, new files templates, transformation scenarios and even custom actions. The bundle is called *document type* and the association is called *Document Type Association*.

In this tutorial a **Document Type Association** will be created for a set of documents. As an example a light documentation framework (similar to DocBook) will be created, then complete customization of the Author editor will be set up.

Note: The samples used in this tutorial can be found in the *Example Files Listings*.

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Editor plugin website*. Also it can be downloaded as a *zip archive from the website*.

Document Type

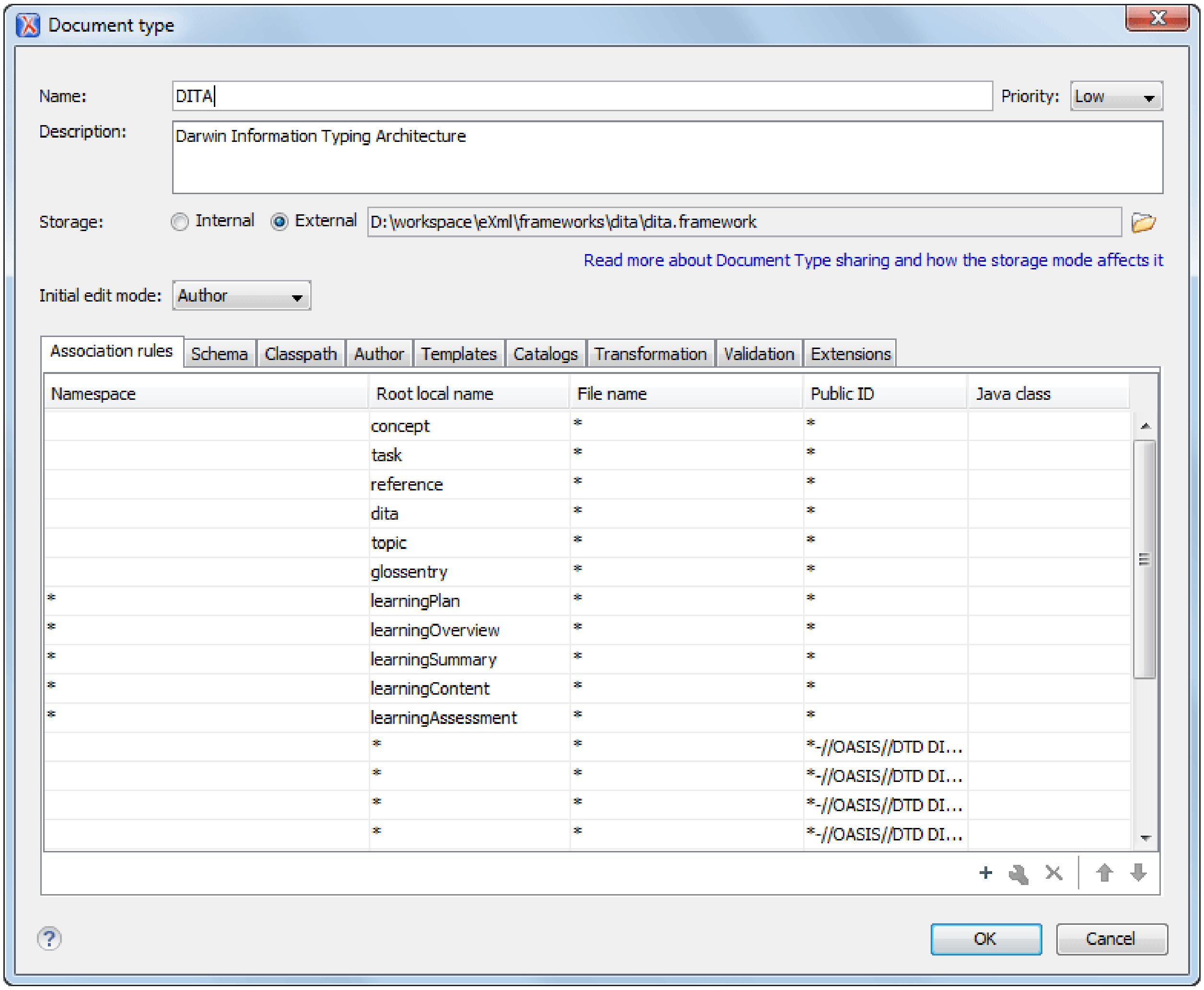
A *document type* or *framework* is associated to an XML file according to a set of rules. It includes also many settings that improve editing in the Tagless editor for the category of XML files it applies for. These settings include:

- a default grammar used for validation and content completion in both Author mode and Text mode
- CSS stylesheet(s) for rendering XML documents in Author mode
- user actions invoked from toolbar or menu in Author mode
- predefined scenarios used for transformation of the class of XML documents defined by the document type
- XML catalogs
- directories with file templates
- user defined extensions for customizing the interaction with the content author in Author mode

The tagless editor comes with some predefined document types already configured when the application is installed on the computer. These document types describe well-known XML frameworks largely used today for authoring XML documents. Editing a document which conforms to one of these types is as easy as opening it or creating it from one of the predefined document templates which also come with the application.

Author Settings

You can add a new *Document Type Association* or edit the properties of an existing one from the **Options** > **Preferences** > **Document Type Association** option pane. All the changes can be made into the *Document type* edit dialog.



Document type

Name: DITA

Priority: Low

Description: Darwin Information Typing Architecture

Storage: ○ Internal ● External D:\workspace\eXml\frameworks\dita\dita.framework

Read more about Document Type sharing and how the storage mode affects it

Initial edit mode: Author

Namespace	Root local name	File name	Public ID	Java class
	concept	*	*	
	task	*	*	
	reference	*	*	
	dita	*	*	
	topic	*	*	
	glossentry	*	*	
*	learningPlan	*	*	
*	learningOverview	*	*	
*	learningSummary	*	*	
*	learningContent	*	*	
*	learningAssessment	*	*	
	*	*	*-//OASIS//DTD DI...	
	*	*	*-//OASIS//DTD DI...	
	*	*	*-//OASIS//DTD DI...	
	*	*	*-//OASIS//DTD DI...	

Figure 163: The Document Type Dialog

You can specify the following properties for a document type:

- **Name** - The name of the document type.
- **Priority** - When there are multiple document types matching the same document, the order in which they are applied is determined by the priority. It can be one of: Lowest , Low, Normal, High, Highest. The predefined document types that are already configured when the application is installed on the computer have the default Low priority.

  Note: The frameworks having the same priority are alphabetically sorted.

- **Description** - The document type description displayed as a tooltip in the *Document Type Association table.*
- **Storage** - The location where the document type is saved. If you select the **External** storage, the document type is saved in the specified file with a mandatory `framework` extension, located in a subfolder of the current `frameworks` directory. If you select the **Internal** storage option, the document type data is saved in the current `.xpr` Oxygen XML Editor plugin project file (for Project-level Document Type Association Options) or in the Oxygen XML Editor plugin internal options (for Global-level Document Type Association Options). You can change the Document Type Association Options level in the *Document Type Association panel.*
- **Initial edit mode** - Allows you to select the initial editing mode (**Editor specific**, **Text**, **Author, Grid** and **Design** (available only for the W3C XML Schema editor)) for this document type. If the **Editor specific** option is selected, the initial edit mode is determined depending on the editor type. You can find the mapping between editors and edit modes in the ***Edit modes*** *preferences page.* You can decide to impose an initial mode for opening files which match the association rules of the document type. For example if the files are usually edited in the *Author* mode you can set it in the **Initial edit mode** combo box.

Note: The initial mode for an document type can also be customized from the **Edit modes** preferences panel located in **Window** > **Preferences** > **oXygen** > **Editor** > **Edit modes**.

You can specify the association rules used for determining a document type for an opened XML document. A rule can define one or more conditions. All conditions need to be fulfilled in order for a specific rule to be chosen. Conditions can specify:

- **Namespace** - The namespace of the document that matches the document type.
- **Root local name of document** - The local name of the document that matches the document type.
- **File name** - The file name (including the extension) of the document that matches the document type.
- **Public ID** (for DTDs) - The PUBLIC identifier of the document that matches the document type.
- **Java class** - Name of Java class that is called for finding if the document type should be used for an XML document. Java class must implement *`ro.sync.ecss.extensions.api.DocumentTypeCustomRuleMatcher`* interface from *Author API*.

In the **Schema** tab, you can specify the type and URI of schema used for validation and content completion of all documents from the document type, when there is no schema detected in the document.

You can choose one of the following schema types:

- DTD
- Relax NG schema (XML syntax)
- Relax NG schema (XML syntax) + Schematron
- Relax NG schema (compact syntax)
- XML Schema
- XML Schema + Schematron rules
- NVDL schema

Configuring Actions, Menus and Toolbars

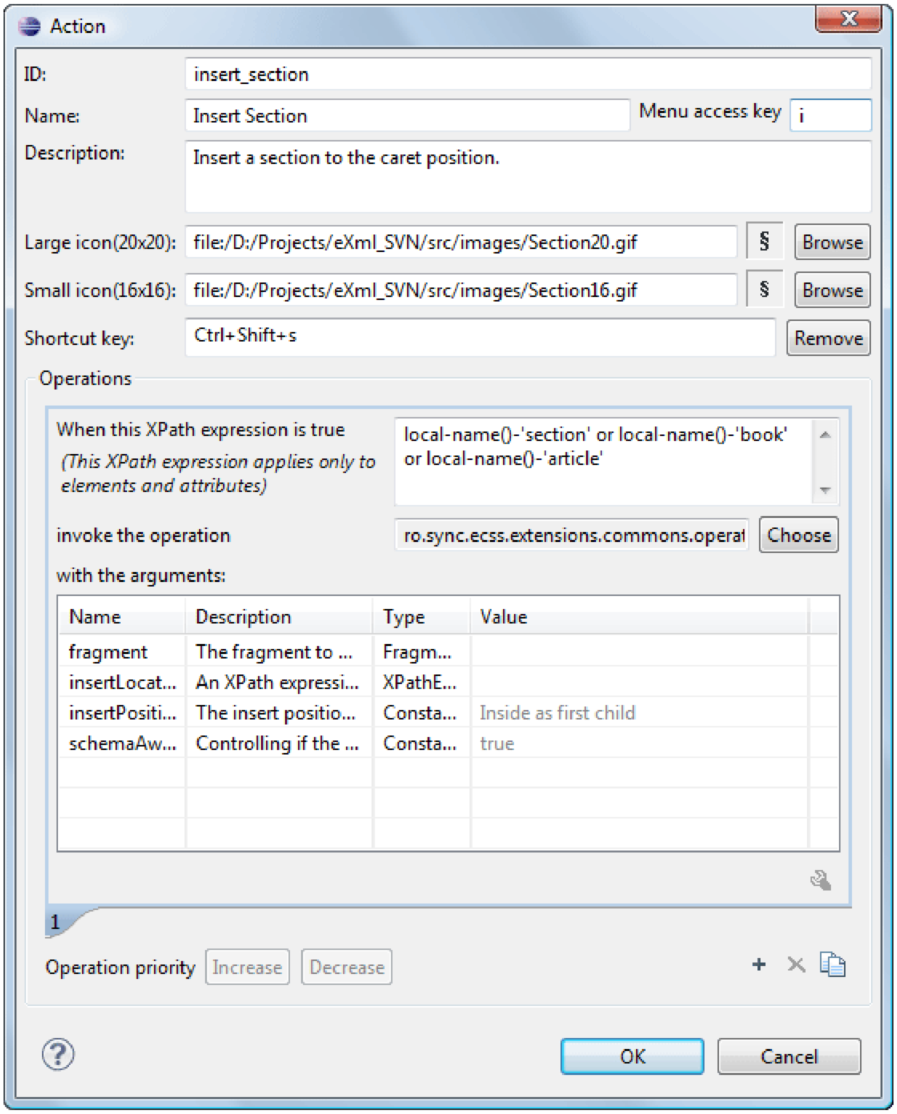
The Author toolbars and menus can be changed to provide a productive editing experience for the content authors. You can create a set of actions that are specific to a document type.

In the example with the `sdf` framework, you created the stylesheet and the validation schema. Now let's add some actions to insert a `section` and a `table`. To add a new action, follow the procedure:

1. Open the **Options Dialog**, and select the **Document Types Association** option pane.
2. In the lower part of the **Document Type Association** dialog, click on the **Author** tab, then select the **Actions** label.
3. To add a new action click on the + **Add** button.

The Insert Section Action

This section shows all the steps needed to define the Insert Section action. We assume the icon files § (`Section16.png`) for the menu item and § (`Section20.png`) for the toolbar, are already available. Although you could use the same icon size for both menu and toolbar, usually the icons from the toolbars are larger than the ones found in the menus. These files should be placed in the `frameworks / sdf` directory.



ID: insert_section

Name: Insert Section

Menu access key: i

Description: Insert a section to the caret position.

Large icon(20x20): file:/D:/Projects/eXml_SVN/src/images/Section20.gif

Small icon(16x16): file:/D:/Projects/eXml_SVN/src/images/Section16.gif

Shortcut key: Ctrl+Shift+s

Operations

When this XPath expression is true *(This XPath expression applies only to elements and attributes)*: local-name()-'section' or local-name()-'book' or local-name()-'article'

invoke the operation: ro.sync.ecss.extensions.commons.operat

with the arguments:

Name	Description	Type	Value
fragment	The fragment to ...	Fragm...	
insertLocat...	An XPath expressi...	XPathE...	
insertPositi...	The insert positio...	Consta...	Inside as first child
schemaAw...	Controlling if the ...	Consta...	true

1

Operation priority

Figure 164: The Action Edit Dialog

1. Set the **ID** field to **insert_section**. This is an unique action identifier.
2. Set the **Name** field to **Insert Section**. This will be the action's name, displayed as a tooltip when the action is placed in the toolbar, or as the menu item name.
3. Set the **Menu access key** to **i**. On Windows, the menu items can be accessed using (ALT + letter) combination, when the menu is visible. The letter is visually represented by underlining the first letter from the menu item name having the same value.
4. Set the **Description** field to **Insert a section at caret position**.
5. Set the **Large icon (20x20)** field to **${frameworks} / sdf / Section20.png**. A good practice is to store the image files inside the framework directory and use *editor variable* `${frameworks}` to make the image relative to the framework location.

   If the images are bundled in a jar archive together with some Java operations implementation for instance, it might be convenient for you to refer the images not by the file name, but by their relative path location in the class-path.

   If the image file `Section20.png` is located in the **images** directory inside the jar archive, you can refer to it by using **/images/Section20.png**. The jar file must be added into the **Classpath** list.
6. Set the **Small icon (16x16)** field to **${frameworks} / sdf / Section16.png**.

7. Click the text field next to **Shortcut key** and set it to **Ctrl+Shift+S**. This will be the key combination to trigger the action using the keyboard only.

   The shortcut is enabled only by *adding the action to the main menu of the Author mode* which contains all the actions that the author will have in a menu for the current document type.

8. At this time the action has no functionality added to it. Next you must define how this action operates. An action can have multiple operation modes, each of them activated by the evaluation of an XPath version 2.0 expression. The scope of the XPath expression must be only element nodes and attribute nodes of the edited document, otherwise the expression will not return a match and will not fire the action. For this example we'll suppose you want allow the action to add a section only if the current element is either a `book`, `article` or another `section`.

   a) Set the XPath expression field to:

   ```
   local-name()='section' or local-name()='book' or
    local-name()='article'
   ```

   b) Set the **invoke operation** field to `InsertFragmentOperation` built-in operation, designed to insert an XML fragment at caret position. This belongs to a set of built-in operations, a complete list of which can be found in the *Author Default Operations* section. This set can be expanded with your own Java operation implementations.

   c) Configure the arguments section as follows:

   ```
   <section xmlns=
   "http://www.oxygenxml.com/sample/documentation">
        <title/>
   </section>
   ```

   `insertLocation` - leave it empty. This means the location will be at the caret position.

   `insertPosition` - select "**Inside**".

The Insert Table Action

You will create an action that inserts into the document a table with three rows and three columns. The first row is the table header. Similarly to the insert section action, you will use the `InsertFragmentOperation`.

Place the icon files for the menu item and for the toolbar in the `frameworks / sdf` directory.

1. Set **ID** field to **insert_table**.
2. Set **Name** field to **Insert table**.
3. Set **Menu access key** field to **t**.
4. Set **Description** field to **Adds a section element**.
5. Set **Toolbar icon** to **${frameworks} / sdf / toolbarIcon.png**.
6. Set **Menu icon** to **${frameworks} / sdf / menuIcon.png**.
7. Set **Shortcut key** to **Ctrl+Shift+T**.
8. Set up the action's functionality:

   a) Set **XPath expression** field to `true()`.

   `true()` is equivalent with leaving this field empty.

   b) Set **Invoke operation** to use **InvokeFragmentOperation** built-in operation that inserts an XML fragment to the caret position.

   c) Configure operation's arguments as follows:

   fragment - set it to:

   ```
   <table xmlns=
   "http://www.oxygenxml.com/sample/documentation">
     <header><td/><td/><td/></header>
   ```

```
  <tr><td/><td/><td/></tr>
  <tr><td/><td/><td/></tr>
</table>
```

`insertLocation` - to add tables at the end of the section use the following code:

```
ancestor::section/*[last()]
```

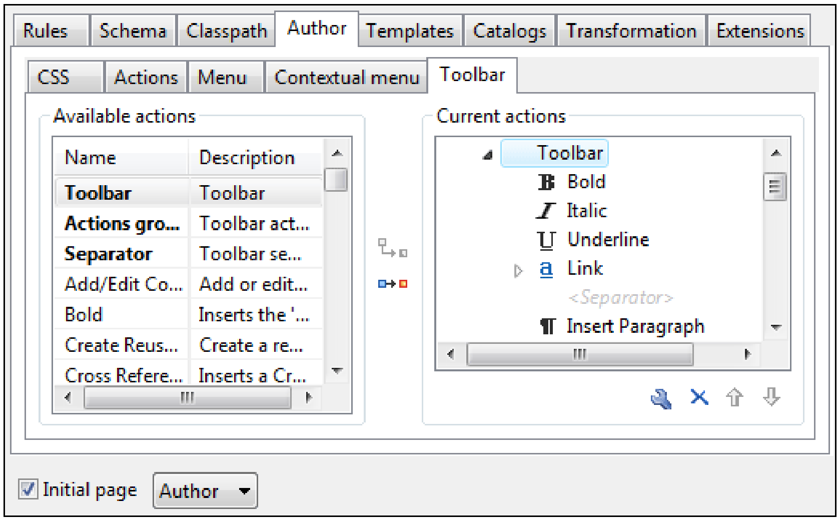
`insertPosition` - Select **After**.

Configuring the Toolbars

Now that you have defined the ***Insert Section*** action and the ***Insert Table*** action, you can add them to the toolbar. You can configure additional toolbars on which to add your custom actions.

The first thing to check is that the toolbar **Author custom actions 1** is displayed when switching to the **Author** mode: right click in the upper part of application window, in the area that contains the toolbar buttons and check if **Author custom actions 1** in the displayed menu if it is unchecked.

1. Open the Document Type edit dialog for the **SDF** framework and select on the **Author** tab. Next click on the **Toolbar** label.



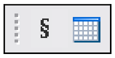
Name	Description
Toolbar	Toolbar
Actions gro...	Toolbar act...
Separator	Toolbar se...
Add/Edit Co...	Add or edit...
Bold	Inserts the '...
Create Reus...	Create a re...
Cross Refere...	Inserts a Cr...

Figure 165: Configuring the Toolbar

The panel is divided in two sections: the left side contains a list of actions, while the right one contains an action tree, displaying the list of actions added in the toolbar. The special entry called *Separator* allows you to visually separate the actions in the toolbar.

2. Select the **Insert section** action in the left panel section and the **Toolbar** label in the right panel section, then press the **Add as child** button.
3. Select the **Insert table** action in the left panel section and the **Insert section** in the right panel section. Press the **Add as sibling** button.
4. When opening a **Simple Documentation Framework** test document in Author mode, the toolbar below will be displayed at the top of the editor.

Figure 166: Author Custom Actions Toolbar

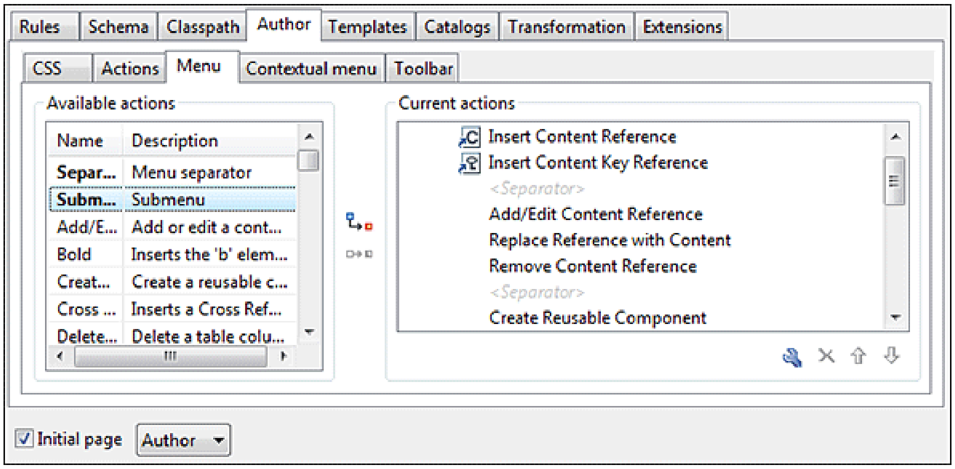


Tip: If you have many custom toolbar actions or want to group actions according to their category you can add additional toolbars with custom names and split the actions to better suit your purpose.

Configuring the Main Menu

Defined actions can be grouped into customized menus in the Oxygen XML Editor plugin menu bar.

1. Open the Document Type dialog for the **SDF** framework and click on the **Author** tab.
2. Click on the **Menu** label. In the left side you have the list of actions and some special entries:
   - **Submenu** - Creates a submenu. You can nest an unlimited number of menus.
   - **Separator** - Creates a separator into a menu. This way you can logically separate the menu entries.
3. The right side of the panel displays the menu tree with **Menu** entry as root. To change its name click on this label to select it, then press the **Edit** button. Enter **SD Framework** as name, and **D** as menu access key.
4. Select the **Submenu** label in the left panel section and the **SD Framework** label in the right panel section, then press the **Add as child** button. Change the submenu name to **Table**, using the **Edit** button.
5. Select the **Insert section** action in the left panel section and the **Table** label in the right panel section, then press the **Add as sibling** button.
6. Now select the **Insert table** action in the left panel section and the **Table** in the right panel section. Press the **Add as child** button.



Name	Description
Separ...	Menu separator
Subm...	Submenu
Add/E...	Add or edit a cont...
Bold	Inserts the 'b' elem...
Creat...	Create a reusable c...
Cross ...	Inserts a Cross Ref...
Delete...	Delete a table colu...

Insert Content Reference
Insert Content Key Reference
<Separator>
Add/Edit Content Reference
Replace Reference with Content
Remove Content Reference
<Separator>
Create Reusable Component

☑ Initial page: Author

Figure 167: Configuring the Menu

When opening a **Simple Documentation Framework** test document in Author mode, the menu you created is displayed in the editor menu bar, between the **Tools** and the **Document** menus. The upper part of the menu contains generic Author actions (common to all document types) and the two actions created previously (with **Insert table** under the **Table** submenu).

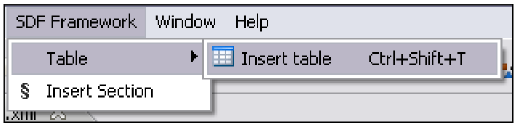
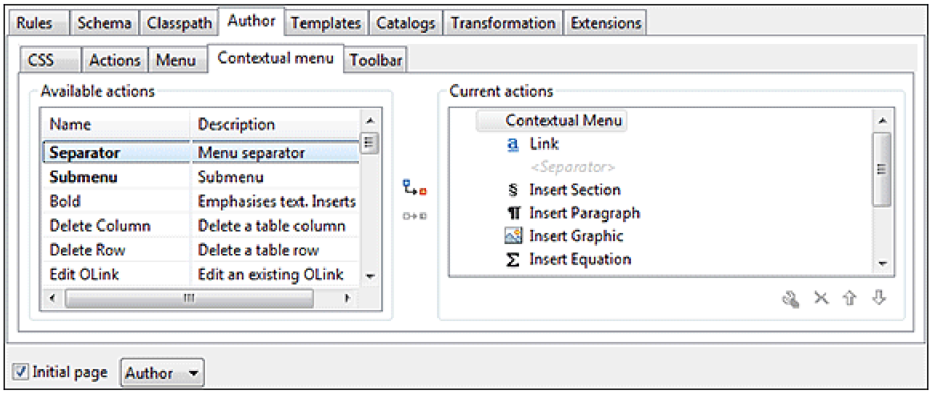


Figure 168: Author Menu

Configuring the Contextual Menu

The contextual menu is shown when you right click (**ctrl + mouse click** on Mac) in the Author editing area. In fact you are configuring the bottom part of the menu, since the top part is reserved for a list of generic actions like Copy, Paste, Undo, etc.

1. Open the Document Type dialog for the **SDF** framework and click on the **Author** tab. Next click on the **Contextual Menu** label.
2. Follow the same steps as explained in the *Configuring the Main Menu*, except changing the menu name because the contextual menu does not have a name.



Name	Description
Separator	Menu separator
Submenu	Submenu
Bold	Emphasises text. Inserts
Delete Column	Delete a table column
Delete Row	Delete a table row
Edit OLink	Edit an existing OLink

Contextual Menu
- Link
- <Separator>
- Insert Section
- Insert Paragraph
- Insert Graphic
- Insert Equation

☑ Initial page: Author

Figure 169: Configuring the Contextual Menu

To test it, open the test file, and open the contextual menu. In the lower part there is shown the **Table** sub-menu and the **Insert section** action.

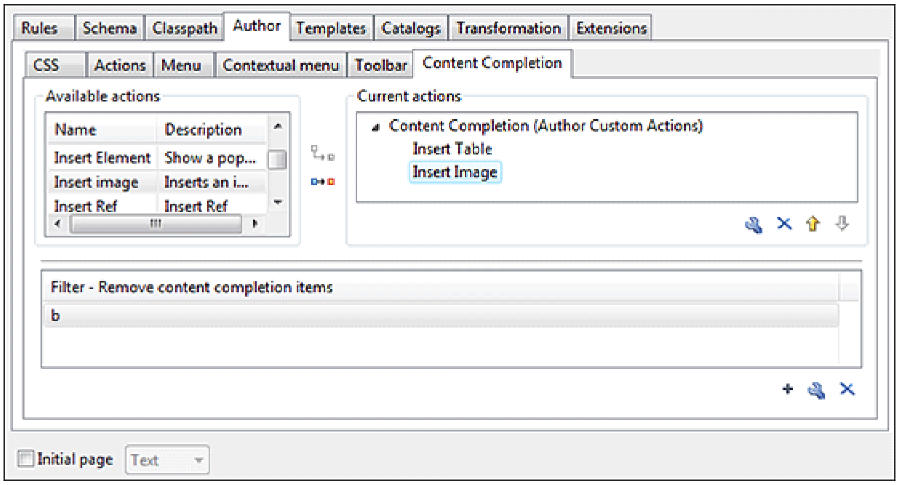
Customize Content Completion

You can customize the content of the following **Author** controls, adding items (which, when invoked, perform custom actions) or filtering the default contributed ones:

- Content Completion window;
- **Elements** view;
- **Element Insert** menus (from the **Outline** view or breadcrumb contextual menus).

You can use the content completion customization support in the *Simple Documentation Framework* following the next steps:

1. Open the **Document type** edit dialog for the **SDF** framework and select the **Author** tab. Next click on the **Content Completion** tab.



Name	Description
Insert Element	Show a pop...
Insert image	Inserts an i...
Insert Ref	Insert Ref

Content Completion (Author Custom Actions)
- Insert Table
- Insert Image

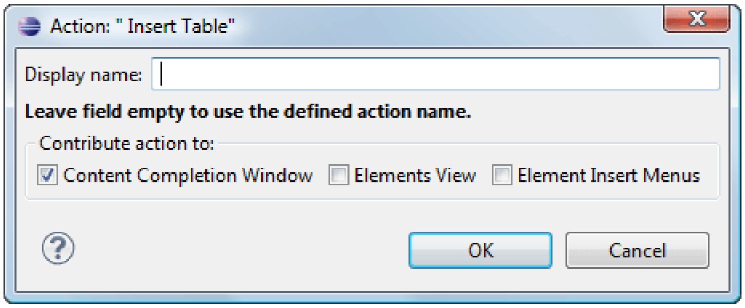
Filter - Remove content completion items
b

□ Initial page: Text

Figure 170: Customize content completion

The top side of the **Content Completion** section contains the list with all the actions defined within the simple documentation framework and the list of actions that you decided to include in the **Content Completion** items lists. The bottom side contains the list with all the items that you decided to remove from the **Content Completion** items lists.

2. If you want to add a custom action to the list of current **Content Completion** items, select the action item from the **Available actions** list and press the **Add as child** or **Add as sibling** button to include it in the **Current actions** list. The following dialog appears, giving you the possibility to select where to provide the selected action:



Display name:

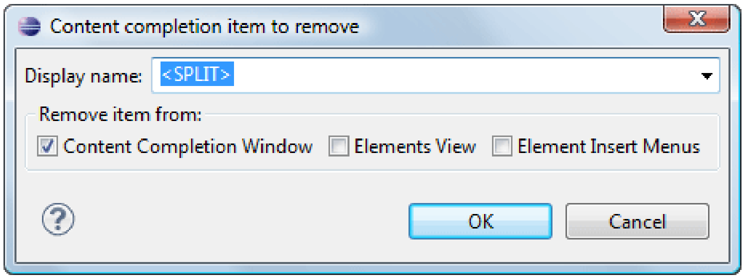
Leave field empty to use the defined action name.

Contribute action to:

☑ Content Completion Window □ Elements View □ Element Insert Menus

Figure 171: Insert action dialog

3. If you want to exclude a certain item from the **Content Completion** items list, you can use the + **Add** button from the **Filter - Remove content completion items** list. The following dialog is displayed, allowing you to input the item name and to choose the controls that filter it.



Display name: <SPLIT>

Remove item from:

☑ Content Completion Window □ Elements View □ Element Insert Menus

Figure 172: Remove item dialog

Author Default Operations

Below are listed all the operations and their arguments.

`InsertFragmentOperation`
Inserts an XML fragment at the current cursor position. The selection - if there is one, remains unchanged. The fragment will be inserted in the current context of the cursor position meaning that if the current XML document uses some namespace declarations then the inserted fragment must use the same declarations. The inserted fragment will not be copied and pasted to the cursor position, but the namespace declarations of the fragment will be adapted if needed to the existing namespace declarations of the XML document.

`InsertOrReplaceFragmentOperation`
Similar to `InsertFragmentOperation`, except it removes the selected content before inserting the fragment.

`InsertOrReplaceTextOperation`
Inserts a text at current position removing the selected content, if any.

`text` The text section to insert.

`SurroundWithFragmentOperation`
Surrounds the selected content with a text fragment. Since the fragment can have multiple nodes, the surrounded content will be always placed in the first leaf element. If there is no selection, the operation will simply insert the fragment at the caret position.

`SurroundWithTextOperation`
This operation has two arguments (two text values) that will be inserted before and after the selected content. If there is no selected content, the two sections will be inserted at the caret position. The arguments of the operation are:

`header`	The text that will be placed before the selection.
`footer`	The text that will be placed after the selection.

`InsertEquationOperation`

Inserts a fragment containing a MathML equation at caret offset. The operation argument is:

`fragment`	The XML fragment containing the MathML content which should be inserted

`InsertXIncludeOperation`

Insert an XInclude element at caret offset.

`ChangeAttributeOperation`

This operation allows adding/modifying/removing an attribute. You can use this operation in your own Author action to modify the value for a certain attribute on a specific XML element. The arguments of the operation are:

`name`	The attribute local name.
`namespace`	The attribute namespace.
`elementLocation`	The XPath location that identifies the element.
`value`	The new value for the attribute. If empty or null the attribute will be removed.

`UnwrapTagsOperation`

This operation allows removing the element tags either from the current element or for an element identified with an XPath location. The argument of the operation is:

`unwrapElementLocation`	An XPath expression indicating the element to unwrap. If it is not defined, then the element at caret is unwrapped.

`ToggleSurroundWithElementOperation`

This operation allows wrapping and unwrapping content in a specific element with specific attributes. Useful to implement toggle actions like highlighting text as bold, italic, or underline. The arguments of the operation are:

`element`	The element to wrap content (or unwrap).
`schemaAware`	This argument applies only on the surround with element operation and controls if the insertion is schema aware or not.

Author operations can take parameters that might contain the following editor variables:

- **${caret}** - The position where the caret is inserted. This variable can be used in a *code template* , in Author operations, or in a selection plugin.
- **${selection}** - The XML content of the current selection in the editor panel. This variable can be used in a *code template* and Author operations,
- **${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 Editor plugin checks that the URL is valid before passing it to the transformation.
    - `password` - input characters are hidden.
    - `generic` - the input is treated as generic text that requires no special handling.
  - `'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 Editor plugin installation folder.
- **${pd}** - Current project folder as file path.
- **${oxygenInstallDir}** - Oxygen XML Editor plugin installation folder as file path.
- **${homeDir}** - The path (as file path) of the user home folder.
- **${pn}** - Current project name.
- **${env(VAR_NAME)}** - Value of the *VAR_NAME* environment variable.
- **${system(var.name)}** - Value of the *var.name* system variable.
- **${date(pattern)}** - Current date. Follows the given pattern. Example: yyyy-MM-dd.

The arguments of `InsertFragmentOperation` operation

fragment

The value for this argument is a text. This is parsed by Oxygen XML Author as it was already in the document at the caret position. You can use entity references declared in the document and it is namespace aware. The fragment may have multiple roots.

> **Note:**
>
> You can even use namespace prefixes that are not declared in the inserted fragment, if they are declared in the document where the insertion is done. For the sake of clarity, you should always prefix and declare namespaces in the inserted fragment!

> **Note:**
>
> If the fragment contains namespace declarations that are identical to those found in the document, the namespace declaration attributes will be removed from elements contained by the inserted fragment.

There are two possible scenarios:

1. **Prefixes that are not bound explicitly**

   For instance, the fragment:

   ```
   <x:item id="dty2"/>
   &ent;
   <x:item id="dty3"/>
   ```

   Can be correctly inserted in the document: ('|' marks the insertion point):

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE x:root [
       <!ENTITY ent "entity">
   ]>

   <x:root xmlns:x="nsp">
     |
   </x:root>
   ```

   Result:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE x:root [
       <!ENTITY ent "entity">
   ]>
   <x:root xmlns:x="nsp">
       <x:item id="dty2"/>
       &ent;
       <x:item id="dty3"/>
   </x:root>
   ```

2. **Default namespaces**

If there is a default namespace declared in the document and the document fragment does not declare a namespace, the elements from the fragment are considered to be in **no namespace**.

For instance the fragment:

```
<item id="dty2"/>
<item id="dty3"/>
```

Inserted in the document:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="nsp">
|
</root>
```

Gives the result document:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="nsp">
    <item xmlns="" id="dty2"/>
    <item xmlns="" id="dty3"/>
</root>
```

`insertLocation` An XPath expression that is relative to the current node. It selects the reference node for the fragment insertion.

`insertPosition` One of the three constants: "**Inside**", "**After**", or "**Before**" , showing where the insertion is made relative to the reference node selected by the `insertLocation`. "**Inside**" has the meaning of the first child of the reference node.

The arguments of `SurroundWithFragmentOperation`

The Author operation `SurroundWithFragmentOperation` has only one argument:

- fragment -

  The XML fragment that will surround the selection. For example let's consider the fragment:

```
<F>
   <A></A>
   <B>
     <C></C>
   </B>
</F>
```

  and the document:

```
<doc>
  <X></X>
  <Y></Y>
  <Z></Z>
<doc>
```

  Considering the selected content to be surrounded is the sequence of elements `X` and `Y`, then the result is:

```
<doc>
   <F>
      <A>
         <X></X>
         <Y></Y>
      </A>
      <B>
        <C></C>
      </B>
   </F>
  <Z></Z>
<doc>
```

  Because the element `A` was the first leaf in the fragment, it received the selected content. The fragment was then inserted in the place of the selection.

How to Add a Custom Action to an Existing Document Type

This task explains how to add a custom Author operation to an existing document type.

1. Download the Author SDK toolkit:*http://www.oxygenxml.com/developer.html#XML_Editor_Authoring_SDK*
2. Create a Java project with a custom implementation of *ro.sync.ecss.extensions.api.AuthorOperation* which performs your custom operation and updates the Author mode using our API like: `AuthorAccess.getDocumentController().insertXMLFragment`.
3. Pack the operation class inside a Java *jar* library.
4. Copy the *jar* library to the `OXYGEN_INSTALL_DIR/frameworks/framework_dir` directory.
5. Go to Oxygen **Preferences** > **Document Type Association** page and edit the document type (you need write access to the `OXYGEN_INSTALLATION_DIR`).
   a) In the **Classpath** tab, add a new entry like: `${frameworks}/docbook/customAction.jar`.
   b) In the **Author** tab, add a new action which uses your custom operation.
   c) Mount the action to the toolbars or menus.

6. Share the modifications with your colleagues. The files which should be shared are your `customAction.jar` library and the `.framework` configuration file from the `OXYGEN_INSTALL_DIR/frameworks/framework_dir` directory.

Java API - Extending Author Functionality through Java

Oxygen XML Author has a built-in set of operations covering the insertion of text and XML fragments (see the *Author Default Operations*) and the execution of XPath expressions on the current document edited in Author mode. However, there are situations in which you need to extend this set. For instance if you need to enter an element whose attributes should be edited by the user through a graphical user interface. Or the users must send the selected element content or even the whole document to a server, for some kind of processing or the content authors must extract pieces of information from a server and insert it directly into the edited XML document. Or you need to apply an XPath expression on the current Author document and process the nodes of the result nodeset.

The following sections contain the Java programming interface (API) available to the developers. You will need the *Oxygen Author SDK* available *on the Oxygen XML Editor plugin website* which includes the source code of the Author operations in the predefined document types and the full documentation in Javadoc format of the public API available for the developer of Author custom actions.

The next Java examples are making use of AWT classes. If you are developing extensions for the Oxygen XML Editor plugin XML Editor plugin for Eclipse you will have to use their SWT counterparts.

It is assumed you already read the *Configuring Actions, Menus, Toolbar* section and you are familiar with the Oxygen XML Editor plugin Author customization. You can find the XML schema, CSS and XML sample in the *Example Files Listings*.

> **Attention:**
>
> Make sure the Java classes of your custom Author operations are compiled with the same Java version used by Oxygen XML Editor plugin . Otherwise the classes may not be loaded by the Java virtual machine. For example if you run with a Java 1.6 virtual machine but the Java classes of your custom Author operations are compiled with a Java 1.7 virtual machine then the custom operations cannot be loaded and used by the Java 1.6 virtual machine.

Example 1. Step by Step Example. Simple Use of a Dialog from an Author Operation.

Let's start adding functionality for inserting images in the **Simple Documentation Framework** (shortly SDF). The images are represented by the `image` element. The location of the image file is represented by the value of the `href` attribute. In the Java implementation you will show a dialog with a text field, in which the user can enter a full URL, or he can browse for a local file.

1. Create a new Java project, in your IDE of choice. Create the `lib` folder in the project folder. Copy the `oxygen.jar` file from the `{oXygen_installation_directory}/lib` folder into the newly created `lib` folder. `oxygen.jar` contains the Java interfaces you have to implement and the API needed to access the Author features.
2. Create the `simple.documentation.framework.InsertImageOperation` class that implements the *`ro.sync.ecss.extensions.api.AuthorOperation`* interface. This interface defines three methods: `doOperation`, `getArguments` and `getDescription`

   A short description of these methods follows:

   - The `doOperation` method is invoked when the action is performed either by pressing the toolbar button, by selecting the menu item or by pressing the shortcut key. The arguments taken by this methods can be one of the following combinations:
     - an object of type *`ro.sync.ecss.extensions.api.AuthorAccess`* and a map
     - argument names and values
   - The `getArguments` method is used by Oxygen XML Editor plugin when the action is configured. It returns the list of arguments (name and type) that are accepted by the operation.
   - The `getDescription` method is used by Oxygen XML Editor plugin when the operation is configured. It returns a description of the operation.

Here is the implementation of these three methods:

```
/**
 * Performs the operation.
 */
public void doOperation(
               AuthorAccess authorAccess,
               ArgumentsMap arguments)
 throws IllegalArgumentException,
                  AuthorOperationException {

 JFrame oxygenFrame = (JFrame) authorAccess.getParentFrame();
 String href = displayURLDialog(oxygenFrame);
 if (href.length() != 0) {
     // Creates the image XML fragment.
     String imageFragment =
       "<image xmlns='http://www.oxygenxml.com/sample/documentation' href='"

        + href + "'/>";

     // Inserts this fragment at the caret position.
     int caretPosition = authorAccess.getCaretOffset();
     authorAccess.insertXMLFragment(imageFragment, caretPosition);
 }
}

/**
 * Has no arguments.
 *
 * @return null.
 */
public ArgumentDescriptor[] getArguments() {
 return null;
}

/**
 * @return A description of the operation.
 */
public String getDescription() {
 return "Inserts an image element. Asks the user for a URL reference.";
}
```

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

Important:

Make sure you always specify the namespace of the inserted fragments.

```
<image xmlns='http://www.oxygenxml.com/sample/documentation'
  href='path/to/image.png'/>
```

3. Package the compiled class into a jar file. An example of an ANT script that packages the `classes` folder content into a jar archive named `sdf.jar` is listed below:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="project" default="dist">
    <target name="dist">
   <jar destfile="sdf.jar" basedir="classes">
    <fileset dir="classes">
     <include name="**/*"/>
     </fileset>
   </jar>
    </target>
</project>
```

4. Copy the `sdf.jar` file into the `frameworks / sdf` folder.
5. Add the `sdf.jar` to the Author class path. To do this, open the **Options** > **Preferences** > **Document Type Association** dialog, select **SDF** and press the **Edit** button.
6. Select the **Classpath** tab in the lower part of the dialog and press the + **Add** button . In the displayed dialog enter the location of the jar file, relative to the Oxygen XML Editor plugin `frameworks` folder.
7. Let's create now the action which will use the defined operation. Click on the **Actions** label. Copy the icon files for the menu item and for the toolbar in the `frameworks / sdf` folder.
8. Define the action's properties:

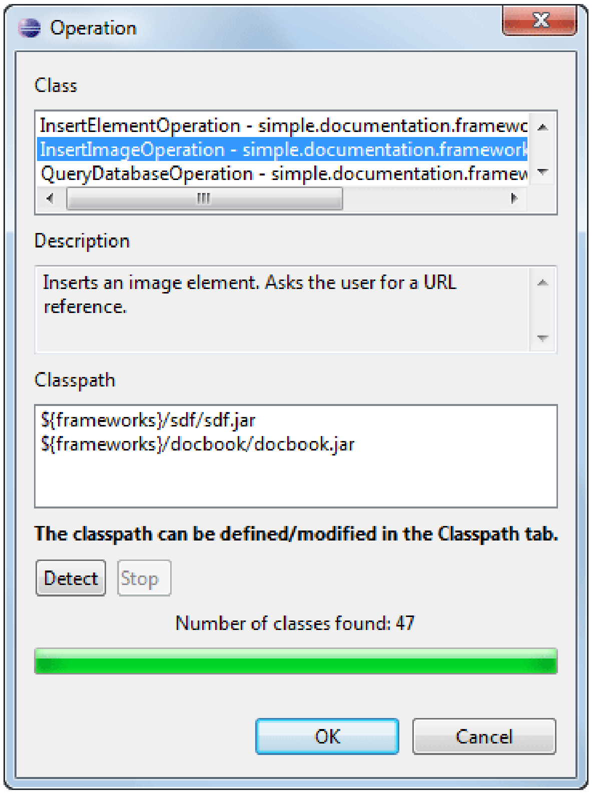
   - Set **ID** to **insert_image**.
   - Set **Name** to **Insert image**.
   - Set **Menu access key** to letter **i**.
   - Set **Toolbar action** to **${frameworks}/sdf/toolbarImage.png**.
   - Set **Menu icon** to **${frameworks}/sdf/menuImage.png**.
   - Set **Shortcut key** to **Ctrl+Shift+i**.

9. Now let's set up the operation. You want to add images only if the current element is a `section`, `book` or `article`.

   - Set the value of **XPath expression** to

     ```
     local-name()='section' or local-name()='book'
      or local-name()='article'
     ```

   - Set the **Invoke operation** field to `simple.documentation.framework.InsertImageOperation`.



InsertElementOperation - simple.documentation.framewc
InsertImageOperation - simple.documentation.framework
QueryDatabaseOperation - simple.documentation.framew

Inserts an image element. Asks the user for a URL reference.

${frameworks}/sdf/sdf.jar
${frameworks}/docbook/docbook.jar

The classpath can be defined/modified in the Classpath tab.

Number of classes found: 47

Figure 173: Selecting the Operation

10. Add the action to the toolbar, using the **Toolbar** panel.

To test the action, you can open the *sdf_sample.xml* sample, then place the caret inside a `section` between two para elements for instance. Press the button associated with the action from the toolbar. In the dialog select an image URL and press **OK**. The image is inserted into the document.

Example 2. Operations with Arguments. Report from Database Operation.

In this example you will create an operation that connects to a relational database and executes an SQL statement. The result should be inserted in the edited XML document as a `table`. To make the operation fully configurable, it will have arguments for the *database connection string*, the *user name*, the *password* and the *SQL expression*.

1. Create a new Java project in your preferred IDE. Create the `lib` folder in the Java project directory and copy the `oxygen.jar` file from the `{oXygen_installation_directory}/lib` directory.
2. Create the class `simple.documentation.framework.QueryDatabaseOperation`. This class must implements the *ro.sync.ecss.extensions.api.AuthorOperation* interface.

```

import ro.sync.ecss.extensions.api.ArgumentDescriptor;
import ro.sync.ecss.extensions.api.ArgumentsMap;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperation;
import ro.sync.ecss.extensions.api.AuthorOperationException;

public class QueryDatabaseOperation implements AuthorOperation{
```

3. Now define the operation's arguments. For each of them you will use a `String` constant representing the argument name:

```
private static final String ARG_JDBC_DRIVER ="jdbc_driver";
private static final String ARG_USER ="user";
private static final String ARG_PASSWORD ="password";
private static final String ARG_SQL ="sql";
private static final String ARG_CONNECTION ="connection";
```

4. You must describe each of the argument name and type. To do this implement the `getArguments` method which will return an array of argument descriptors:

```
public ArgumentDescriptor[] getArguments() {
  ArgumentDescriptor args[] = new ArgumentDescriptor[] {
    new ArgumentDescriptor(
      ARG_JDBC_DRIVER,
      ArgumentDescriptor.TYPE_STRING,
      "The name of the Java class that is the JDBC driver."),
    new ArgumentDescriptor(
      ARG_CONNECTION,
      ArgumentDescriptor.TYPE_STRING,
      "The database URL connection string."),
    new ArgumentDescriptor(
      ARG_USER,
      ArgumentDescriptor.TYPE_STRING,
      "The name of the database user."),
    new ArgumentDescriptor(
      ARG_PASSWORD,
      ArgumentDescriptor.TYPE_STRING,
      "The database password."),
    new ArgumentDescriptor(
      ARG_SQL,
      ArgumentDescriptor.TYPE_STRING,
      "The SQL statement to be executed.")
  };
  return args;
 }
```

   These names, types and descriptions will be listed in the **Arguments** table when the operation is configured.

5. When the operation is invoked, the implementation of the `doOperation` method extracts the arguments, forwards them to the method that connects to the database and generates the XML fragment. The XML fragment is then inserted at the caret position.

```
public void doOperation(AuthorAccess authorAccess, ArgumentsMap map)
   throws IllegalArgumentException, AuthorOperationException {

  // Collects the arguments.
  String jdbcDriver =
   (String)map.getArgumentValue(ARG_JDBC_DRIVER);
  String connection =
   (String)map.getArgumentValue(ARG_CONNECTION);
  String user =
   (String)map.getArgumentValue(ARG_USER);
  String password =
   (String)map.getArgumentValue(ARG_PASSWORD);
  String sql =
   (String)map.getArgumentValue(ARG_SQL);

  int caretPosition = authorAccess.getCaretOffset();
  try {
   authorAccess.insertXMLFragment(
     getFragment(jdbcDriver, connection, user, password, sql),
     caretPosition);
  } catch (SQLException e) {
   throw new AuthorOperationException(
     "The operation failed due to the following database error: "
     + e.getMessage(), e);
  } catch (ClassNotFoundException e) {
   throw new AuthorOperationException(
     "The JDBC database driver was not found. Tried to load ' "
     + jdbcDriver + "'", e);
  }
 }
```

6. The `getFragment` method loads the JDBC driver, connects to the database and extracts the data. The result is a `table` element from the `http://www.oxygenxml.com/sample/documentation` namespace. The `header` element contains the names of the SQL columns. All the text from the XML fragment is escaped. This means that the '<' and '&' characters are replaced with the '&lt;' and '&amp;' character entities to ensure the fragment is well-formed.

```
private String getFragment(
  String jdbcDriver,
  String connectionURL,
  String user,
  String password,
  String sql) throws
   SQLException,
   ClassNotFoundException {

      Properties pr = new Properties();
      pr.put("characterEncoding", "UTF8");
      pr.put("useUnicode", "TRUE");
      pr.put("user", user);
      pr.put("password", password);

      // Loads the database driver.
      Class.forName(jdbcDriver);
      // Opens the connection
      Connection connection =
         DriverManager.getConnection(connectionURL, pr);
      java.sql.Statement statement =
         connection.createStatement();
      ResultSet resultSet =
         statement.executeQuery(sql);

      StringBuffer fragmentBuffer = new StringBuffer();
      fragmentBuffer.append(
        "<table xmlns=" +
        "'http://www.oxygenxml.com/sample/documentation'>");

      //
      // Creates the table header.
      //
      fragmentBuffer.append("<header>");
      ResultSetMetaData metaData = resultSet.getMetaData();
      int columnCount = metaData.getColumnCount();
      for (int i = 1; i <= columnCount; i++) {
          fragmentBuffer.append("<td>");
          fragmentBuffer.append(
            xmlEscape(metaData.getColumnName(i)));
          fragmentBuffer.append("</td>");
      }
      fragmentBuffer.append("</header>");

      //
      // Creates the table content.
      //
      while (resultSet.next()) {
          fragmentBuffer.append("<tr>");
          for (int i = 1; i <= columnCount; i++) {
              fragmentBuffer.append("<td>");
              fragmentBuffer.append(
                xmlEscape(resultSet.getObject(i)));
              fragmentBuffer.append("</td>");
          }
          fragmentBuffer.append("</tr>");
      }

      fragmentBuffer.append("</table>");

      // Cleanup
      resultSet.close();
      statement.close();
      connection.close();
      return fragmentBuffer.toString();
}
```

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

7. Package the compiled class into a jar file.
8. Copy the jar file and the JDBC driver files into the `frameworks / sdf` directory.
9. Add the jars to the Author class path. For this, Open the options Document Type Dialog, select **SDF** and press the **Edit** button. Select the **Classpath** tab in the lower part of the dialog.
10. Click on the **Actions** label. The action properties are:

    - Set **ID** to **clients_report**.
    - Set **Name** to **Clients Report**.
    - Set **Menu access key** to letter **r**.
    - Set **Description** to **Connects to the database and collects the list of clients**.
    - Set **Toolbar icon** to **${frameworks}/sdf/TableDB20.png** (image `TableDB20.png` is already stored in the `frameworks / sdf` folder).
    - Leave empty the **Menu icon**.
    - Set **shortcut key** to **Ctrl+Shift+C**.

11. The action will work only if the current element is a **section**. Set up the operation as follows:

    - Set **XPath expression** to:

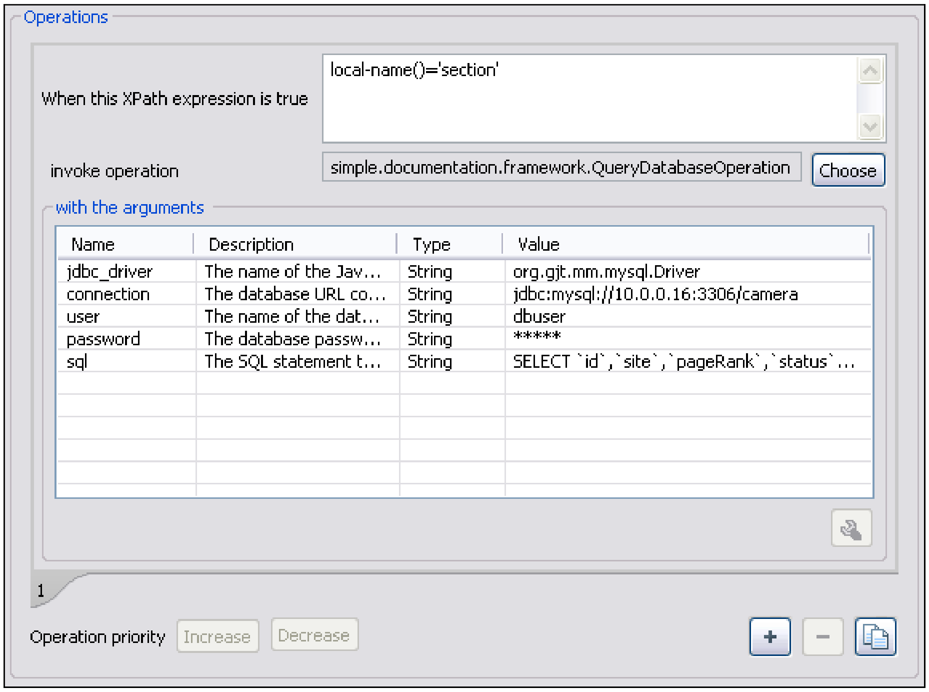
      ```
      local-name()='section'
      ```

    - Use the Java operation defined earlier to set the **Invoke operation** field. Press the **Choose** button, then select `simple.documentation.framework.QueryDatabaseOperation`. Once selected, the list of arguments is displayed. In the figure below the first argument, *jdbc_driver*, represents the class name of the MySQL JDBC driver. The connection string has the URL syntax : *jdbc://<database_host>:<database_port>/<database_name>*.

      The SQL expression used in the example follows, but it can be any valid SELECT expression which can be applied to the database:

      ```
      SELECT userID, email FROM users
      ```

12. Add the action to the toolbar, using the **Toolbar** panel.



When this XPath expression is true: local-name()='section'

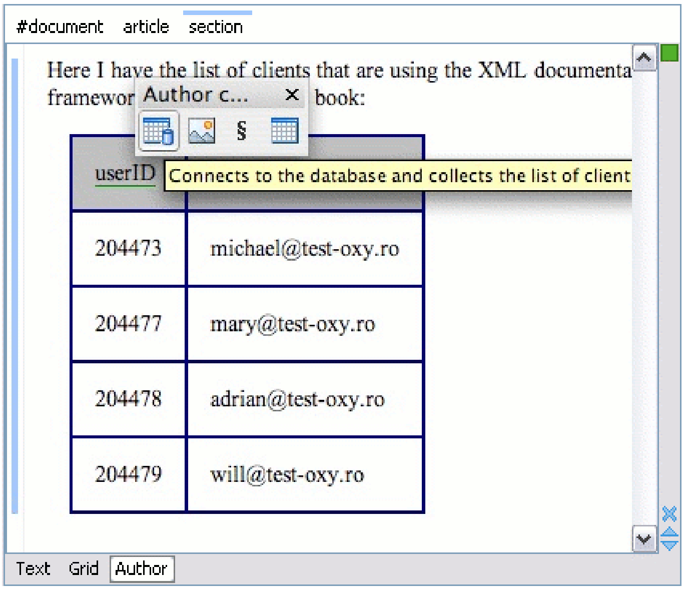
invoke operation: simple.documentation.framework.QueryDatabaseOperation

with the arguments

Name	Description	Type	Value
jdbc_driver	The name of the Jav…	String	org.gjt.mm.mysql.Driver
connection	The database URL co…	String	jdbc:mysql://10.0.0.16:3306/camera
user	The name of the dat…	String	dbuser
password	The database passw…	String	*****
sql	The SQL statement t…	String	SELECT `id`,`site`,`pageRank`,`status`…

Figure 174: Java Operation Arguments Setup

To test the action you can open the *sdf_sample.xml* sample place the caret inside a `section` between two para elements for instance. Press the **Create Report** button from the toolbar. You can see below the toolbar with the action button and sample table inserted by the **Clients Report** action.



Here I have the list of clients that are using the XML documenta framewor book:

Connects to the database and collects the list of client

userID	
204473	michael@test-oxy.ro
204477	mary@test-oxy.ro
204478	adrian@test-oxy.ro
204479	will@test-oxy.ro

Figure 175: Table Content Extracted from the Database

Creating the Basic Association

Let us go through an example of creating a document type and editing an XML document of this type. We will call our document type **Simple Documentation Framework**.

First Step - XML Schema

Our documentation framework will be very simple. The documents will be either `articles` or `books`, both composed of `sections`. The `sections` may contain `titles`, `paragraphs`, `figures`, `tables` and other `sections`. To complete the picture, each section will include a `def` element from another namespace.

The first schema file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.oxygenxml.com/sample/documentation"
    xmlns:doc="http://www.oxygenxml.com/sample/documentation"
    xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts"
    elementFormDefault="qualified">

    <xs:import namespace=
    "http://www.oxygenxml.com/sample/documentation/abstracts"
     schemaLocation=
    "abs.xsd"/>
```

The namespace of the documents will be `http://www.oxygenxml.com/sample/documentation`. The namespace of the `def` element is `http://www.oxygenxml.com/sample/documentation/abstracts`.

Now let's define the structure of the sections. They all start with a title, then have the optional `def` element then either a sequence of other sections, or a mixture of paragraphs, images and tables.

```
<xs:element name="book" type="doc:sectionType"/>
<xs:element name="article" type="doc:sectionType"/>
<xs:element name="section" type="doc:sectionType"/>

<xs:complexType name="sectionType">
    <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element ref="abs:def" minOccurs="0"/>
        <xs:choice>
            <xs:sequence>
                <xs:element ref="doc:section" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:choice maxOccurs="unbounded">
                <xs:element ref="doc:para"/>
                <xs:element ref="doc:image"/>
                <xs:element ref="doc:table"/>
            </xs:choice>
        </xs:choice>
    </xs:sequence>
</xs:complexType>
```

The paragraph contains text and other styling markup, such as bold (`b`) and italic (`i`) elements.

```
<xs:element name="para" type="doc:paragraphType"/>

<xs:complexType name="paragraphType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="b"/>
        <xs:element name="i"/>
    </xs:choice>
</xs:complexType>
```

The `image` element has an attribute with a reference to the file containing image data.

```
<xs:element name="image">
    <xs:complexType>
        <xs:attribute name="href" type="xs:anyURI" use="required"/>
    </xs:complexType>
</xs:element>
```

The `table` contains a header row and then a sequence of rows (`tr` elements) each of them containing the cells. Each cell has the same content as the paragraphs.

```
 <xs:element name="table">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="header">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td" maxOccurs="unbounded"
                            type="doc:paragraphType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="tr" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td" type="doc:tdType"
                             maxOccurs="unbounded"/>

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


<xs:complexType name="tdType">
    <xs:complexContent>
        <xs:extension base="doc:paragraphType">
            <xs:attribute name="row_span" type="xs:integer"/>
            <xs:attribute name="column_span" type="xs:integer"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

The `def` element is defined as a text only element in the imported schema `abs.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=
     "http://www.oxygenxml.com/sample/documentation/abstracts">
    <xs:element name="def" type="xs:string"/>
</xs:schema>
```

Now the XML data structure will be styled.

Schema Settings

In *the dialog for editing the document type properties*, in the bottom section there are a series of tabs. The first one refers to the schema that is used for validation of the documents that match the defined **Association Rules**.

Important:

If the document refers a schema, using for instance a `DOCTYPE` declaration or a `xsi:schemaLocation` attribute, the schema from the document type association will not be used when validating.

Schema Type Select from the combo box the value **XML Schema**.

Schema URI Enter the value `${frameworks}/sdf/schema/sdf.xsd`. We should use the `${frameworks}` editor variable in the schema URI path instead of a full path in order to be valid for different Oxygen XML Editor plugin installations.

> **Important:**
>
> The `${frameworks}` variable is expanded at the validation time into the absolute location of the directory containing the frameworks.

Second Step - The CSS

If you read the *Simple Customization Tutorial* then you already have some basic notions about creating simple styles. The example document contains elements from different namespaces, so you will use CSS Level 3 extensions supported by the Author layout engine to associate specific properties with that element.

Defining the General Layout

Now the basic layout of the rendered documents is created.

Elements that are stacked one on top of the other are: `book`, `article`, `section`, `title`, `figure`, `table`, `image`. These elements are marked as having `block` style for display. Elements that are placed one after the other in a flowing sequence are: `b`, `i`. These will have `inline` display.

```
/* Vertical flow */
book,
section,
para,
title,
image,
ref {
    display:block;
}

/* Horizontal flow */
b,i {
    display:inline;
}
```

Important:

Having `block` display children in an `inline` display parent, makes Oxygen XML Editor plugin Author change the style of the parent to `block` display.

Styling the `section` Element

The title of any section must be bold and smaller than the title of the parent section. To create this effect a sequence of CSS rules must be created. The `*` operator matches any element, it can be used to match titles having progressive depths in the document.

```
title{
    font-size: 2.4em;
    font-weight:bold;
}
* * title{
    font-size: 2.0em;
}
* * * title{
    font-size: 1.6em;
}
* * * * title{
    font-size: 1.2em;
}
```

It's useful to have before the title a constant text, indicating that it refers to a section. This text can include also the current section number. The `:before` and `:after` pseudo elements will be used, plus the CSS counters.

First declare a counter named `sect` for each `book` or `article`. The counter is set to zero at the beginning of each such element:

```
book,
article{
    counter-reset:sect;
}
```

The `sect` counter is incremented with each `section`, that is a direct child of a `book` or an `article` element.

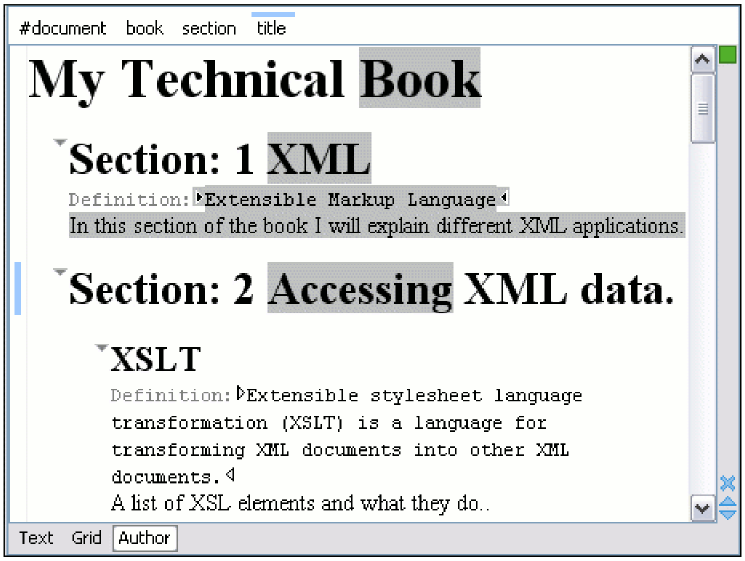
```
book > section,
article > section{
    counter-increment:sect;
}
```

The "static" text that will prefix the section title is composed of the constant "Section ", followed by the decimal value of the `sect` counter and a dot.

```
book > section > title:before,
article > section > title:before{
    content: "Section " counter(sect) ". ";
}
```

To make the documents easy to read, you add a margin to the sections. In this way the higher nesting level, the larger the left side indent. The margin is expressed relatively to the parent bounds:

```
section{
    margin-left:1em;
    margin-top:1em;
}
```



My Technical Book

Section: 1 XML

Definition: Extensible Markup Language

In this section of the book I will explain different XML applications.

Section: 2 Accessing XML data.

XSLT

Definition: Extensible stylesheet language transformation (XSLT) is a language for transforming XML documents into other XML documents.

A list of XSL elements and what they do..

Figure 176: A sample of nested sections and their titles.

In the above screenshot you can see a sample XML document rendered by the CSS stylesheet. The selection "avoids" the text that is generated by the CSS "content" property. This happens because the CSS generated text is not present in the XML document and is just a visual aid.

Styling the Inline Elements

The "bold" style is obtained by using the `font-weight` CSS property with the value `bold`, while the "italic" style is specified by the `font-style` property:

```
b {
    font-weight:bold;
}

i {
    font-style:italic;
}
```

Styling Images

The CSS 2.1 does not specify how an element can be rendered as an image. To overpass this limitation, Oxygen XML Editor plugin Author supports a CSS Level 3 extension allowing to load image data from an URL. The URL of the image must be specified by one of the element attributes and it is resolved through the catalogs specified in Oxygen XML Editor plugin .

```
image{
    display:block;
    content: attr(href, url);
    margin-left:2em;
}
```

Our `image` element has the required attribute `href` of type `xs:anyURI`. The `href` attribute contains an image location so the rendered content is obtained by using the function:

```
attr(href, url)
```

Important:

The first argument is the name of the attribute pointing to the image file. The second argument of the `attr` function specifies the type of the content. If the type has the `url` value, then Oxygen XML Editor plugin identifies the content as being an image. If the type is missing, then the content will be the text representing the attribute value.

Important:

Oxygen XML Editor plugin Author handles both absolute and relative specified URLs. If the image has an *absolute* URL location (e.g: "http://www.oasis-open.org/images/standards/oasis_standard.jpg") then it is loaded directly from this location. If the image URL is *relative* specified to the XML document (e.g: "images/my_screenshot.jpg") then the location is obtained by adding this value to the location of the edited XML document.

An image can also be referenced by the name of a DTD entity which specifies the location of the image file. For example if the document declares an entity **graphic** which points to a JPEG image file:

```
<!ENTITY graphic SYSTEM "depo/keyboard_shortcut.jpg" NDATA JPEG>
```

and the image is referenced in the XML document by specifying the name of the entity as the value of an attribute:

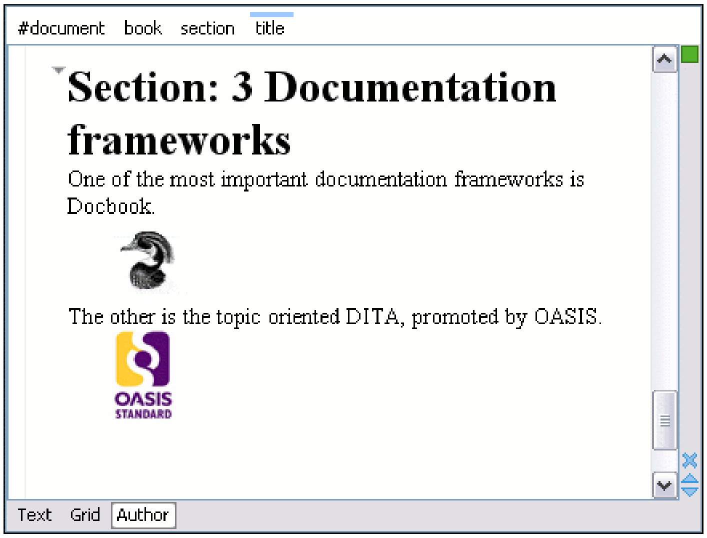
```
<mediaobject>
    <imageobject>
        <imagedata entityref="graphic" scale="50"/>
    </imageobject>
</mediaobject>
```

The CSS should use the functions `url`, `attr` and `unparsed-entity-uri` for displaying the image in the Author mode:

```
imagedata[entityref]{
    content: url(unparsed-entity-uri(attr(entityref)));
}
```

To take into account the value of the `width` attribute of the `imagedata` and use it for resizing the image, the CSS can define the following rule:

```
imagedata[width]{
  width:attr(width, length);
}
```



Section: 3 Documentation frameworks

One of the most important documentation frameworks is Docbook.

The other is the topic oriented DITA, promoted by OASIS.

Figure 177: Samples of images in Author

Testing the Document Type Association

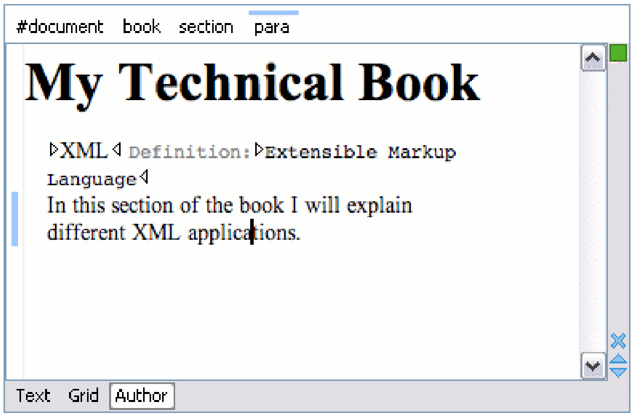
To test the new Document Type create an XML instance that is conforming with the *Simple Documentation Framework* association rules. You will not specify an XML Schema location directly in the document, using an `xsi:schemaLocation` attribute; Oxygen XML Editor plugin will detect instead its associated document type and use the specified schema.

```
<book xmlns="http://www.oxygenxml.com/sample/documentation"
      xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">

    <title>My Technical Book</title>
    <section>
        <title>XML</title>
        <abs:def>Extensible Markup Language</abs:def>
        <para>In this section of the book I will
           explain different XML applications.</para>
    </section>
</book>
```

When trying to validate the document there should be no errors. Now modify the `title` to `title2`. Validate again. This time there should be one error:

```
  cvc-complex-type.2.4.a: Invalid content was found starting with element
  'title2'. One of '{"http://www.oxygenxml.com/sample/documentation":title}'

  is expected.
```

Undo the tag name change. Press on the **Author** button at the bottom of the editing area. Oxygen XML Editor plugin should load the CSS from the document type association and create a layout similar to this:



My Technical Book

XML Definition: Extensible Markup Language

In this section of the book I will explain different XML applications.

Organizing the Framework Files

First create a new folder called `sdf` (from "Simple Documentation Framework") in `{oXygen_installation_directory}/frameworks`. This folder will be used to store all files related to the documentation framework. The following folder structure will be created:

```
oxygen
  frameworks
     sdf
       schema
       css
```

Important:

The `frameworks` directory is the container where all the oXygen framework customizations are located.

Each subdirectory contains files related to a specific type of XML documents: schemas, catalogs, stylesheets, CSSs, etc.

Distributing a framework means delivering a framework directory.

Important:

It is assumed that you have the right to create files and folder inside the oXygen installation directory. If you do not have this right, you will have to install another copy of the program in a folder you have access to, the home directory for instance, or your desktop. You can download the "all platforms" distribution from the oXygen website and extract it in the chosen folder.

To test your framework distribution you will need to copy it in the `frameworks` directory of the newly installed application and start oXygen by running the provided start-up script files.

You should copy the created schema files `abs.xsd` and `sdf.xsd`, `sdf.xsd` being the master schema, to the `schema` directory and the CSS file `sdf.css` to the `css` directory.

Packaging and Deploying

Using a file explorer, go to the Oxygen XML Editor plugin `frameworks` directory. Select the `sdf` directory and make an archive from it. Move it to another Oxygen XML Editor plugin installation (eventually on another computer). Extract it in the `frameworks` directory. Start Oxygen XML Editor plugin and test the association as explained above.

If you create multiple document type associations and you have a complex directory structure it might be easy from the deployment point of view to use an Oxygen XML Editor plugin All Platforms distribution. Add your framework files to it, repackage it and send it to the content authors.

Attention:

When deploying your customized sdf directory please make sure that your sdf directory contains the `sdf.framework` file (that is the file defined as External Storage in Document Type Association dialog shall always be stored inside the sdf directory). If your external storage points somewhere else Oxygen XML Editor plugin will not be able to update the Document Type Association options automatically on the deployed computers.

Configuring New File Templates

You will create a set of document templates that the content authors will use as starting points for creating new *Simple Document Framework* books and articles.

Each of the Document Type Associations can point to a directory usually named `templates` containing the file templates. All files found here are considered templates for the respective document type. The template name is taken from the file name, and the template type is detected from the file extension.

1. Create the templates directory into the `frameworks / sdf` directory. The directory tree for the documentation framework is now:

```
oxygen
    frameworks
        sdf
            schema
            css
            templates
```

2. Now let's create in this `templates` directory two files, one for the *book* template and another for the *article* template.

   The `Book.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://www.oxygenxml.com/sample/documentation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">
    <title>Book Template Title</title>
    <section>
        <title>Section Title</title>
        <abs:def/>
        <para>This content is copyrighted:</para>
        <table>
            <header>
                <td>Company</td>
                <td>Date</td>
            </header>
            <tr>
                <td/>
                <td/>
            </tr>
        </table>
    </section>
    </book>
```

   The `Article.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<article
    xmlns="http://www.oxygenxml.com/sample/documentation"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <title></title>
    <section>
        <title></title>
        <para></para>
        <para></para>
    </section>
</article>
```

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

3. Open the Document Type dialog for the **SDF** framework and click on the **Templates** tab. Enter in the **Templates directory** text field the value `${frameworksDir} / sdf / templates`. As you already seen before, it is recommended that all the file references made from a Document Type Association to be relative to the `${frameworksDir}` directory. Binding a Document Type Association to an absolute file (e. g.: "C:\some_dir\templates") makes the association difficult to share between users.
4. To test the templates settings, press the **File**/**New** menu item to display the **New** dialog. The names of the two templates are prefixed with the name of the Document Type Association, in our case **SDF**. Selecting one of them should create a new XML file with the content specified in the template file.

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 Editor plugin commands of external engines or other external tools, in transformation scenarios, validation scenarios and Author operations:

- **${oxygenHome}** - Oxygen XML Editor plugin installation folder as URL.
- **${oxygenInstallDir}** - Oxygen XML Editor plugin installation folder as file path.
- **${frameworks}** - The path (as URL) of the `frameworks` subfolder of the Oxygen XML Editor plugin install folder.
- **${frameworksDir}** - The path (as file path) of the `frameworks` subfolder of the Oxygen XML Editor plugin installation folder.
- **${home}** - The path (as URL) of the user home folder.
- **${homeDir}** - The path (as file path) of the user home folder.
- **${pdu}** - Current project folder as URL.
- **${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.
- **${caret}** - The position where the caret is inserted. This variable can be used in a *code template* , in Author operations, or in a selection plugin.
- **${selection}** - The XML content of the current selection in the editor panel. This variable can be used in a *code template* and Author operations,
- **${id}** - Application-level unique identifier.
- **${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 Editor plugin checks that the URL is valid before passing it to the transformation.
    - `password` - input characters are hidden.
    - `generic` - the input is treated as generic text that requires no special handling.

  - `'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 *configured in Preferences.*

Configuring XML Catalogs

In the XML sample file for **SDF** you did not use a xsi:schemaLocation attribute, but instead you let the editor use the schema from the association. However there are cases in which you must refer for instance the location of a schema file from a remote web location and an Internet connection may not be available. In such cases an XML catalog may be used to map the web location to a local file system entry. The following procedure presents an example of using an XML catalogs, by modifying our `sdf.xsd` XML Schema file from the *Example Files Listings*.

1. Create a catalog file that will help the parser locate the schema for validating the XML document. The file must map the location of the schema to a local version of the schema.

   Create a new XML file called `catalog.xml` and save it into the `{oXygen_installation_directory} / frameworks / sdf` directory. The content of the file should be:

   ```
   <?xml version="1.0"?>
   <catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
       <system systemId="http://www.oxygenxml.com/SDF/abs.xsd"
               uri="schema/abs.xsd"/>
       <uri name="http://www.oxygenxml.com/SDF/abs.xsd"
                  uri="schema/abs.xsd"/>
   </catalog>
   ```

2. Add catalog files to your Document Type Association using the Catalogs tab from the Document Type dialog.

To test the catalog settings, restart Oxygen XML Editor plugin and try to validate a new sample **Simple Documentation Framework** document. There should be no errors.

The `sdf.xsd` schema that validates the document refers the other file `abs.xsd` through an import element:

```
<xs:import namespace=
 "http://www.oxygenxml.com/sample/documentation/abstracts"
 schemaLocation="http://www.oxygenxml.com/SDF/abs.xsd"/>
```

The `schemaLocation` attribute references the `abs.xsd` file:

```
xsi:schemaLocation="http://www.oxygenxml.com/sample/documentation/abstracts

    http://www.oxygenxml.com/SDF/abs.xsd"/>
```

The catalog mapping is:

```
http://www.oxygenxml.com/SDF/abs.xsd -> schema/abs.xsd
```

This means that all the references to http://www.oxygenxml.com/SDF/abs.xsd must be resolved to the `abs.xsd` file located in the `schema` directory. The URI element is used by URI resolvers, for example for resolving a URI reference used in an XSLT stylesheet.

Configuring Transformation Scenarios

When distributing a framework to the users, it is a good idea to have the transformation scenarios already configured. This would help the content authors publish their work in different formats. Being contained in the **Document Type Association** the scenarios can be distributed along with the actions, menus, toolbars, catalogs, etc.

These are the steps that allow you to create a transformation scenario for your framework.

1. Create a `xsl` folder inside the `frameworks / sdf` folder.

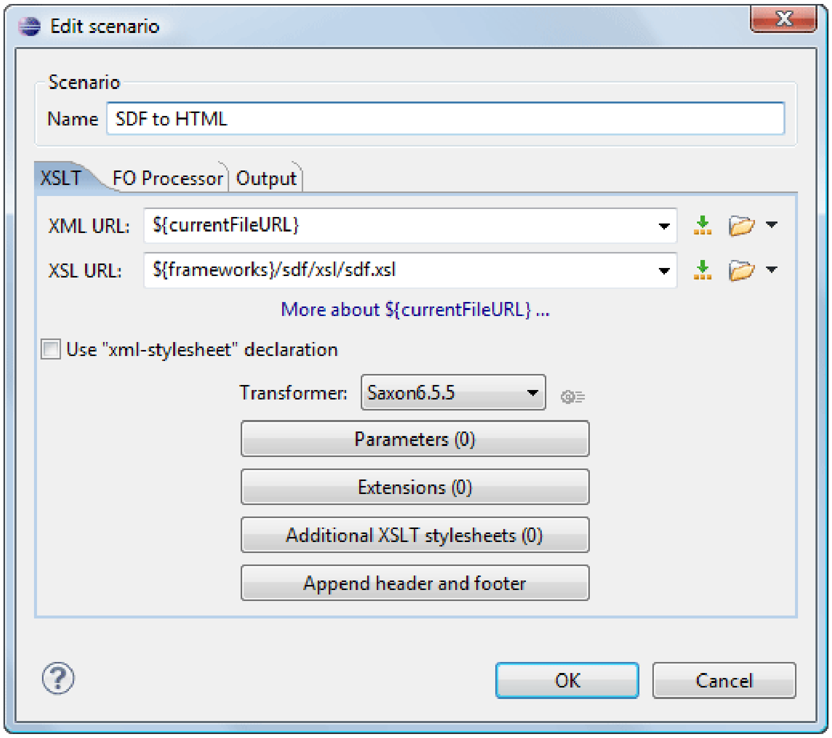
   The folder structure for the documentation framework should be:

```
oxygen
  frameworks
     sdf
       schema
       css
       templates
       xsl
```

2. Create the `sdf.xsl` file in the `xsl` folder. The complete content of the `sdf.xsl` file is found in the *Example Files Listings*.
3. Open the **Options**/**Preferences**/**Document Type Associations**. Open the **Document Type** dialog for the **SDF** framework then choose the **Transformation** tab. Click the **New** button.

   In the **Edit Scenario** dialog, fill the following fields:

   - Fill in the **Name** field with *SDF to HTML*. This will be the name of your transformation scenario.
   - Set the **XSL URL** field to `${frameworks}/sdf/xsl/sdf.xsl`.
   - Set the **Transformer** to *Saxon 9B*.


Edit scenario

Scenario

Name: SDF to HTML

XSLT | FO Processor | Output

XML URL: ${currentFileURL}

XSL URL: ${frameworks}/sdf/xsl/sdf.xsl

More about ${currentFileURL} ...

☐ Use "xml-stylesheet" declaration

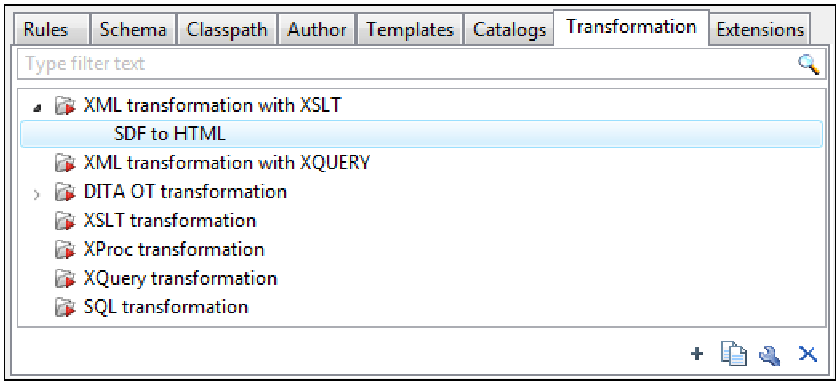
Transformer: Saxon6.5.5

Figure 178: Configuring a transformation scenario

4. Change to the **Output** tab. Configure the fields as follows:
   - Set the **Save as** field to `${cfd}/${cfn}.html`. This means the transformation output file will have the name of the XML file and the *html* extension and will be stored in the same folder.
   - Enable the **Open in Browser/System Application** option.

     Note: If you already set the **Default Internet browser** option in the **Global** preferences page, it will take precedence over the default system application settings.
   - Enable the **Saved file** option.

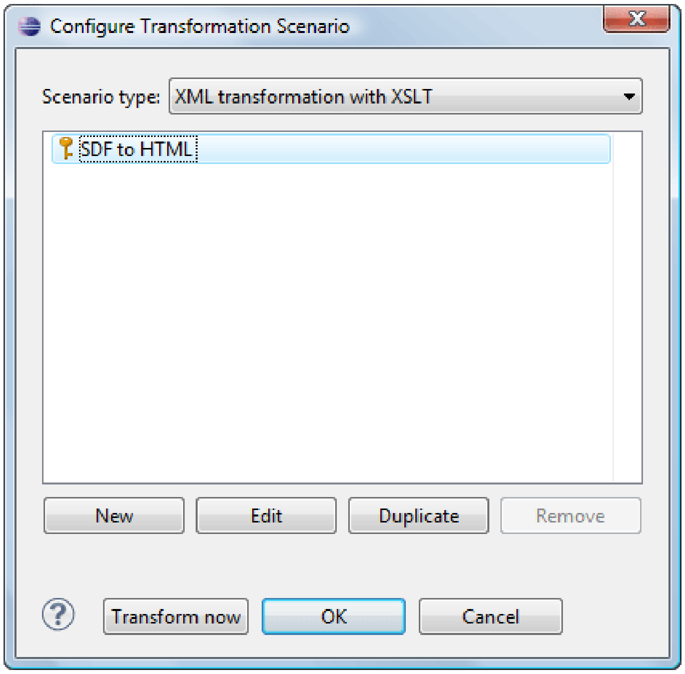
Now the scenario is listed in the **Transformation** tab:



- XML transformation with XSLT
  - SDF to HTML
- XML transformation with XQUERY
- DITA OT transformation
- XSLT transformation
- XProc transformation
- XQuery transformation
- SQL transformation

Figure 179: The transformation tab

To test the transformation scenario you just created, open the **SDF** XML sample from the *Example Files Listings*. Click on the ▶ **Apply Transformation Scenario** button to display the **Configure Transformation Dialog**. Its scenario list contains the scenario you defined earlier *SDF to HTML*. Click it then choose **Transform now**. The HTML file should be saved in the same folder as the XML file and displayed in the browser.



Scenario type: XML transformation with XSLT

SDF to HTML

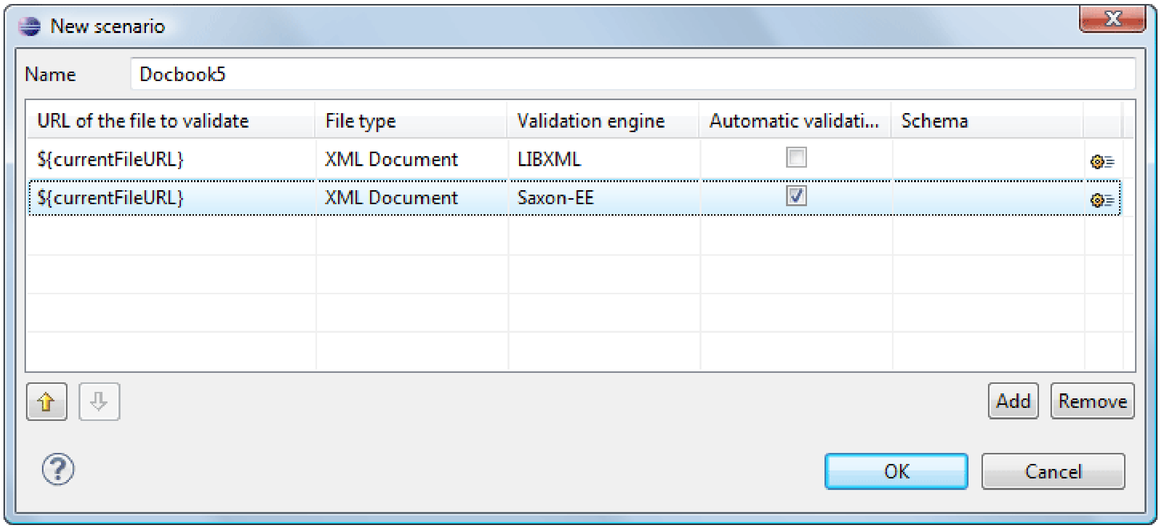
Figure 180: Selecting the predefined scenario

Configuring Validation Scenarios

You can distribute a framework with a series of already configured validation scenarios. Also, this provides enhanced validation support allowing you to use multiple grammars to check the document. For example, you can use Schematron rules to impose guidelines, otherwise impossible to enforce using conventional validation.

To associate a validation scenario with a specific framework, follow these steps:

1. Open the **Options**/**Preferences**/**Document Type Associations**. Open the **Document Type** dialog for the **SDF** framework, then choose the **Validation** tab. This tab holds a list of document types for which you can define validation scenarios. To set one of the validation scenarios as default for a specific document type, select it and press / **Toggle default**.
2. Press the **New** button to add a new scenario.
3. Press the **Add** button to add a new validation unit with default settings.
   The dialog that lists all validation units of the scenario is opened.



Name: Docbook5

URL of the file to validate	File type	Validation engine	Automatic validati...	Schema
${currentFileURL}	XML Document	LIBXML	☐	
${currentFileURL}	XML Document	Saxon-EE	☑	

Figure 181: Add / Edit a Validation Unit

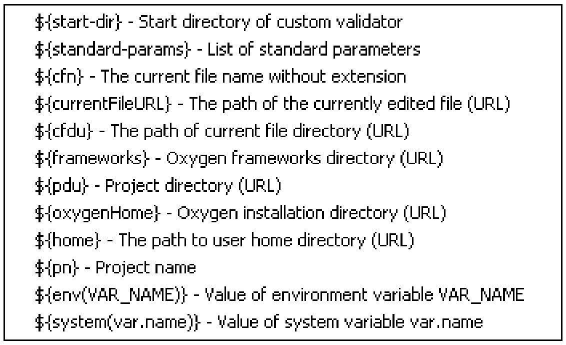
The table holds the following information:

- **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 Editor plugin automatically selects the file type depending on the value of the **URL of the file to validate** field.
- **Validation engine** - One of the engines available in Oxygen XML Editor plugin for validation of the type of document to which the current module belongs. **Default engine** is the default setting and means that the validation is done by the default engine set in Preferences pages for the type of the current document (XML document, XML Schema, XSLT stylesheet, XQuery file, etc) instead of a validation scenario.
- **Automatic validation** - If this option is checked, then the validation operation defined by this row of the table is applied also by *the automatic validation feature.* If the **Automatic validation** feature is *disabled in Preferences* then this option does not take effect as the Preference setting has higher priority.
- **Schema** - Active when you set the **File type** to **XML Document**.
- **Settings** - Contains an action that allows you to set a schema, when validating XML documents, or a list of extensions when validating XSL or XQuery documents.

4. Edit the URL of the main validation module.

Specify the URL of the main module:

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



${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 182: Insert an Editor Variable

5. Select the type of the validated document.

Note that it determines the list of possible validation engines.

6. Select the validation engine.

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

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

Configuring Extensions

You can add extensions to your Document Type Association using the **Extensions** tab from the Document Type dialog.

Configuring an Extensions Bundle

Starting with Oxygen XML Editor plugin 10.3 version a single bundle was introduced acting as a provider for all other extensions. The individual extensions can still be set and if present they take precedence over the single provider, but this practice is being discouraged and the single provider should be used instead.

The extensions bundle is represented by the *`ro.sync.ecss.extensions.api.ExtensionsBundle`* class. The provided implementation of the `ExtensionsBundle` is instantiated when the rules of the Document Type

Association defined for the custom framework match a document opened in the editor. Therefor references to objects which need to be persistent throughout the application running session must not be kept in the bundle because the next detection event can result in creating another `ExtensionsBundle` instance.

☞ **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Editor plugin website*. Also it can be downloaded as a *zip archive from the website*.

1. Create a new Java project, in your IDE. Create the `lib` folder in the Java project folder and copy in it the `oxygen.jar` file from the `{oXygen_installation_directory}/lib` folder.
2. Create the class `simple.documentation.framework.SDFExtensionsBundle` which must extend the abstract class `ro.sync.ecss.extensions.api.ExtensionsBundle`.

   ```
   public class SDFExtensionsBundle extends ExtensionsBundle {
   ```

3. A **Document Type ID** and a short description should be defined first by implementing the methods `getDocumentTypeID` and `getDescription`. The Document Type ID is used to uniquely identify the current framework. Such an ID must be provided especially if options related to the framework need to be persistently stored and retrieved between sessions.

   ```
   public String getDocumentTypeID() {
        return "Simple.Document.Framework.document.type";
   }

   public String getDescription() {
        return "A custom extensions bundle used for the Simple Document" +
                      "Framework document type";
   }
   ```

4. In order to be notified about the activation of the custom Author extension in relation with an opened document an *`ro.sync.ecss.extensions.api.AuthorExtensionStateListener`* should be implemented. The **activation** and **deactivation** events received by this listener should be used to perform custom initializations and to register / remove listeners like *`ro.sync.ecss.extensions.api.AuthorListener`*, *`ro.sync.ecss.extensions.api.AuthorMouseListener`* or *`ro.sync.ecss.extensions.api.AuthorCaretListener`*. The custom author extension state listener should be provided by implementing the method `createAuthorExtensionStateListener`.

   ```
   public AuthorExtensionStateListener createAuthorExtensionStateListener() {
        return new SDFAuthorExtensionStateListener();
   }
   ```

   The `AuthorExtensionStateListener` is instantiated and notified about the activation of the framework when the rules of the Document Type Association match a document opened in the Author editor mode. The listener is notified about the deactivation when another framework is activated for the same document, the user switches to another mode or the editor is closed. A complete description and implementation of an `ro.sync.ecss.extensions.api.AuthorExtensionStateListener` can be found in the *Implementing an Author Extension State Listener*.

   If Schema Aware mode is active in Oxygen, all actions that can generate invalid content will be redirected toward the *`ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler`*. The handler can either resolve a specific case, let the default implementation take place or reject the edit entirely by throwing an *`ro.sync.ecss.extensions.api.InvalidEditException`*. The actions that are forwarded to this handler include typing, delete or paste.

   See the *Implementing an Author Schema Aware Editing Handler* section for more details about this handler.

5. Customizations of the content completion proposals are permitted by creating a schema manager filter extension. The interface that declares the methods used for content completion proposals filtering is *`ro.sync.contentcompletion.xml.SchemaManagerFilter`*. The filter can be applied on elements,

attributes or on their values. Responsible for creating the content completion filter is the method `createSchemaManagerFilter`. A new `SchemaManagerFilter` will be created each time a document matches the rules defined by the Document Type Association which contains the filter declaration.

```
public SchemaManagerFilter createSchemaManagerFilter() {
     return new SDFSchemaManagerFilter();
}
```

A detailed presentation of the schema manager filter can be found in *Configuring a Content completion handler* section.

6. The Author supports link based navigation between documents and document sections. Therefore, if the document contains elements defined as links to other elements, for example links based on the **id** attributes, the extension should provide the means to find the referred content. To do this an implementation of the *`ro.sync.ecss.extensions.api.link.ElementLocatorProvider`* interface should be returned by the `createElementLocatorProvider` method. Each time an element pointed by a link needs to be located the method is invoked.

```
public ElementLocatorProvider createElementLocatorProvider() {
     return new DefaultElementLocatorProvider();
}
```

The section that explains how to implement an element locator provider is *Configuring a Link target element finder*.

7. The drag and drop functionality can be extended by implementing the *`ro.sync.exml.editor.xmleditor.pageauthor.AuthorDnDListener`* interface. Relevant methods from the listener are invoked when the mouse is dragged, moved over, or exits the Author editor mode, when the drop action changes, and when the drop occurs. Each method receives the `DropTargetEvent` containing information about the drag and drop operation. The drag and drop extensions are available on Author mode for both Oxygen XML Editor plugin Eclipse plugin and standalone application. The Text mode corresponding listener is available only for Oxygen XML Editor plugin Eclipse plugin. The methods corresponding to each implementation are: `createAuthorAWTDndListener`, `createTextSWTDndListener` and `createAuthorSWTDndListener`.

```
public AuthorDnDListener createAuthorAWTDndListener() {
     return new SDFAuthorDndListener();
}
```

For more details about the Author drag and drop listeners see the *Configuring a custom Drag and Drop listener* section.

8. Another extension which can be included in the bundle is the reference resolver. In our case the references are represented by the **ref** element and the attribute indicating the referred resource is **location**. To be able to obtain the content of the referred resources you will have to implement a Java extension class which implements the *`ro.sync.ecss.extensions.api.AuthorReferenceResolver`*. The method responsible for creating the custom references resolver is `createAuthorReferenceResolver`. The method is called each time a document opened in an Author editor mode matches the Document Type Association where the extensions bundle is defined. The instantiated references resolver object is kept and used until another extensions bundle corresponding to another Document Type is activated as result of the detection process.

```
public AuthorReferenceResolver createAuthorReferenceResolver() {
     return new ReferencesResolver();
}
```

A more detailed description of the references resolver can be found in the *Configuring a References Resolver* section.

9. To be able to dynamically customize the default CSS styles for a certain *`ro.sync.ecss.extensions.api.node.AuthorNode`* an implementation of the

`ro.sync.ecss.extensions.api.StylesFilter` can be provided. The extensions bundle method responsible for creating the `StylesFilter` is `createAuthorStylesFilter`. The method is called each time a document opened in an Author editor mode matches the document type association where the extensions bundle is defined. The instantiated filter object is kept and used until another extensions bundle corresponding to another Document Type is activated as a result of the detection process.

```
public StylesFilter createAuthorStylesFilter() {
     return new SDFStylesFilter();
}
```

See the *Configuring CSS styles filter* section for more details about the styles filter extension.

10. In order to edit data in custom tabular format implementations of the `ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider` and `the` `ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider` interfaces should be provided. The two methods from the `ExtensionsBundle` specifying these two extension points are `createAuthorTableCellSpanProvider` and `createAuthorTableColumnWidthProvider`.

    ```
    public AuthorTableCellSpanProvider createAuthorTableCellSpanProvider() {
         return new TableCellSpanProvider();
    }

    public AuthorTableColumnWidthProvider
              createAuthorTableColumnWidthProvider() {
         return new TableColumnWidthProvider();
    }
    ```

    The two table information providers are not reused for different tables. The methods are called for each table in the document so new instances should be provided every time. Read more about the cell span and column width information providers in *Configuring a Table Cell Span Provider* and *Configuring a Table Column Width Provider* sections.

    If the functionality related to one of the previous extension point does not need to be modified then the developed `ro.sync.ecss.extensions.api.ExtensionsBundle` should not override the corresponding method and leave the default base implementation to return **null**.

11. Package the compiled class into a jar file.
12. Copy the jar file into the `frameworks / sdf` directory.
13. Add the jar file to the Author class path.
14. Register the Java class by clicking on the **Extensions** tab. Press the **Choose** button and select from the displayed dialog the name of the class: `SDFExtensionsBundle`.

    Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

Customize Profiling Conditions

For each document type, you can configure the phrase-type elements that wrap the profiled content by setting a custom `ro.sync.ecss.extensions.api.ProfilingConditionalTextProvider`. This configuration is set by default for DITA and Docbook frameworks.

Preserve Style and Format on Copy and Paste from External Applications

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

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

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

For other document types the default behavior of the paste operation is to keep only the text content without the styling but it can be customized by setting an XSLT stylesheet in that document type. The XSLT stylesheet must accept as input an XHTML flavor of the copied content and transform it to the equivalent XML markup that is appropriate for the target document type of the paste operation. The stylesheet is *set up* by implementing the `getImporterStylesheetFileName` method of an instance object of *the `AuthorExternalObjectInsertionHandler` class* which is returned by the `createExternalObjectInsertionHandler` method of *the `ExtensionsBundle` instance* of the target document type.

Implementing an Author Extension State Listener

The *`ro.sync.ecss.extensions.api.AuthorExtensionStateListener`* implementation is notified when the Author extension where the listener is defined is activated or deactivated in the Document Type detection process.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Editor plugin website*. Also it can be downloaded as a *zip archive from the website*.

```
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorExtensionStateListener;

public class SDFAuthorExtensionStateListener implements
  AuthorExtensionStateListener {
  private AuthorListener sdfAuthorDocumentListener;
  private AuthorMouseListener sdfMouseListener;
  private AuthorCaretListener sdfCaretListener;
  private OptionListener sdfOptionListener;
```

The **activation** event received by this listener when the rules of the Document Type Association match a document opened in the Author editor mode, should be used to perform custom initializations and to register listeners like *`ro.sync.ecss.extensions.api.AuthorListener`*, *`ro.sync.ecss.extensions.api.AuthorMouseListener`* or *`ro.sync.ecss.extensions.api.AuthorCaretListener`*.

```
 public void activated(AuthorAccess authorAccess) {
   // Get the value of the option.
   String option = authorAccess.getOptionsStorage().getOption(
               "sdf.custom.option.key", "");
   // Use the option for some initializations...

   // Add an option listener.
   authorAccess.getOptionsStorage().addOptionListener(sdfOptionListener);

   // Add author document listeners.
   sdfAuthorDocumentListener = new SDFAuthorListener();
   authorAccess.getDocumentController().addAuthorListener(
               sdfAuthorDocumentListener);

   // Add mouse listener.
   sdfMouseListener = new SDFAuthorMouseListener();
   authorAccess.getEditorAccess().addAuthorMouseListener(sdfMouseListener);

   // Add caret listener.
   sdfCaretListener = new SDFAuthorCaretListener();
   authorAccess.getEditorAccess().addAuthorCaretListener(sdfCaretListener);
```

```
    // Other custom initializations...

  }
```

The authorAccess parameter received by the `activated` method can be used to gain access to Author specific actions and informations related to components like the editor, document, workspace, tables, or the change tracking manager.

If options specific to the custom developed Author extension need to be stored or retrieved, a reference to the *`ro.sync.ecss.extensions.api.OptionsStorage`* can be obtained by calling the `getOptionsStorage` method from the author access. The same object can be used to register *`ro.sync.ecss.extensions.api.OptionListener`* listeners. An option listener is registered in relation with an option **key** and will be notified about the value changes of that option.

An `AuthorListener` can be used if events related to the Author document modifications are of interest. The listener can be added to the *`ro.sync.ecss.extensions.api.AuthorDocumentController`*. A reference to the document controller is returned by the `getDocumentController` method from the author access. The document controller can also be used to perform operations involving document modifications.

To provide access to Author editor component related functionality and information, the author access has a reference to the *`ro.sync.ecss.extensions.api.access.AuthorEditorAccess`* that can be obtained when calling the `getEditorAccess` method. At this level `AuthorMouseListener` and `AuthorCaretListener` can be added which will be notified about mouse and caret events occurring in the Author editor mode.

The **deactivation** event is received when another framework is activated for the same document, the user switches to another editor mode or the editor is closed. The `deactivate` method is typically used to unregister the listeners previously added on the `activate` method and to perform other actions. For example, options related to the deactivated author extension can be saved at this point.

```
  public void deactivated(AuthorAccess authorAccess) {
    // Store the option.
    authorAccess.getOptionsStorage().setOption(
                "sdf.custom.option.key", optionValue);

    // Remove the option listener.
    authorAccess.getOptionsStorage().removeOptionListener(sdfOptionListener);

    // Remove document listeners.
    authorAccess.getDocumentController().removeAuthorListener(
                sdfAuthorDocumentListener);

    // Remove mouse listener.
   authorAccess.getEditorAccess().removeAuthorMouseListener(sdfMouseListener);


    // Remove caret listener.
   authorAccess.getEditorAccess().removeAuthorCaretListener(sdfCaretListener);


    // Other actions...

  }
```

Implementing an Author Schema Aware Editing Handler

You can implement your own handler for actions like typing, delete or paste by providing an implementation of *`ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler`*. The *Schema Aware Editing* must be **On** or **Custom** in order for this handler to be called. The handler can either resolve a specific case, let the default implementation take place or reject the edit entirely by throwing an `InvalidEditException`.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Editor plugin website*. Also it can be downloaded as a *zip archive from the website*.

```
package simple.documentation.framework.extensions;

/**
 * Specific editing support for SDF documents.
 * Handles typing and paste events inside section and tables.
 */
public class SDFSchemaAwareEditingHandler implements
AuthorSchemaAwareEditingHandler {
```

Typing events can be handled using the `handleTyping` method. For example, the `SDFSchemaAwareEditingHandler` checks if the schema is not a learned one, was loaded successfully and *Smart Paste* is active. If these conditions are met, the event will be handled.

```
/**
 * @see
ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler#handleTyping(int,
 char, ro.sync.ecss.extensions.api.AuthorAccess)
 */
public boolean handleTyping(int offset, char ch, AuthorAccess authorAccess)
throws InvalidEditException {
  boolean handleTyping = false;
  AuthorSchemaManager authorSchemaManager =
authorAccess.getDocumentController().getAuthorSchemaManager();
  if (!authorSchemaManager.isLearnSchema() &&
      !authorSchemaManager.hasLoadingErrors() &&
     authorSchemaManager.getAuthorSchemaAwareOptions().isEnableSmartTyping())
 {
    try {
      AuthorDocumentFragment characterFragment =

authorAccess.getDocumentController().createNewDocumentTextFragment(String.valueOf(ch));

     handleTyping = handleInsertionEvent(offset, new AuthorDocumentFragment[]
 {characterFragment}, authorAccess);
    } catch (AuthorOperationException e) {
      throw new InvalidEditException(e.getMessage(), "Invalid typing event: "
 + e.getMessage(), e, false);
    }
  }
  return handleTyping;
}
```

Implementing the `AuthorSchemaAwareEditingHandler` gives the possibility to handle other events like: the keyboard delete event at the given offset (using Delete or Backspace keys), delete element tags, delete selection, join elements or paste fragment.

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

Configuring a Content Completion Handler

You can filter or contribute to items offered for content completion by implementing the *`ro.sync.contentcompletion.xml.SchemaManagerFilter`* interface.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

```
import java.util.List;

import ro.sync.contentcompletion.xml.CIAttribute;
import ro.sync.contentcompletion.xml.CIElement;
import ro.sync.contentcompletion.xml.CIValue;
import ro.sync.contentcompletion.xml.Context;
import ro.sync.contentcompletion.xml.SchemaManagerFilter;
import ro.sync.contentcompletion.xml.WhatAttributesCanGoHereContext;
import ro.sync.contentcompletion.xml.WhatElementsCanGoHereContext;
import ro.sync.contentcompletion.xml.WhatPossibleValuesHasAttributeContext;

public class SDFSchemaManagerFilter implements SchemaManagerFilter {
```

You can implement the various callbacks of the interface either by returning the default values given by Oxygen XML Editor plugin or by contributing to the list of proposals. The filter can be applied on elements, attributes or on their values. Attributes filtering can be implemented using the `filterAttributes` method and changing the default content completion list of *`ro.sync.contentcompletion.xml.CIAttribute`* for the element provided by the current *`ro.sync.contentcompletion.xml.WhatAttributesCanGoHereContext`* context. For example, the `SDFSchemaManagerFilter` checks if the element from the current context is the `table` element and adds the `frame` attribute to the `table` list of attributes.

```
/**
 * Filter attributes of the "table" element.
 */
public List<CIAttribute> filterAttributes(List<CIAttribute> attributes,
    WhatAttributesCanGoHereContext context) {
  // If the element from the current context is the 'table' element add the
  // attribute named 'frame' to the list of default content completion proposals

  if (context != null) {
    ContextElement contextElement = context.getParentElement();
    if ("table".equals(contextElement.getQName())) {
      CIAttribute frameAttribute = new CIAttribute();
      frameAttribute.setName("frame");
      frameAttribute.setRequired(false);
      frameAttribute.setFixed(false);
      frameAttribute.setDefaultValue("void");
      if (attributes == null) {
        attributes = new ArrayList<CIAttribute>();
      }
      attributes.add(frameAttribute);
    }
  }
  return attributes;
}
```

The elements that can be inserted in a specific context can be filtered using the `filterElements` method. The `SDFSchemaManagerFilter` uses this method to replace the `td` child element with the `th` element when `header` is the current context element.

```
public List<CIElement> filterElements(List<CIElement> elements,
    WhatElementsCanGoHereContext context) {
  // If the element from the current context is the 'header' element remove
the
  // 'td' element from the list of content completion proposals and add the
  // 'th' element.
  if (context != null) {
```

```
  Stack<ContextElement> elementStack = context.getElementStack();
  if (elementStack != null) {
   ContextElement contextElement = context.getElementStack().peek();
   if ("header".equals(contextElement.getQName())) {
    if (elements != null) {
     for (Iterator<CIElement> iterator = elements.iterator();
iterator.hasNext();) {
      CIElement element = iterator.next();
      // Remove the 'td' element
      if ("td".equals(element.getQName())) {
       elements.remove(element);
       break;
      }
     }
    } else {
     elements = new ArrayList<CIElement>();
    }
    // Insert the 'th' element in the list of content completion proposals

    CIElement thElement = new SDFElement();
    thElement.setName("th");
    elements.add(thElement);
   }
  }
 } else {
  // If the given context is null then the given list of content completion
 elements contains
  // global elements.
 }
 return elements;
}
```

The elements or attributes values can be filtered using the `filterElementValues` or `filterAttributeValues` methods.

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

Configuring a Link target element finder

The link target reference finder represents the support for finding references from links which indicate specific elements inside an XML document. This support will only be available if a schema is associated with the document type.

If you do not define a custom link target reference finder, the `DefaultElementLocatorProvider` implementation will be used by default. The interface which should be implemented for a custom link target reference finder is *`ro.sync.ecss.extensions.api.link.ElementLocatorProvider`*. As an alternative, the *`ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider`* implementation can also be extended.

The used `ElementLocatorProvider` will be queried for an *`ElementLocator`* when a link location must be determined (when a link is clicked). Then, to find the corresponding (linked) element, the obtained `ElementLocator` will be queried for each element from the document.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

The `DefaultElementLocatorProvider` implementation

The `DefaultElementLocatorProvider` implementation offers support for the most common types of links:

- links based on ID attribute values
- XPointer element() scheme

The method `getElementLocator` determines what `ElementLocator` should be used. In the default implementation it checks if the link is an XPointer element() scheme otherwise it assumes it is an ID. A non-null `IDTypeVerifier` will always be provided if a schema is associated with the document type.

The `link` string argument is the "anchor" part of the of the URL which is composed from the value of the link property specified for the link element in the CSS.

```
public ElementLocator getElementLocator(IDTypeVerifier idVerifier,
               String link) {
  ElementLocator elementLocator = null;
  try {
    if(link.startsWith("element(")){
      // xpointer element() scheme
      elementLocator = new XPointerElementLocator(idVerifier, link);
    } else {
      // Locate link element by ID
      elementLocator = new IDElementLocator(idVerifier, link);
    }
  } catch (ElementLocatorException e) {
    logger.warn("Exception when create element locator for link: "
        + link + ". Cause: " + e, e);
  }
  return elementLocator;
}
```

The `XPointerElementLocator` implementation

`XPointerElementLocator` is an implementation of the abstract class *`ro.sync.ecss.extensions.api.link.ElementLocator`* for links that have one of the following XPointer element() scheme patterns:

element(*elementID*) Locate the element with the specified id.

element(*/1/2/3*) A child sequence appearing alone identifies an element by means of stepwise navigation, which is directed by a sequence of integers separated by slashes (/); each integer n locates the nth child element of the previously located element.

element(*elementID/3/4*) A child sequence appearing after a *NCName* identifies an element by means of stepwise navigation, starting from the element located by the given name.

The constructor separates the id/integers which are delimited by slashes(/) into a sequence of identifiers (an XPointer path). It will also check that the link has one of the supported patterns of the XPointer element() scheme.

```
public XPointerElementLocator(IDTypeVerifier idVerifier, String link)
                      throws ElementLocatorException {
  super(link);
  this.idVerifier = idVerifier;

  link = link.substring("element(".length(), link.length() - 1);

  StringTokenizer stringTokenizer = new StringTokenizer(link, "/", false);
  xpointerPath = new String[stringTokenizer.countTokens()];
  int i = 0;
  while (stringTokenizer.hasMoreTokens()) {
    xpointerPath[i] = stringTokenizer.nextToken();
    boolean invalidFormat = false;

    // Empty xpointer component is not supported
    if(xpointerPath[i].length() == 0){
      invalidFormat = true;
    }
    if(i > 0){
      try {
        Integer.parseInt(xpointerPath[i]);
      } catch (NumberFormatException e) {
        invalidFormat = true;
      }
    }

    if(invalidFormat){
      throw new ElementLocatorException(
       "Only the element() scheme is supported when locating XPointer links."

        + "Supported formats: element(elementID), element(/1/2/3),
              element(elemID/2/3/4).");
    }
    i++;
  }

  if(Character.isDigit(xpointerPath[0].charAt(0))){
    // This is the case when xpointer have the following pattern /1/5/7
    xpointerPathDepth = xpointerPath.length;
  } else {
    // This is the case when xpointer starts with an element ID
    xpointerPathDepth = -1;
    startWithElementID  = true;
  }
}
```

The method `startElement` will be invoked at the beginning of every element in the XML document(even when the element is empty). The arguments it takes are

uri	The namespace URI, or the empty string if the element has no namespace URI or if namespace processing is disabled.
localName	Local name of the element.
qName	Qualified name of the element.
atts	Attributes attached to the element. If there are no attributes, this argument will be empty.

The method returns `true` if the processed element is found to be the one indicated by the link.

The `XPointerElementLocator` implementation of the `startElement` will update the depth of the current element and keep the index of the element in its parent. If the `xpointerPath` starts with an element ID then the current element ID is verified to match the specified ID. If this is the case the depth of the XPointer is updated taking into account the depth of the current element.

If the XPointer path depth is the same as the current element depth then the kept indices of the current element path are compared to the indices in the XPointer path. If all of them match then the element has been found.

```
public boolean startElement(String uri, String localName,
        String name, Attr[] atts) {
  boolean linkLocated = false;
  // Increase current element document depth
  startElementDepth ++;

  if (endElementDepth != startElementDepth) {
    // The current element is the first child of the parent
    currentElementIndexStack.push(new Integer(1));
  } else {
    // Another element in the parent element
    currentElementIndexStack.push(new Integer(lastIndexInParent + 1));
  }

  if (startWithElementID) {
    // This the case when xpointer path starts with an element ID.
    String xpointerElement = xpointerPath[0];
    for (int i = 0; i < atts.length; i++) {
      if(xpointerElement.equals(atts[i].getValue())){
        if(idVerifier.hasIDType(
            localName, uri, atts[i].getQName(), atts[i].getNamespace())){
          xpointerPathDepth = startElementDepth + xpointerPath.length - 1;
          break;
        }
      }
    }
  }

  if (xpointerPathDepth == startElementDepth){
    // check if xpointer path matches with the current element path
    linkLocated = true;
    try {
      int xpointerIdx = xpointerPath.length - 1;
      int stackIdx = currentElementIndexStack.size() - 1;
      int stopIdx = startWithElementID ? 1 : 0;
      while (xpointerIdx >= stopIdx && stackIdx >= 0) {
        int xpointerIndex = Integer.parseInt(xpointerPath[xpointerIdx]);
        int currentElementIndex =
          ((Integer)currentElementIndexStack.get(stackIdx)).intValue();
        if(xpointerIndex != currentElementIndex) {
          linkLocated = false;
          break;
        }

        xpointerIdx--;
        stackIdx--;
      }

    } catch (NumberFormatException e) {
      logger.warn(e,e);
    }
  }
  return linkLocated;
}
```

The method `endElement` will be invoked at the end of every element in the XML document (even when the element is empty).

The `XPointerElementLocator` implementation of the `endElement` updates the depth of the current element path and the index of the element in its parent.

```
public void endElement(String uri, String localName, String name) {
  endElementDepth = startElementDepth;
  startElementDepth --;
  lastIndexInParent = ((Integer)currentElementIndexStack.pop()).intValue();
}
```

The `IDElementLocator` implementation

The `IDElementLocator` is an implementation of the abstract class *`ro.sync.ecss.extensions.api.link.ElementLocator`* for links that use an **id**.

The constructor only assigns field values and the method `endElement` is empty for this implementation.

The method `startElement` checks each of the element's attribute values and when one matches the link, it considers the element found if one of the following conditions is satisfied:

- the qualified name of the attribute is `xml:id`
- the attribute type is ID

The attribute type is checked with the help of the method `IDTypeVerifier.hasIDType`.

```
public boolean startElement(String uri, String localName,
        String name, Attr[] atts) {
  boolean elementFound = false;
  for (int i = 0; i < atts.length; i++) {
    if (link.equals(atts[i].getValue())) {
      if("xml:id".equals(atts[i].getQName())) {
        // xml:id attribute
        elementFound = true;
      } else {
        // check if attribute has ID type
        String attrLocalName =
          ExtensionUtil.getLocalName(atts[i].getQName());
        String attrUri = atts[i].getNamespace();
        if (idVerifier.hasIDType(localName, uri, attrLocalName, attrUri)) {
          elementFound = true;
        }
      }
    }
  }

  return elementFound;
}
```

Creating a customized link target reference finder

If you need to create a custom link target reference finder you can do so by creating the class which will implement the *`ro.sync.ecss.extensions.api.link.ElementLocatorProvider`* interface. As an alternative, your class could extend *`ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider`*, the default implementation.

Note: The complete source code of the *`ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider`*, *`ro.sync.ecss.extensions.commons.IDElementLocator`* or *`ro.sync.ecss.extensions.commons.XPointerElementLocator`* can be found in the Oxygen Default Frameworks project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

Configuring a custom Drag and Drop listener

You can add your own drag and drop listener implementation of `ro.sync.ecss.extensions.api.DnDHandler`. You can choose from three interfaces to implement depending on whether you are using the framework with the Oxygen XML Editor plugin Eclipse plugin or the standalone version or if you want to add the handler for the Text or Author modes.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

Table 1: Interfaces for the DnD listener

Interface	Description
`ro.sync.exml.editor.xmleditor.pageauthor.AuthorCustomDnDHandler`	Receives callbacks from the Oxygen XML Editor plugin standalone application for Drag And Drop in Author mode.
`com.oxygenxml.editor.editors.author.AuthorDnDListener`	Receives callbacks from the Oxygen XML Editor plugin Eclipse plugin for Drag And Drop in Author mode.
`com.oxygenxml.editor.editors.TextDnDListener`	Receives callbacks from the Oxygen XML Editor plugin Eclipse plugin for Drag And Drop in Text mode.

Configuring a References Resolver

You need to provide a handler for resolving references and obtain the content they refer. In our case the element which has references is **ref** and the attribute indicating the referred resource is **location**. You will have to implement a Java extension class for obtaining the referred resources.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

1. Create the class `simple.documentation.framework.ReferencesResolver`. This class must implement the *`ro.sync.ecss.extensions.api.AuthorReferenceResolver`* interface.

```
import ro.sync.ecss.extensions.api.AuthorReferenceResolver;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.node.AttrValue;
import ro.sync.ecss.extensions.api.node.AuthorElement;
import ro.sync.ecss.extensions.api.node.AuthorNode;

public class ReferencesResolver
      implements AuthorReferenceResolver {
```

2. The `hasReferences` method verifies if the handler considers the node to have references. It takes as argument an `AuthorNode` that represents the node which will be verified. The method will return `true` if the node is considered to have references. In our case, to be a reference the node must be an element with the name *ref* and it must have an attribute named *location*.

```
public boolean hasReferences(AuthorNode node) {
  boolean hasReferences = false;
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      hasReferences = attrValue != null;
    }
  }
  return hasReferences;
}
```

3. The method `getDisplayName` returns the display name of the node that contains the expanded referred content. It takes as argument an `AuthorNode` that represents the node for which the display name is needed. The referred content engine will ask this `AuthorReferenceResolver` implementation what is the display name for each node which is considered a reference. In our case the display name is the value of the *location* attribute from the *ref* element.

```
public String getDisplayName(AuthorNode node) {
  String displayName = "ref-fragment";
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        displayName = attrValue.getValue();
      }
    }
  }
  return displayName;
}
```

4. The method `resolveReference` resolves the reference of the node and returns a `SAXSource` with the parser and the parser's input source. It takes as arguments an `AuthorNode` that represents the node for which the reference needs resolving, the *systemID* of the node, the `AuthorAccess` with access methods to the Author data model and a SAX `EntityResolver` which resolves resources that are already opened in another editor or resolve resources through the XML catalog. In the implementation you need to resolve the reference relative to the *systemID*, and create a parser and an input source over the resolved reference.

```
public SAXSource resolveReference(
    AuthorNode node,
    String systemID,
    AuthorAccess authorAccess,
    EntityResolver  entityResolver) {
  SAXSource saxSource = null;

  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        String attrStringVal = attrValue.getValue();
        try {
          URL absoluteUrl = new URL(new URL(systemID),
              authorAccess.correctURL(attrStringVal));

          InputSource inputSource = entityResolver.resolveEntity(null,
              absoluteUrl.toString());
          if(inputSource == null) {
            inputSource = new InputSource(absoluteUrl.toString());
          }

          XMLReader xmlReader = authorAccess.newNonValidatingXMLReader();
          xmlReader.setEntityResolver(entityResolver);

          saxSource = new SAXSource(xmlReader, inputSource);
        } catch (MalformedURLException e) {
          logger.error(e, e);
        } catch (SAXException e) {
          logger.error(e, e);
        } catch (IOException e) {
          logger.error(e, e);
        }
      }
    }
  }

  return saxSource;
}
```

5. The method `getReferenceUniqueID` should return an unique identifier for the node reference. The unique identifier is used to avoid resolving the references recursively. The method takes as argument an `AuthorNode` that represents the node with the reference. In the implementation the unique identifier is the value of the *location* attribute from the *ref* element.

```
public String getDisplayName(AuthorNode node) {
  String displayName = "ref-fragment";
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        displayName = attrValue.getValue();
      }
    }
  }
  return displayName;
}
```

6. The method `getReferenceSystemID`should return the *systemID* of the referred content. It takes as arguments an `AuthorNode` that represents the node with the reference and the `AuthorAccess` with access methods to the Author data model. In the implementation you use the value of the *location* attribute from the *ref* element and resolve it relatively to the XML base URL of the node.

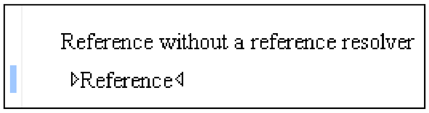
```
public String getReferenceSystemID(AuthorNode node,
                                   AuthorAccess authorAccess) {
  String systemID = null;
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        String attrStringVal = attrValue.getValue();
        try {
          URL absoluteUrl = new URL(node.getXMLBaseURL(),
              authorAccess.correctURL(attrStringVal));
          systemID = absoluteUrl.toString();
        } catch (MalformedURLException e) {
          logger.error(e, e);
        }
      }
    }
  }
  return systemID;
}
```

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

In the listing below, the XML document contains the **ref** element:

```
<ref location="referred.xml">Reference</ref>
```

When no reference resolver is specified, the reference has the following layout:

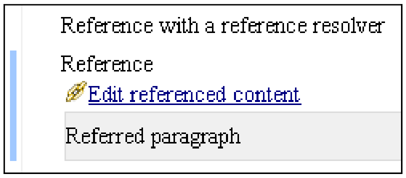


Reference without a reference resolver

▷Reference◁

Figure 183: Reference with no specified reference resolver

When the above implementation is configured, the reference has the expected layout:



Reference with a reference resolver

Reference

Edit referenced content

Referred paragraph

Figure 184: Reference with reference resolver

Configuring CSS Styles Filter

You can modify the CSS styles for each *`ro.sync.ecss.extensions.api.node.AuthorNode`* rendered in the Author mode using an implementation of *`ro.sync.ecss.extensions.api.StylesFilter`*. You can implement the various callbacks of the interface either by returning the default value given by Oxygen XML Editor plugin or by contributing to the value. The received styles `ro.sync.ecss.css.Styles` can be processed and values can be overwritten with your own. For example you can override the `KEY_BACKGROUND_COLOR` style to return your own implementation of *`ro.sync.exml.view.graphics.Color`* or override the `KEY_FONT` style to return your own implementation of *`ro.sync.exml.view.graphics.Font`*.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

For instance in our simple document example the filter can change the value of the `KEY_FONT` property for the `table` element:

```
package simple.documentation.framework;

import ro.sync.ecss.css.Styles;
import ro.sync.ecss.extensions.api.StylesFilter;
import ro.sync.ecss.extensions.api.node.AuthorNode;
import ro.sync.exml.view.graphics.Font;

public class SDFStylesFilter implements StylesFilter {

     public Styles filter(Styles styles, AuthorNode authorNode) {
         if (AuthorNode.NODE_TYPE_ELEMENT == authorNode.getType()
           && "table".equals(authorNode.getName())) {
          styles.setProperty(Styles.KEY_FONT, new Font(null, Font.BOLD, 12));

         }
         return styles;
     }
}
```

Configuring a Table Column Width Provider

In the documentation framework the `table` element as well as the table columns can have specified widths. In order for these widths to be considered by Author we need to provide the means to determine them. As explained in the *Styling the Table Element* section which describes the CSS properties needed for defining a table, if you use the table element attribute **width** Oxygen XML Editor plugin can determine the table width automatically. In this example the table has `col` elements with **width** attributes that are not recognized by default. You will need to implement a Java extension class to determine the column widths.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

1. Create the class `simple.documentation.framework.TableColumnWidthProvider`. This class must implement the *`ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider`* interface.

```
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperationException;
import ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider;
import ro.sync.ecss.extensions.api.WidthRepresentation;
import ro.sync.ecss.extensions.api.node.AuthorElement;

public class TableColumnWidthProvider
      implements AuthorTableColumnWidthProvider {
```

2. Method `init` is taking as argument an *`ro.sync.ecss.extensions.api.node.AuthorElement`* that represents the XML `table` element. In our case the column widths are specified in `col` elements from the `table` element. In such cases you must collect the span information by analyzing the `table` element.

```
 public void init(AuthorElement tableElement) {
  this.tableElement = tableElement;
  AuthorElement[] colChildren =
tableElement.getElementsByLocalName("customcol");
  if (colChildren != null && colChildren.length > 0) {
   for (int i = 0; i < colChildren.length; i++) {
    AuthorElement colChild = colChildren[i];
    if (i == 0) {
     colsStartOffset = colChild.getStartOffset();
    }
    if (i == colChildren.length - 1) {
     colsEndOffset = colChild.getEndOffset();
    }
    // Determine the 'width' for this col.
    AttrValue colWidthAttribute = colChild.getAttribute("width");
    String colWidth = null;
    if (colWidthAttribute != null) {
     colWidth = colWidthAttribute.getValue();
     // Add WidthRepresentation objects for the columns this 'customcol'
specification
     // spans over.
     colWidthSpecs.add(new WidthRepresentation(colWidth, true));
    }
   }
  }
 }
```

3. The method `isTableAcceptingWidth` should check if the table cells are `td`.

```
public boolean isTableAcceptingWidth(String tableCellsTagName) {
  return "td".equals(tableCellsTagName);
}
```

4. The method `isTableAndColumnsResizable` should check if the table cells are `td`. This method determines if the table and its columns can be resized by dragging the edge of a column.

```
public boolean isTableAndColumnsResizable(String tableCellsTagName) {
  return "td".equals(tableCellsTagName);
}
```

5. Methods `getTableWidth` and `getCellWidth` are used to determine the table and column width. The table layout engine will ask this *`ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider`* implementation what is the table width for each table element and the cell width for each cell element from the table that was marked as cell in the CSS using the property `display:table-cell`. The implementation is simple and

just parses the value of the **width** attribute. The methods must return `null` for the tables / cells that do not have a specified width.

```
 public WidthRepresentation getTableWidth(String tableCellsTagName) {
  WidthRepresentation toReturn = null;
  if (tableElement != null && "td".equals(tableCellsTagName)) {
   AttrValue widthAttr = tableElement.getAttribute("width");
   if (widthAttr != null) {
    String width = widthAttr.getValue();
    if (width != null) {
     toReturn = new WidthRepresentation(width, true);
    }
   }
  }
  return toReturn;
 }
```

```
 public List<WidthRepresentation> getCellWidth(AuthorElement cellElement, int 
 colNumberStart, 
   int colSpan) {
  List<WidthRepresentation> toReturn = null;
  int size = colWidthSpecs.size();
  if (size >= colNumberStart && size >= colNumberStart + colSpan) {
   toReturn = new ArrayList<WidthRepresentation>(colSpan);
   for (int i = colNumberStart; i < colNumberStart + colSpan; i ++) {
    // Add the column widths
    toReturn.add(colWidthSpecs.get(i));
   }
  }
  return toReturn;
 }
```

6. Methods `commitTableWidthModification` and `commitColumnWidthModifications` are used to commit changes made to the width of the table or its columns when using the mouse drag gestures.

```
 public void commitTableWidthModification(AuthorDocumentController 
authorDocumentController, 
   int newTableWidth, String tableCellsTagName) throws AuthorOperationException 
 {
  if ("td".equals(tableCellsTagName)) {
   if (newTableWidth > 0) {
    if (tableElement != null) {
     String newWidth = String.valueOf(newTableWidth);

     authorDocumentController.setAttribute(
       "width",
       new AttrValue(newWidth),
       tableElement);
    } else {
     throw new AuthorOperationException("Cannot find the element representing 
 the table.");
    }
   }
  }
 }
```

```
public void commitColumnWidthModifications(AuthorDocumentController 
authorDocumentController, 
   WidthRepresentation[] colWidths, String tableCellsTagName) throws 
AuthorOperationException {
  if ("td".equals(tableCellsTagName)) {
   if (colWidths != null && tableElement != null) {
    if (colsStartOffset >= 0 && colsEndOffset >= 0 && colsStartOffset <
colsEndOffset) {
     authorDocumentController.delete(colsStartOffset,
       colsEndOffset);
    }
    String xmlFragment = createXMLFragment(colWidths);
    int offset = -1;
    AuthorElement[] header = tableElement.getElementsByLocalName("header");
    if (header != null && header.length > 0) {
     // Insert the cols elements before the 'header' element
     offset = header[0].getStartOffset();
    }
    if (offset == -1) {
    throw new AuthorOperationException("No valid offset to insert the columns
 width specification.");
    }
    authorDocumentController.insertXMLFragment(xmlFragment, offset);
   }
  }
 }

 private String createXMLFragment(WidthRepresentation[] widthRepresentations)
 {
  StringBuffer fragment = new StringBuffer();
  String ns = tableElement.getNamespace();
  for (int i = 0; i < widthRepresentations.length; i++) {
   WidthRepresentation width = widthRepresentations[i];
   fragment.append("<customcol");
   String strRepresentation = width.getWidthRepresentation();
   if (strRepresentation != null) {
    fragment.append(" width=\"" + width.getWidthRepresentation() + "\"");
   }
   if (ns != null && ns.length() > 0) {
    fragment.append(" xmlns=\"" + ns + "\"");
   }
   fragment.append("/>");
  }
  return fragment.toString();
 }
```

7. The following three methods are used to determine what type of column width specifications the table column width provider support. In our case all types of specifications are allowed:

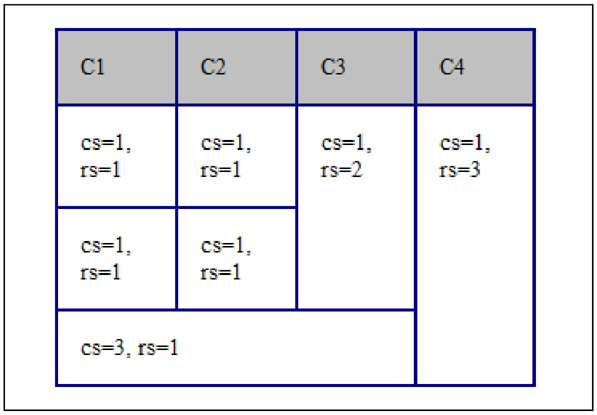
```

 public boolean isAcceptingFixedColumnWidths(String tableCellsTagName) {
  return true;
 }

 public boolean isAcceptingPercentageColumnWidths(String tableCellsTagName)
{
  return true;
 }

 public boolean isAcceptingProportionalColumnWidths(String tableCellsTagName)
 {
  return true;
 }
```

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

In the listing below, the XML document contains the table element:

```
<table width="300">
    <customcol width="50.0px"/>
    <customcol width="1*"/>
    <customcol width="2*"/>
    <customcol width="20%"/>
    <header>
        <td>C1</td>
        <td>C2</td>
        <td>C3</td>
        <td>C4</td>
    </header>
    <tr>
        <td>cs=1, rs=1</td>
        <td>cs=1, rs=1</td>
        <td row_span="2">cs=1, rs=2</td>
        <td row_span="3">cs=1, rs=3</td>
    </tr>
    <tr>
        <td>cs=1, rs=1</td>
        <td>cs=1, rs=1</td>
    </tr>
    <tr>
        <td column_span="3">cs=3, rs=1</td>
    </tr>
</table>
```

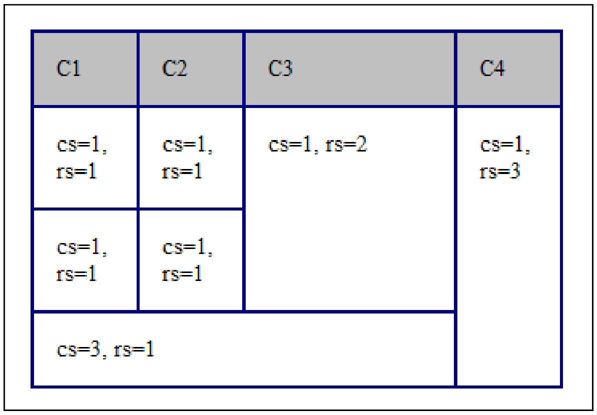
When no table column width provider is specified, the table has the following layout:



<table>
<tr><th>C1</th><th>C2</th><th>C3</th><th>C4</th></tr>
<tr><td>cs=1, rs=1</td><td>cs=1, rs=1</td><td rowspan="2">cs=1, rs=2</td><td rowspan="3">cs=1, rs=3</td></tr>
<tr><td>cs=1, rs=1</td><td>cs=1, rs=1</td></tr>
<tr><td colspan="3">cs=3, rs=1</td></tr>
</table>

Figure 185: Table layout when no column width provider is specified

When the above implementation is configured, the table has the correct layout:



<table>
<tr><th>C1</th><th>C2</th><th>C3</th><th>C4</th></tr>
<tr><td>cs=1, rs=1</td><td>cs=1, rs=1</td><td rowspan="2">cs=1, rs=2</td><td rowspan="3">cs=1, rs=3</td></tr>
<tr><td>cs=1, rs=1</td><td>cs=1, rs=1</td></tr>
<tr><td colspan="3">cs=3, rs=1</td></tr>
</table>

Figure 186: Columns with custom widths

Configuring a Table Cell Span Provider

In the documentation framework the `table` element can have cells that span over multiple columns and rows. As explained in the *Styling the Table Element* section which describes the CSS properties needed for defining a table, you need to indicate Oxygen XML Editor plugin a method to determine the cell spanning. If you use the cell element attributes **rowspan** and **colspan** or **rows** and **cols**, Oxygen XML Editor plugin can determine the cell spanning automatically. In our example the `td` element uses the attributes **row_span** and **column_span** that are not recognized by default. You will need to implement a Java extension class for defining the cell spanning.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

1. Create the class `simple.documentation.framework.TableCellSpanProvider`. This class must implement the *`ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider`* interface.

```
import ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider;
import ro.sync.ecss.extensions.api.node.AttrValue;
import ro.sync.ecss.extensions.api.node.AuthorElement;

public class TableCellSpanProvider
      implements AuthorTableCellSpanProvider {
```

2. The `init` method is taking as argument the *`ro.sync.ecss.extensions.api.node.AuthorElement`* that represents the XML `table` element. In our case the cell span is specified for each of the cells so you leave this method empty. However there are cases like the table CALS model when the cell spanning is specified in the `table` element. In such cases you must collect the span information by analyzing the `table` element.

```
public void init(AuthorElement table) {
}
```

3. The `getColSpan` method is taking as argument the table cell. The table layout engine will ask this `AuthorTableSpanSupport` implementation what is the column span and the row span for each XML element from the table that was marked as cell in the CSS using the property `display:table-cell`. The implementation is simple and just parses the value of **column_span** attribute. The method must return `null` for all the cells that do not change the span specification.

```
public Integer getColSpan(AuthorElement cell) {
  Integer colSpan = null;

  AttrValue attrValue = cell.getAttribute("column_span");
  if(attrValue != null) {
    // The attribute was found.
    String cs = attrValue.getValue();
    if(cs != null) {
      try {
        colSpan = new Integer(cs);
      } catch (NumberFormatException ex) {
        // The attribute value was not a number.
      }
    }
  }
  return colSpan;
}
```

4. The row span is determined in a similar manner:

```
public Integer getRowSpan(AuthorElement cell) {
  Integer rowSpan = null;

  AttrValue attrValue = cell.getAttribute("row_span");
  if(attrValue != null) {
    // The attribute was found.
    String rs = attrValue.getValue();
    if(rs != null) {
      try {
        rowSpan = new Integer(rs);
      } catch (NumberFormatException ex) {
        // The attribute value was not a number.
      }
    }
  }
  return rowSpan;
}
```

5. The method `hasColumnSpecifications` always returns `true` considering column specifications always available.

```
public boolean hasColumnSpecifications(AuthorElement tableElement) {
  return true;
}
```

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

6. In the listing below, the XML document contains the table element:

```
<table>
    <header>
        <td>C1</td>
        <td>C2</td>
        <td>C3</td>
        <td>C4</td>
    </header>
    <tr>
        <td>cs=1, rs=1</td>
        <td column_span="2" row_span="2">cs=2, rs=2</td>
        <td row_span="3">cs=1, rs=3</td>
    </tr>
    <tr>
        <td>cs=1, rs=1</td>
    </tr>
    <tr>
```

```
            <td column_span="3">cs=3, rs=1</td>
        </tr>
</table>
```

When no table cell span provider is specified, the table has the following layout:

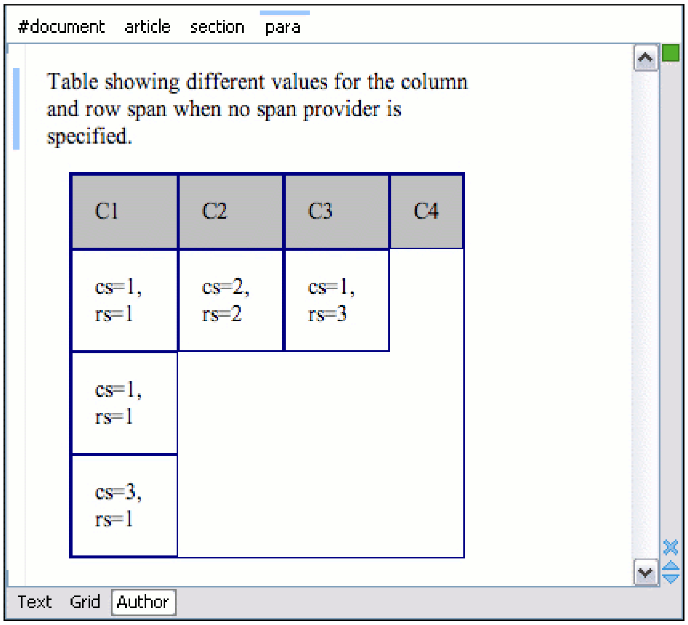


Table showing different values for the column and row span when no span provider is specified.

C1	C2	C3	C4
cs=1, rs=1	cs=2, rs=2	cs=1, rs=3	
cs=1, rs=1			
cs=3, rs=1			

Figure 187: Table layout when no cell span provider is specified

When the above implementation is configured, the table has the correct layout:

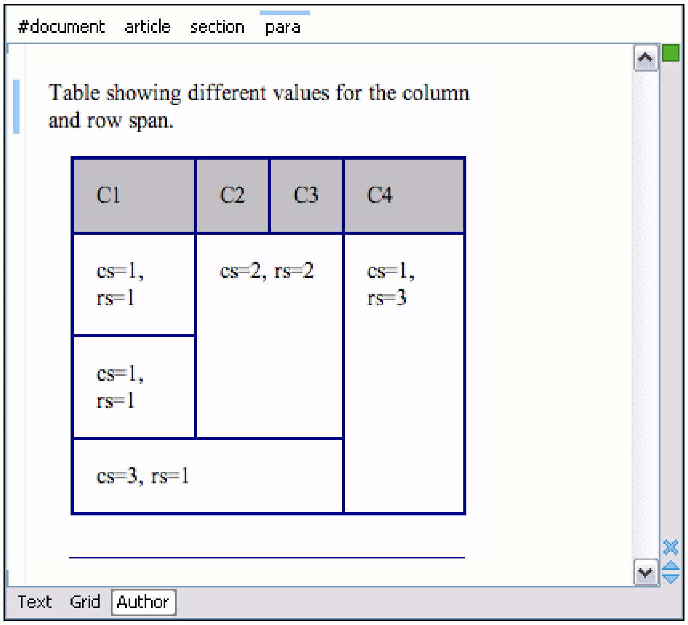


Table showing different values for the column and row span.

C1	C2	C3	C4
cs=1, rs=1	cs=2, rs=2		cs=1, rs=3
cs=1, rs=1			
cs=3, rs=1			

Figure 188: Cells spanning multiple rows and columns.

Configuring an Unique Attributes Recognizer

The `ro.sync.ecss.extensions.api.UniqueAttributesRecognizer` interface can be implemented if you want to provide for your framework the following features:

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

- **Automatic ID generation** - You can automatically generate unique IDs for newly inserted elements. Implementations are already available for the DITA and Docbook frameworks. The following methods can be implemented to accomplish this: `assignUniqueIDs(int startOffset, int endOffset)`, `isAutoIDGenerationActive()`
- **Avoiding copying unique attributes when "Split" is called inside an element** - You can split the current block element by pressing the "Enter" key and then choosing "Split". This is a very useful way to create new paragraphs, for example. All attributes are by default copied on the new element but if those attributes are IDs you sometimes want to avoid creating validation errors in the editor. Implementing the following method, you can decide whether an attribute should be copied or not during the split: `boolean copyAttributeOnSplit(String attrQName, AuthorElement element)`

  Tip:

  The *`ro.sync.ecss.extensions.commons.id.DefaultUniqueAttributesRecognizer`* class is an implementation of the interface which can be extended by your customization to provide easy assignation of IDs in your framework. You can also check out the DITA and Docbook implementations of `ro.sync.ecss.extensions.api.UniqueAttributesRecognizer` to see how they were implemented and connected to the extensions bundle.

Configuring an XML Node Renderer Customizer

You can use this API extension to customize the way an XML node is rendered in the Author **Outline** view, Author breadcrumb navigation bar, Text mode **Outline** view, content completion assistant window or **DITA Maps Manager** view.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

There are two methods to provide an implementation of `ro.sync.exml.workspace.api.node.customizer.XMLNodeRendererCustomizer`:

- as a part of a bundle - returning it from the `createXMLNodeCustomizer()` method of the *`ExtensionsBundle`* associated with your document type in the **Document type** dialog, **Extensions** tab, **Extensions bundle** field.
- as an individual extension - associated with your document type in the **Document type** dialog, **Extensions** tab, **Individual extensions** section, **XML node renderer customizer** field.

Styling the `table` Element.

There are standard CSS properties used to indicate what elements are tables, table rows and table cells. What CSS is missing is the possibility to indicate the cell spanning. Oxygen XML Author offers support for adding an extension to solve this problem. This will be presented in the next chapters.

The table in this example is a simple one. The header must be formatted in a different way than the ordinary rows, so it will have a background color.

```
table{
    display:table;
    border:1px solid navy;
    margin:1em;
    max-width:1000px;
    min-width:150px;
}

table[width]{
  width:attr(width, length);
}
```

```
tr, header{
    display:table-row;
}

header{
    background-color: silver;
    color:inherit
}

td{
  display:table-cell;
  border:1px solid navy;
  padding:1em;
}
```

Because in the schema the `td` tag has the attributes **row_span** and **column_span** that are not automatically recognized by Oxygen XML Author, a Java extension will be implemented which will provide information about the cell spanning. See the section *Configuring a Table Cell Span Provider*.

Because the column widths are specified by the attributes **width** of the elements `customcol` that are not automatically recognized by Oxygen XML Author, it is necessary to implement a Java extension which will provide information about the column widths. See the section *Configuring a Table Column Width Provider*.

Sample project reference

Note: The complete source code can be found in the Simple Documentation Framework project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

Note: The complete source code of the *`ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider`*, *`ro.sync.ecss.extensions.commons.IDElementLocator`* or *`ro.sync.ecss.extensions.commons.XPointerElementLocator`* can be found in the Oxygen Default Frameworks project, included in the *Oxygen Author SDK zip* available for download *on the Oxygen XML Editor plugin website*.

Note: The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML website*. Also it can be downloaded as a *zip archive from the website*.

Customizing the Default CSS of a Document Type

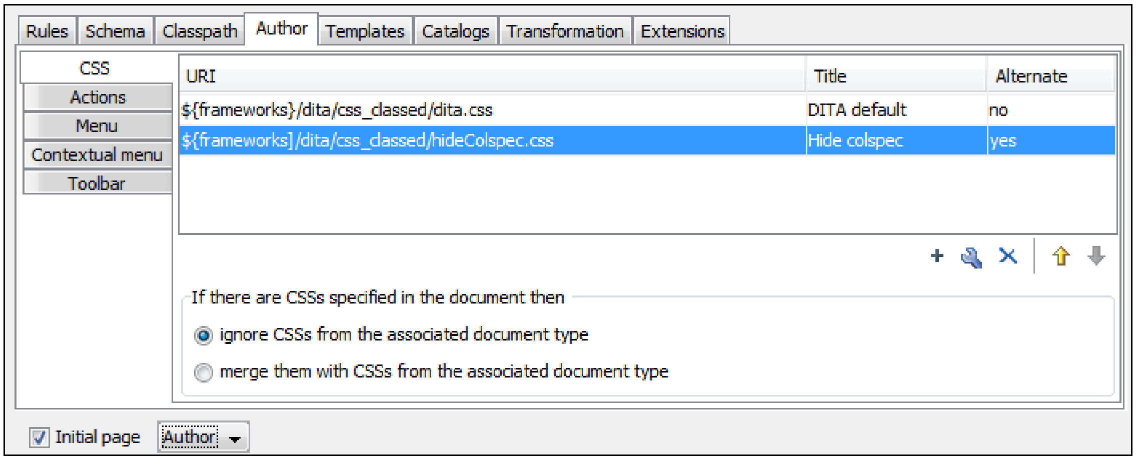
The easiest way of customizing the default CSS stylesheet of a document type is to create a new CSS stylesheet in the same folder as the customized one, import the customized CSS stylesheet and set the new stylesheet as the default CSS of the document type. For example let us customize the default CSS for DITA documents by changing the background color of the *task* and *topic* elements to red.

1. First you create a new CSS stylesheet called `my_dita.css` in the folder `${frameworks}/dita/css_classed` where the default stylesheet called `dita.css` is located. `${frameworks}` is the subfolder `frameworks` of the Oxygen XML Editor. The new stylesheet `my_dita.css` contains:

   ```
   @import "dita.css";

   task, topic{
       background-color:red;
   }
   ```

2. To set the new stylesheet as the default CSS stylesheet for DITA documents first open the Document Type Association preferences panel from menu **Options** > **Preferences** > **Document Type Association**. Select the DITA document type and start editing it by pressing the Edit button. In the Author tab of the document type edit dialog change the

URI of the default CSS stylesheet from `${frameworks}/dita/css_classed/dita.css` to `${frameworks}/dita/css_classed/my_dita.css`.



URI	Title	Alternate
${frameworks}/dita/css_classed/dita.css	DITA default	no
${frameworks}/dita/css_classed/hideColspec.css	Hide colspec	yes

If there are CSSs specified in the document then

● ignore CSSs from the associated document type

○ merge them with CSSs from the associated document type

☑ Initial page Author

Figure 189: Set the location of the default CSS stylesheet

3. Press OK in all the dialogs to validate the changes. Now you can start editing DITA documents based on the new CSS stylesheet. You can edit the new CSS stylesheet itself at any time and see the effects on rendering DITA XML documents in the Author mode by running the *Refresh* action available on the Author toolbar and on the DITA menu.

Document Type Sharing

Oxygen has support for allowing you to share the customizations which you have made for a specific XML type by creating your own *Document Type* in the *Document Type Association* preferences page.

A document type can be shared between authors in two ways:

- Save it externally in a separate framework folder in the `OXYGEN_INSTALL_DIR/frameworks` directory.

  Important: In order for this approach to work you will need to have Oxygen XML Editor plugin installed to a folder with full write access.

  Please see the following steps:

  1. Create a new directory in the `OXYGEN_INSTALL_DIR/frameworks` for your new framework. This directory will contain resources for your framework (CSS files, new file templates, schemas used for validation, catalogs). See the **Docbook** framework structure from the `OXYGEN_INSTALL_DIR/frameworks/docbook` as an example.
  2. Create your new custom document type and save it external in the newly created framework directory (with a name like `custom.framework`).
  3. Configure the custom document type according to your needs, take special care to make all file references relative to the `OXYGEN_INSTALL_DIR/frameworks` directory by using the `${frameworks}` editor variable. The *Author Developer Guide* contains all details necessary for creating and configuring a new document type.
  4. If everything went fine then you should have a new configuration file saved in: `OXYGEN_INSTALL_DIR/frameworks/your_framework_dir/custom.framework` after the Preferences are saved.
  5. You can then share the new framework directory with other users (have them copy it to their `OXYGEN_INSTALL_DIR/frameworks` directory) and the new document type will be available in the list of Document Types when Oxygen XML Editor plugin is started.

- Save the document type at project level in the *Document Type Association* page.

  Please see the following steps:

1. Create a new directory with full write access somewhere on your local drive which will contain the Oxygen project file and associated document type resources (CSS files, new file templates, schemas used for validation, catalogs).
2. From the Oxygen *Project view* create a new project and save it in the newly created directory.
3. In the Oxygen Preferences *Document Type Association* page switch the radio button at the bottom of the page to **Project Options**.
4. Create your new custom document type using the default **internal** storage for it. It will actually be saved in the previously chosen Oxygen XML Editor plugin project `.xpr` file.
5. Configure the custom document type according to your needs, take special care to make all file references relative to the project directory by using the `${pd}` editor variable. The *Author Developer Guide* contains all details necessary for creating and configuring a new document type.
6. You can then share the new project directory with other users and if they open in the *Project view* the customized project then the new document type will be available in the list of Document Types.

Other Author Features

The following features can be found in the Oxygen XML Editor plugin Author SDK:

Persistent Highlights

You can create or remove custom persistent highlights, set their properties, and customize their appearance. They get serialized in the XML document as processing instructions, having the following format:

```
<?oxy_custom_start prop1="val1"....?> xml content <?oxy_custom_end?>
```

The functionality is available in the `AuthorPersistentHighlighter` class, accessible through `AuthorEditorAccess#getPersistentHighlighter()` method. For more information, see JavaDoc online at: *http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/index.html*

CSS Support in Author

Author editing mode supports most CSS 2.1 selectors, a lot of CSS 2.1 properties and a few CSS 3 selectors. Also some custom functions and properties that extend the W3C CSS specification and are useful for URL and string manipulation are available to the developer who creates an Author editing framework.

CSS 2.1 Features

This section enumerates the CSS 2.1 features that are supported by Oxygen XML Editor plugin .

Supported CSS 2.1 Selectors

Expression	Name	Description / Example
`*`	Universal selector	Matches any element
`E`	Type selector	Matches any `E` element (i. e. an element with the local name `E`)
`E F`	Descendant selector	Matches any `F` element that is a descendant of an `E` element.
`E > F`	Child selectors	Matches any `F` element that is a child of an element `E`.
`E:first-child`	The `:first-child` pseudo-class	Matches element `E` when `E` is the first child of its parent.

Expression	**Name**	**Description / Example**
`E:lang(c)`	The `:lang()` pseudo-class	Matches element of type `E` if it is in (human) language `c` (the document language specifies how language is determined).
`E + F`	Adjacent selector	Matches any `F` element immediately preceded by a sibling element `E`.
`E[foo]`	Attribute selector	Matches any `E` element with the `"foo"` attribute set (whatever the value).
`E[foo="warning"]`	Attribute selector	Matches any `E` element whose `"foo"` attribute value is exactly equal to `"warning"`.
`E[foo~="warning"]`	Attribute selector	Matches any `E` element whose `"foo"` attribute value is a list of space-separated values, one of which is exactly equal to `"warning"`.
`E[lang|="en"]`	Attribute selector	Matches any `E` element whose `"lang"` attribute has a hyphen-separated list of values beginning (from the left) with `"en"`.
`E:before and E:after`	Pseudo elements	The `':before'` and `':after'` pseudo-elements can be used to insert generated content before or after an element's content.

Unsupported CSS 2.1 Selectors

Expression	**Name**	**Description / Example**
E#myid	ID selectors	Matches any E element with ID equal to "myid".
E:link, E:visited	The link pseudo-class	Matches element E if E is the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited).
E:active, E:hover, E:focus	The dynamic pseudo-classes	Matches E during certain user actions.
E:first-line	The :first-line pseudo-class	The :first-line pseudo-element applies special styles to the contents of the first formatted line of a paragraph.
E:first-letter	The :first-letter pseudo-class	The :first-letter pseudo-element must select the first letter of the first line of a block, if it is not preceded by any other content (such as images or inline tables) on its line. The :first-letter pseudo-element may be used for "initial caps" and "drop caps", which are common typographical effects.

CSS 2.1 Properties

Oxygen XML validates all CSS 2.1 properties, but does not render in Author mode *aural* and *paged* categories properties, as well as some of the values of the *visual* category, listed below under the **Ignored Values** column.

Name	**Rendered Values**	**Ignored Values**
`'background-attachment'`		ALL
`'background-color'`	`<color> | inherit`	`transparent`
`'background-image'`		ALL
`'background-position'`		ALL
`'background-repeat'`		ALL

Name	**Rendered Values**	**Ignored Values**
'background'		ALL
'border-collapse'		ALL
'border-color'	<color> | inherit	transparent
'border-spacing'		ALL
'border-style'	<border-style> | inherit	
'border-top' 'border-right' 'border-bottom' 'border-left'	[<border-width> || <border-style> || 'border-top-color'] | inherit	
'border-top-color' 'border-right-color' 'border-bottom-color' 'border-left-color'	<color> | inherit	transparent
'border-top-style' 'border-right-style' 'border-bottom-style' 'border-left-style'	<border-style> | inherit	
'border-top-width' 'border-right-width' 'border-bottom-width' 'border-left-width'	<border-width> | inherit	
'border-width'	<border-width> | inherit	
'border'	[<border-width> || <border-style> || 'border-top-color'] | inherit	
'bottom'		ALL
'caption-side'		ALL
'clear'		ALL
'clip'		ALL
'color'	<color> | inherit	
'content'	normal | none | [<string> | <URI> | <counter> | attr(<identifier>) | open-quote | close-quote]+ | inherit	no-open-quote | no-close-quote
'counter-increment'	[<identifier> <integer> ?]+ | none | inherit	
'counter-reset'	[<identifier> <integer> ?]+ | none | inherit	
'cursor'		ALL
'direction'	ltr	rtl | inherit

Name	Rendered Values	Ignored Values
'display'	inline | block | list-item | table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit	run-in | inline-block | inline-table - considered block
'empty-cells'	show | hide | inherit	
'float'		ALL
'font-family'	[[<family-name> | <generic-family>] [, <family-name> | <generic-family>]*] | inherit	
'font-size'	<absolute-size> | <relative-size> | <length> | <percentage> | inherit	
'font-style'	normal | italic | oblique | inherit	
'font-variant'		ALL
'font-weight'	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit	
'font'	[['font-style' || 'font-weight']? 'font-size' [/ 'line-height']? 'font-family'] | inherit	'font-variant' 'line-height' caption | icon | menu | message-box | small-caption | status-bar
'height'		ALL
'left'		ALL
'letter-spacing'		ALL
'line-height'	normal | <number> | <length> | <percentage> | inherit	
'list-style-image'		ALL
'list-style-position'		ALL
'list-style-type'	disc | circle | square | decimal | lower-roman | upper-roman | lower-latin | upper-latin | lower-alpha | upper-alpha | box | diamond | check | hyphen | none | inherit	lower-greek | armenian | georgian

Name	**Rendered Values**	**Ignored Values**
`'list-style'`	`[ 'list-style-type' ] | inherit`	`'list-style-position' || 'list-style-image'`
`'margin-right' 'margin-left'`	`<margin-width> | inherit | auto`	
`'margin-top' 'margin-bottom'`	`<margin-width> | inherit`	
`'margin'`	`<margin-width> | inherit | auto`	
`'max-height'`		ALL
`'max-width'`	`<length> | <percentage> | none | inherit` - supported for block-level and replaced elements, e.g. images, tables, table cells.	
`'min-height'`		ALL
`'min-width'`	`<length> | <percentage> | inherit` - supported for block-level and replaced elements, e. g. images, tables, table cells.	
`'outline-color'`		ALL
`'outline-style'`		ALL
`'outline-width'`		ALL
`'outline'`		ALL
`'overflow'`		ALL

`'padding-top'`
`'padding-right'`
`'padding-bottom'`
`'padding-left'`
	`<padding-width> | inherit`	

`'padding'`	`<padding-width> | inherit`	
`'position'`		ALL
`'quotes'`		ALL
`'right'`		ALL
`'table-layout'`	`auto`	`fixed | inherit`
`'text-align'`	`left | right | center | inherit`	`justify`
`'text-decoration'`	`none | [ underline || overline || line-through ] | inherit`	`blink`
`'text-indent'`		ALL
`'text-transform'`	ALL	
`'top'`		ALL
`'unicode-bidi'`		ALL

Name	Rendered Values	Ignored Values
'vertical-align'	baseline | sub | super | top | text-top | middle | bottom | text-bottom | inherit	<percentage> | <length>
'visibility'	visible | hidden | inherit	collapse
'white-space'	normal | pre | nowrap | pre-wrap | pre-line	
'width'	<length> | <percentage> | auto | inherit - supported for block-level and replaced elements, e.g. images, tables, table cells.	
'word-spacing'		ALL
'z-index'		ALL

CSS 3 Features

This section enumerates the CSS 3 features that are supported by Oxygen XML Editor plugin .

CSS 3 Namespace Selectors

In the CSS 2.1 standard the element selectors are ignoring the namespaces of the elements they are matching. Only the local name of the elements are considered in the selector matching process.

Oxygen XML Author uses a different approach similar to the CSS Level 3 specification. If the element name from the CSS selector is not preceded by a namespace prefix it is considered to match an element with the same local name as the selector value and ANY namespace, otherwise the element must match both the local name and the namespace.

In CSS up to version 2.1 the name tokens from selectors are matching all elements from ANY namespace that have the same local name. Example:

```
<x:b xmlns:x="ns_x"/>
<y:b xmlns:y="ns_y"/>
```

Are both matched by the rule:

```
b {font-weight:bold}
```

Starting with CSS Level 3 you can create selectors that are namespace aware.

Defining both prefixed namespaces and the default namespace

Given the namespace declarations:

```
@namespace sync "http://sync.example.org";
@namespace "http://example.com/foo";
```

In a context where the default namespace applies:

sync|A	represents the name A in the `http://sync.example.org` namespace.
|B	represents the name B that belongs to NO NAMESPACE.
***|C**	represents the name C in ANY namespace, including NO NAMESPACE.
D	represents the name D in the `http://example.com/foo` namespace.

Defining only prefixed namespaces

Given the namespace declaration:

```
@namespace sync "http://sync.example.org";
```

Then:

sync|A represents the name A in the `http://sync.example.org` namespace.

|B represents the name B that belongs to NO NAMESPACE.

***|C** represents the name C in ANY namespace, including NO NAMESPACE.

D represents the name D in ANY namespace, including NO NAMESPACE.

The `attr()` Function: Properties Values Collected from the Edited Document.

In CSS Level 2.1 you may collect attribute values and use them as content *only* for the pseudo-elements. For instance the :before pseudo-element can be used to insert some content before an element. This is valid in CSS 2.1:

```
title:before{
  content: "Title id=(" attr(id) ")";
}
```

If the `title` element from the XML document is:

```
<title id="title12">My title.</title>
```

Then the title will be displayed as:

```
Title id=(title12) My title.
```

In Oxygen XML Author the use of `attr()` function is available not only for the `content` property, but also for any other property. This is similar to the CSS Level 3 working draft: *http://www.w3.org/TR/2006/WD-css3-values-20060919/#functional*. The arguments of the function are:

`attr ( attribute_name , attribute_type , default_value )`

`attribute_name` The attribute name. This argument is required.

`attribute_type` The attribute type. This argument is optional. If it is missing, argument's type is considered `string`. This argument indicates what is the meaning of the attribute value and helps to perform conversions of this value. Oxygen XML Editor plugin Author accepts one of the following types:

- **`color`** The value represents a color. The attribute may specify a color in different formats. Oxygen XML Author supports colors specified either by name: `red`, `blue`, `green`, etc. or as an RGB hexadecimal value `#FFEEFF`.
- **`url`** The value is an URL pointing to a media object. Oxygen XML Editor plugin Author supports only images. The attribute value can be a complete URL, or a relative one to the XML document. Please note that this URL is also resolved through the catalog resolver.
- **`integer`** The value must be interpreted as an integer.
- **`number`** The value must be interpreted as a float number.
- **`length`** The value must be interpreted as an integer.

`percentage` The value must be interpreted relative to another value (length, size) expressed in percents.

`em` The value must be interpreted as a size. 1 em is equal to the *font-size* of the relevant font.

`ex` The value must be interpreted as a size. 1 ex is equal to the *height* of the **x** character of the relevant font.

`px` The value must be interpreted as a size expressed in pixels relative to the viewing device.

`mm` The value must be interpreted as a size expressed in millimeters.

`cm` The value must be interpreted as a size expressed in centimeters.

`in` The value must be interpreted as a size expressed in inches. 1 inch is equal to 2.54 centimeters.

`pt` The value must be interpreted as a size expressed in points. The points used by CSS2 are equal to 1/72th of an inch.

`pc` The value must be interpreted as a size expressed in picas. 1 pica is equal to 12 points.

`default_value` This argument specifies a value that is used by default if the attribute value is missing. This argument is optional.

Usage samples for the attr() function

Consider the following XML instance:

```
<sample>
    <para bg_color="#AAAAFF">Blue paragraph.</para>
    <para bg_color="red">Red paragraph.</para>
    <para bg_color="red" font_size="2">Red paragraph with large
font.</para>
    <para bg_color="#00AA00" font_size="0.8" space="4">
        Green paragraph with small font and margin.</para>
</sample>
```

The `para` elements have `bg_color` attributes with RGB color values like `#AAAAFF`. You can use the `attr()` function to change the elements appearance in the editor based on the value of this attribute:

```
background-color:attr(bg_color, color);
```

The attribute `font_size` represents the font size in *em* units. You can use this value to change the style of the element:

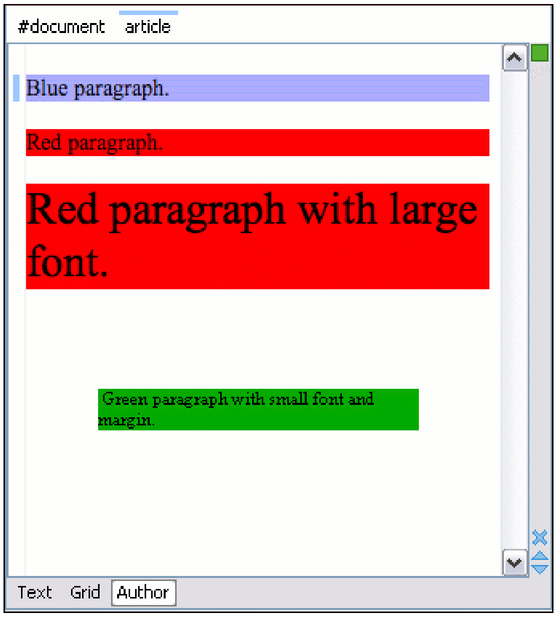
```
font-size:attr(font_size, em);
```

The complete CSS rule is:

```
para{
 display:block;
 background-color:attr(bg_color, color);
 font-size:attr(font_size, em);
```

```
  margin:attr(space, em);
 }
```

The document is rendered as:



Blue paragraph.

Red paragraph.

Red paragraph with large font.

Green paragraph with small font and margin.

Styling Elements from other Namespace

In the CSS Level 1, 2, and 2.1 there is no way to specify if an element X from the namespace Y should be presented differently from the element X from the namespace Z. In the upcoming CSS Level 3, it is possible to differentiate elements by their namespaces. Oxygen XML Editor plugin Author supports this CSS Level 3 functionality. For more information see the *Namespace Selectors* section.

To match the `def` element its namespace will be declared, bind it to the *abs* prefix, and then write a CSS rule:

```
@namespace abs "http://www.oxygenxml.com/sample/documentation/abstracts";

abs|def{
    font-family:monospace;
    font-size:smaller;
}
abs|def:before{
    content:"Definition:";
    color:gray;
}
```

Additional Custom Selectors

Oxygen XML Author provides support for selecting additional types of nodes. These custom selectors apply to: *document*, *doctype sections*, *processing-instructions*, *comments*, *CDATA sections*, *reference sections*, and *entities*.

Note: The custom selectors are presented in the default CSS for Author mode and all of their properties are marked with an *!important* flag. For this reason, you have to set the *!important* flag on each property of the custom selectors from your CSS to be applicable.

In order for the custom selectors to work in your CSSs, declare the Author extensions namespace at the beginning of the stylesheet documents:

```
@namespace oxy url('http://www.oxygenxml.com/extensions/author');
```

- The *oxy|document* selector matches the entire document:

  ```
  oxy|document {
      display:block !important;
  }
  ```

- The following example changes the rendering of doctype sections:

  ```
  oxy|doctype {
      display:block !important;
      color:blue !important;
      background-color:transparent !important;
  }
  ```

- To match the processing instructions, you can use the *oxy|processing-instruction* selector:

  ```
  oxy|processing-instruction {
      display:block !important;
      color:purple !important;
      background-color:transparent !important;
  }
  ```

- The XML comments display in Author mode can be changed using the *oxy|comment* selector:

  ```
  oxy|comment {
      display:block !important;
      color:green !important;
      background-color:transparent !important;
  }
  ```

- The *oxy|cdata* selector matches CDATA sections:

  ```
  oxy|cdata{
      display:block !important;
      color:gray !important;
      background-color:transparent !important;
  }
  ```

- The *oxy|entity* selector matches the entities content:

  ```
  oxy|entity {
      display:morph !important;
      editable:false !important;
      color:orange !important;
      background-color:transparent !important;
  }
  ```

- The *references to entities*, *XInclude*, and *DITA conrefs* are expanded by default in Author mode and *the referred content is displayed.* The referred resources are loaded and displayed inside the element or entity that refers them.

- You can use the *reference* property to customize the way these references are rendered in Author mode:

  ```
  oxy|reference {
    border:1px solid gray !important;
  }
  ```

In Author mode, content is highlighted when parts of text contain:

- *comments;*
- changes and *Track Changes* was active when the content was modified.

If this content is referred, the Author mode does not display the highlighted areas in the new context. If you want to mark the existence of this comments and changes you can use the *oxy|reference[comments]*, *oxy|reference[changeTracking]*, and *oxy|reference[changeTracking][comments]* selectors.

Note: Two artificial attributes (*comments* and *changeTracking*) are set on the reference node, containing information about the number of comments and track changes in the content.

- The following example represents the customization of the reference fragments that contain comments:

  ```
  oxy|reference[comments]:before {
    content: "Comments: " attr(comments) !important;
  }
  ```

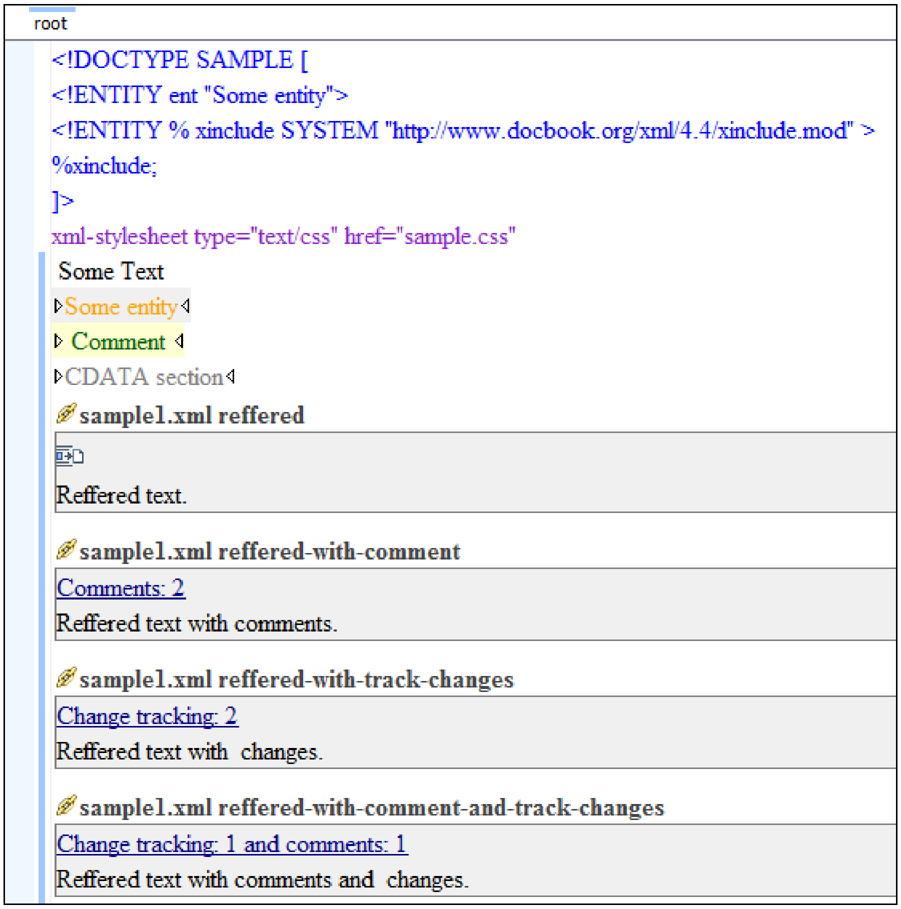
- To match reference fragments based on the fact that they contain change tracking inside, use the *oxy|reference[changeTracking]* selector.

  ```
  oxy|reference[changeTracking]:before {
    content: "Change tracking: " attr(changeTracking) !important;
  }
  ```

- Here is an example of how you can set a custom color to the reference containing both track changes and comments:

  ```
  oxy|reference[changeTracking][comments]:before {
    content: "Change tracking: " attr(changeTracking) " and comments: "
  attr(comments) !important;
  }
  ```

A sample document rendered using these rules:



root

<!DOCTYPE SAMPLE [
<!ENTITY ent "Some entity">
<!ENTITY % xinclude SYSTEM "http://www.docbook.org/xml/4.4/xinclude.mod" >
%xinclude;
]>
xml-stylesheet type="text/css" href="sample.css"

Some Text

Some entity

Comment

CDATA section

sample1.xml reffered

Reffered text.

sample1.xml reffered-with-comment

Comments: 2

Reffered text with comments.

sample1.xml reffered-with-track-changes

Change tracking: 2

Reffered text with changes.

sample1.xml reffered-with-comment-and-track-changes

Change tracking: 1 and comments: 1

Reffered text with comments and changes.

Oxygen CSS Extensions

CSS stylesheets provide support mainly for displaying documents. When editing documents some extensions of the W3C CSS specification are useful, for example:

- property for marking foldable elements in large files
- enforcing a display mode for the XML tags regardless of the current mode selected by the author user
- construct a URL from a relative path location
- string processing functions

Media Type oxygen

The style sheets can specify how a document is to be presented on different media: on the screen, on paper, speech synthesizer, etc. You can specify that some of the features of your CSS stylesheet should be taken into account only in the Oxygen XML Editor plugin Author and ignored in the rest. This can be accomplished by using the media type oxygen.

For instance using the following CSS:

```
b{
 font-weight:bold;
 display:inline;
}

@media oxygen{
 b{
  text-decoration:underline;
 }
}
```

would make a text bold if the document was opened in a web browser that does not recognize `@media oxygen` and bold and underlined in Oxygen XML Editor plugin Author.

You can use this media type to group specific Oxygen XML Editor plugin CSS features and also to hide them when opening the documents with other viewers.

Folding Elements: `-oxy-foldable`, `-oxy-not-foldable-child` and `-oxy-folded` properties

Oxygen XML Author allows you to declare some elements to be *foldable* (collapsible). This is especially useful when working with large documents organized in logical blocks, editing a large DocBook article or book for instance. Oxygen XML Editor plugin marks the foldable content with a small blue triangle. When you hover with your mouse pointer over this marker, a dotted line borders the collapsible content. The following contextual actions are available:

- (Ctrl+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.

To define the element whose content can be folded by the user, you must use the property: `-oxy-foldable:true;`. To define the elements that are folded by default, use the `-oxy-folded:true` property.

Note: The `-oxy-folded` property works in conjunction with the `-oxy-foldable` property. Thus, the `folded` property is ignored if the `-oxy-foldable` property is not set on the same element.

When collapsing an element, it is useful to keep some of its content visible, like a short description of the collapsed region. The property `-oxy-not-foldable-child` is used to identify the child elements that are kept visible. It accepts as value an element name or a list of comma separated element names. If the element is marked as *foldable* (`-oxy-foldable:true;`) but it doesn't have the property `-oxy-not-foldable-child` or none of the specified non-foldable children exists, then the element is still foldable. In this case the element kept visible when folded will be the `before` pseudo-element.

Note: Deprecated properties `foldable`, `not-foldable-child`, and `folded` are also supported.

Folding DocBook Elements

All the elements below can have a `title` child element and are considered to be logical sections. You mark them as being foldable leaving the `title` element visible.

```
set,
book,
part,
reference,
chapter,
preface,
article,
sect1,
sect2,
sect3,
sect4,
section,
appendix,
figure,
example,
table {
    foldable:true;
    not-foldable-child: title;
}
```

Placeholders for empty elements: `-oxy-show-placeholder` and `-oxy-placeholder-content` properties

Oxygen XML Author displays the element name as pseudo-content for empty elements, if the *Show placeholders for empty elements* option is enabled and there is no before or after content set is CSS for this type of element.

To control the displayed pseudo-content for empty elements, you can use the `-oxy-placeholder-content` CSS property.

The `-oxy-show-placeholder` property allows you to decide whether the placeholder must be shown. The possible values are:

- `always` - Always display placeholders.
- `default` - Always display placeholders if before or after content are not set is CSS.
- `inherit` - The placeholders are displayed according to **Show placeholders for empty elements** option (if `before` and `after` content is not declared).

Note: Deprecated properties `show-placeholder` and `placeholder-content` are also supported.

Read-only elements: `-oxy-editable` property

If you want to inhibit editing a certain element content, you can set the `-oxy-editable` (deprecated property `editable` is also supported) CSS property to `false`.

Display Elements: `-oxy-morph` value

Oxygen XML Author allows you to specify that an element has a `-oxy-morph` display type (deprecated `morph` property is also supported), meaning that the element is inline if all its children are inline.

`whitespace` property: `-oxy-trim-when-ws-only` value

Oxygen XML Author allows you to set the `whitespace` property to `-oxy-trim-when-ws-only`, meaning that the leading and trailing whitespaces are removed.

Link Elements

Oxygen XML Author allows you to declare some elements to be *links*. This is especially useful when working with many documents which refer each other. The links allow for an easy way to get from one document to another. Clicking on the link marker will open the referred resource in an editor.

To define the element which should be considered a link, you must use the property `link` on the before or after pseudo element. The value of the property indicates the location of the linked resource. Since links are usually indicated by the value of an attribute in most cases it will have a value similar to `attr(href)`

Docbook Link Elements

All the elements below are defined to be links on the before pseudo element and their value is defined by the value of an attribute.

```
*[href]:before{
    link:attr(href);
    content: "Click " attr(href) " for opening" ;
}
```

```
ulink[url]:before{
    link:attr(url);
    content: "Click to open: " attr(url);
}

olink[targetdoc]:before{
    link: attr(targetdoc);
    content: "Click to open: " attr(targetdoc);
}
```

Display Tag Markers

Oxygen XML Author allows you to choose whether tag markers of an element should never be presented or the current display mode should be respected. This is especially useful when working with `:before` and `:after` pseudo-elements in which case the element range is already visually defined so the tag markers are redundant.

The property is named `-oxy-display-tags`, with the following possible values:

- *none* - Tags markers must not be presented regardless of the current *Display mode.*
- *default* - The tag markers will be created depending on the current *Display mode.*
- *inherit* - The value of the property is inherited from an ancestor element.

```
display-tags
    Value: none | default | inherit
    Initial: default
    Applies to: all nodes(comments, elements, CDATA, etc)
    Inherited: false
    Media: all
```

Docbook Para elements

In this example the **para** element from Docbook is using an `:before` and `:after` element so you don't want its tag markers to be visible.

```
para:before{
    content: "{";
}

para:after{
    content: "}";
}

para{
    display-tags: none;
    display:block;
    margin: 0.5em 0;
}
```

Custom CSS Functions

In Oxygen XML Author there are implemented a few Oxygen XML Editor plugin specific custom CSS functions. Imbricated custom functions are also supported.

> The result of the functions below will be the local name of the current node with the first letter capitalized.
>
> `oxy_capitalize(oxy_local-name())`

The `oxy_local-name()` Function

This function evaluates the local name of the current node. It does not have any arguments.

The `oxy_name()` Function

This function evaluates the qualified name of the current node. It does not have any arguments.

The `oxy_url()` function

This function extends the standard CSS **oxy_url()** function, by allowing you to specify additional relative path components (parameters **loc_1** to **loc_n**). Oxygen XML Editor plugin uses all these parameters to construct an absolute location.

`oxy_url ( location , loc_1 , loc_2 )`

location	The location as string. If not absolute, will be solved relative to the CSS file URL.
loc_1 ... loc_n	Relative location path components as string. (optional)

> The following function:
>
> ```
> url('http://www.oxygenxml.com/css/test.css', '../dir1/',
> 'dir2/dir3/', '../../dir4/dir5/test.xml')
> ```
>
> returns
>
> ```
> 'http://www.oxygenxml.com/dir1/dir4/dir5/test.xml'
> ```

The `oxy_base-uri()` Function

This function evaluates the base URL in the context of the current node. It does not have any arguments and takes into account the `xml:base` context of the current node. See the *XML Base specification* for more details.

The `oxy_parent-url()` Function

This function evaluates the parent URL of an URL received as string.

`oxy_parent-url ( URL )`

URL	The URL as string.

The `oxy_capitalize()` Function

This function capitalizes the first letter of the text received as argument.

`oxy_capitalize ( text )`

text	The text for which the first letter will be capitalized.

The `oxy_uppercase()` Function

This function transforms to upper case the text received as argument.

`oxy_uppercase ( text )`

text	The text to be capitalized.

The `oxy_lowercase()` Function

This function transforms to lower case the text received as argument.

`oxy_lowercase ( text )`

text	The text to be lower cased.

The `oxy_concat()` Function

This function concatenates the received string arguments.

`oxy_concat ( str_1 , str_2 )`

str_1 ... str_n	The string arguments to be concatenated.

The `oxy_replace()` Function

This function has two signatures:

- `oxy_replace ( text , target , replacement )`

  This function replaces each substring of the text that matches the literal target string with the specified literal replacement string.

text	The text in which the replace will occur.
target	The target string to be replaced.
replacement	The string replacement.

- `oxy_replace ( text , target , replacement , isRegExp )`

  This function replaces each substring of the text that matches the target string with the specified replacement string.

text	The text in which the replace will occur.
target	The target string to be replaced.
replacement	The string replacement.
isRegExp	If *true* the target and replacement arguments are considered regular expressions in PERL syntax, if *false* they are considered literal strings.

The `oxy_unparsed-entity-uri()` Function

This function returns the URI value of an unparsed entity name.

`oxy_unparsed-entity-uri ( unparsedEntityName )`

unparsedEntityName	The name of an unparsed entity defined in the DTD.

This function can be useful to display images which are referred with unparsed entity names.

CSS for displaying the image in Author for an *`imagedata`* with *`entityref`* to an unparsed entity

```
imagedata[entityref]{
content: oxy_url(oxy_unparsed-entity-uri(attr(entityref)));
}
```

The `oxy_attributes()` Function

This function concatenates the attributes for an element and returns the serialization.

`oxy_attributes ( )`

oxy_attributes()

For the following XML fragment:`<element att1="x" xmlns:a="2" x="&quot;"/>` the `oxy_attributes()` function will return `att1="x" xmlns:a="2" x="""`.

The `oxy_substring()` Function

This function has two signatures:

- `oxy_substring ( text , startOffset )`

  Returns a new string that is a substring of the original **text** string. It begins with the character at the specified index and extends to the end of **text** string.

text	The original string.
startOffset	The beginning index, inclusive

- `substring ( text , startOffset , endOffset )`

  Returns a new string that is a substring of the original **text** string. The substring begins at the specified **startOffset** and extends to the character at index **endOffset** - 1.

text	The original string.
startOffset	The beginning index, inclusive
endOffset	The ending index, exclusive.

`oxy_substring('abcd', 1)` returns the string `'bcd'`.

`oxy_substring('abcd', 4)` returns an empty string.

`oxy_substring('abcd', 1, 3)` returns the string `'bc'`.

The `oxy_indexof()` Function

This function has two signatures:

- `oxy_indexof ( text , toFind )`

  Returns the index within **text** string of the first occurrence of the **toFind** substring.

text	Text to search in.
toFind	The searched substring.

- `oxy_indexof ( text , toFind , fromOffset )`

  Returns the index within **text** string of the first occurrence of the **toFind** substring. The search starts from **fromOffset** index.

text	Text to search in.
toFind	The searched substring.
fromOffset	The index from which to start the search.

`oxy_indexof('abcd', 'bc')` returns 1.

`oxy_indexof('abcdbc', 'bc', 2)` returns 4.

The `oxy_lastindexof()` Function

This function has two signatures:

- `oxy_lastindexof ( text , toFind )`

  Returns the index within **text** string of the rightmost occurrence of the **toFind** substring.

text	Text to search in.

toFind	The searched substring.

- `oxy_lastindexof ( text , toFind , fromOffset )`

  The search starts from **fromOffset** index. Returns the index within **text** string of the last occurrence of the **toFind** substring, searching backwards starting from the **fromOffset** index.

text	Text to search in.
toFind	The searched substring.
fromOffset	The index from which to start the search backwards.

`oxy_lastindexof('abcdbc', 'bc')` returns 4.

`oxy_lastindexof('abcdbccdbc', 'bc', 2)` returns 1.

The `oxy_xpath()` Function

This function has one signature:

- `oxy_xpath ( expression )`

  Evaluates the given XPath expression and returns the result.

  Note: The entities are ignored when the XPath expressions are evaluated.

expression	XPath expression to be evaluated.

The following example counts the number of words from a paragraph and displays the result in front of it:

```
para:before{ content: concat("|Number of words:",
oxy_xpath("count(tokenize(normalize-space(string-join(text(), '')),
' '))"), "| "); }
```

Arithmetic Functions

You can use any of the arithmetic functions implemented in the `java.lang.Math` class: *http://download.oracle.com/javase/6/docs/api/java/lang/Math.html*.

In addition to that, the following functions are available:

Syntax	Details
oxy_add(`param1`, ... , `paramN`, `'returnType'`)	Adds the values of all parameters from `param1` to `paramN`.
oxy_subtract(`param1`, ..., `paramN`, `'returnType'`)	Subtracts the values of parameters `param2` to `paramN` from `param1`.
oxy_multiply(`param1`, ..., `paramN`, `'returnType'`)	Multiplies the values of parameters from `param1` to `paramN`.
oxy_divide(`param1`, `param2`, `'returnType'`)	Performs the division of `param1` to `param2`.
oxy_modulo(`param1`, `param2`, `'returnType'`)	Returns the reminder of the division of `param1` to `param2`.

Note: The `returnType` can be `'integer'`, `'number'`, or any of the supported CSS measuring types.

Example Files Listings - The Simple Documentation Framework Files

This section lists the files used in the customization tutorials: the XML Schema, CSS files, XML files, XSLT stylesheets.

XML Schema files

`sdf.xsd`

This sample file can also be found in the *Author SDK distribution* in the `"oxygenAuthorSDK\samples\Simple Documentation Framework - SDF\framework\schema"` directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.oxygenxml.com/sample/documentation"
    xmlns:doc="http://www.oxygenxml.com/sample/documentation"
    xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts"
    elementFormDefault="qualified">

    <xs:import
        namespace="http://www.oxygenxml.com/sample/documentation/abstracts"
        schemaLocation="abs.xsd"/>

    <xs:element name="book" type="doc:sectionType"/>
    <xs:element name="article" type="doc:sectionType"/>
    <xs:element name="section" type="doc:sectionType"/>

    <xs:complexType name="sectionType">
        <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element ref="abs:def" minOccurs="0"/>
            <xs:choice>
                <xs:sequence>
                    <xs:element ref="doc:section"
                        maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:choice maxOccurs="unbounded">
                    <xs:element ref="doc:para"/>
                    <xs:element ref="doc:ref"/>
                    <xs:element ref="doc:image"/>
                    <xs:element ref="doc:table"/>
                </xs:choice>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="para" type="doc:paragraphType"/>

    <xs:complexType name="paragraphType" mixed="true">
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="b"/>
            <xs:element name="i"/>
            <xs:element name="link"/>
        </xs:choice>
    </xs:complexType>

    <xs:element name="ref">
        <xs:complexType>
            <xs:attribute name="location" type="xs:anyURI"
                use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="image">
        <xs:complexType>
            <xs:attribute name="href" type="xs:anyURI"
                use="required"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="table">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="customcol" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:attribute name="width" type="xs:string"/>
                    </xs:complexType>
                </xs:element>
                <xs:element name="header">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="td"
                                maxOccurs="unbounded"
                                type="doc:paragraphType"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="tr" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="td"
                                type="doc:tdType"
                                maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="width" type="xs:string"/>
        </xs:complexType>
    </xs:element>


    <xs:complexType name="tdType">
        <xs:complexContent>
            <xs:extension base="doc:paragraphType">
                <xs:attribute name="row_span"
                    type="xs:integer"/>
                <xs:attribute name="column_span"
                    type="xs:integer"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

`abs.xsd`

This sample file can also be found in the *Author SDK distribution* in the `"oxygenAuthorSDK\samples\Simple Documentation Framework - SDF\framework\schema"` directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=
 "http://www.oxygenxml.com/sample/documentation/abstracts">
```

```
    <xs:element name="def" type="xs:string"/>
</xs:schema>
```

CSS Files

sdf.css

This sample file can also be found in the *Author SDK distribution* in the `"oxygenAuthorSDK\samples\Simple Documentation Framework - SDF\framework\css"` directory.

```
/* Element from another namespace */
@namespace abs "http://www.oxygenxml.com/sample/documentation/abstracts";

abs|def{
    font-family:monospace;
    font-size:smaller;
}
abs|def:before{
    content:"Definition:";
    color:gray;
}

/* Vertical flow */
book,
section,
para,
title,
image,
ref {
    display:block;
}

/* Horizontal flow */
b,i {
    display:inline;
}

section{
    margin-left:1em;
    margin-top:1em;
}

section{
    foldable:true;
    not-foldable-child: title;
}

link[href]:before{
    display:inline;
    link:attr(href);
    content: "Click to open: " attr(href);
}

/* Title rendering*/
title{
    font-size: 2.4em;
    font-weight:bold;
}

* * title{
    font-size: 2.0em;
```

```
}
* * * title{
    font-size: 1.6em;
}
* * * * title{
    font-size: 1.2em;
}

book,
article{
    counter-reset:sect;
}
book > section,
article > section{
    counter-increment:sect;
}
book > section > title:before,
article > section > title:before{
    content: "Section: " counter(sect) " ";
}

/* Inlines rendering*/
b {
    font-weight:bold;
}

i {
    font-style:italic;
}

/*Table rendering */
table{
    display:table;
    border:1px solid navy;
    margin:1em;
    max-width:1000px;
    min-width:150px;
}

table[width]{
  width:attr(width, length);
}

tr, header{
    display:table-row;
}

header{
    background-color: silver;
    color:inherit
}

td{
  display:table-cell;
  border:1px solid navy;
  padding:1em;
}

image{
    display:block;
    content: attr(href, url);
    margin-left:2em;
}
```

XML Files

sdf_sample.xml

This sample file can also be found in the *Author SDK distribution* in the "oxygenAuthorSDK\samples\Simple Documentation Framework - SDF\framework" directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://www.oxygenxml.com/sample/documentation"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">
    <title>My Technical Book</title>
    <section>
        <title>XML</title>
        <abs:def>Extensible Markup Language</abs:def>
        <para>In this section of the book I will explain
            different XML applications.</para>
    </section>
    <section>
        <title>Accessing XML data.</title>
        <section>
            <title>XSLT</title>
            <abs:def>Extensible stylesheet language
                transformation (XSLT) is a language for
                transforming XML documents into other XML
                documents.</abs:def>
            <para>A list of XSL elements and what they do..</para>
            <table>
                <header>
                    <td>XSLT Elements</td>
                    <td>Description</td>
                </header>
                <tr>
                    <td>
                        <b>xsl:stylesheet</b>
                    </td>
                    <td>The <i>xsl:stylesheet</i> element is
                        always the top-level element of an
                        XSL stylesheet. The name
                            <i>xsl:transform</i> may be used
                        as a synonym.</td>
                </tr>
                <tr>
                    <td>
                        <b>xsl:template</b>
                    </td>
                    <td>The <i>xsl:template</i> element has
                        an optional mode attribute. If this
                        is present, the template will only
                        be matched when the same mode is
                        used in the invoking
                            <i>xsl:apply-templates</i>
                        element.</td>
                </tr>
                <tr>
                    <td>
                        <b>for-each</b>
                    </td>
                    <td>The xsl:for-each element causes
                        iteration over the nodes selected by
                        a node-set expression.</td>
                </tr>
                <tr>
                    <td column_span="2">End of the list</td>
                </tr>
            </table>
        </section>
        <section>
            <title>XPath</title>
            <abs:def>XPath (XML Path Language) is a terse
                (non-XML) syntax for addressing portions of
                an XML document. </abs:def>
            <para>Some of the XPath functions.</para>
            <table>
                <header>
                    <td>Function</td>
                    <td>Description</td>
                </header>
                <tr>
                    <td>format-number</td>
                    <td>The <i>format-number</i> function
                        converts its first argument to a
                        string using the format pattern
                        string specified by the second
                        argument and the decimal-format
                        named by the third argument, or the
                        default decimal-format, if there is
                        no third argument</td>
                </tr>
                <tr>
                    <td>current</td>
                    <td>The <i>current</i> function returns
                        a node-set that has the current node
                        as its only member.</td>
                </tr>
                <tr>
                    <td>generate-id</td>
                    <td>The <i>generate-id</i> function
                        returns a string that uniquely
                        identifies the node in the argument
                        node-set that is first in document
                        order.</td>
                </tr>
            </table>
        </section>
    </section>
    <section>
        <title>Documentation frameworks</title>
        <para>One of the most important documentation
            frameworks is Docbook.</para>
        <image
            href="http://www.xmlhack.com/images/docbook.png"/>
        <para>The other is the topic oriented DITA, promoted
            by OASIS.</para>
        <image
href="http://www.oasis-open.org/images/standards/oasis_standard.jpg"
        />
    </section>
</book>
```

XSL Files

`sdf.xsl`

This sample file can also be found in the *Author SDK distribution* in the `"oxygenAuthorSDK\samples\Simple Documentation Framework - SDF\framework\xsl"` directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
    xpath-default-namespace=
    "http://www.oxygenxml.com/sample/documentation">

    <xsl:template match="/">
        <html><xsl:apply-templates/></html>
    </xsl:template>

    <xsl:template match="section">
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="image">
        <img src="{@href}"/>
    </xsl:template>

    <xsl:template match="para">
        <p>
            <xsl:apply-templates/>
        </p>
    </xsl:template>

    <xsl:template match="abs:def"
        xmlns:abs=
        "http://www.oxygenxml.com/sample/documentation/abstracts">
        <p>
            <u><xsl:apply-templates/></u>
        </p>
    </xsl:template>

    <xsl:template match="title">
        <h1><xsl:apply-templates/></h1>
    </xsl:template>

    <xsl:template match="b">
        <b><xsl:apply-templates/></b>
    </xsl:template>

    <xsl:template match="i">
        <i><xsl:apply-templates/></i>
    </xsl:template>

    <xsl:template match="table">
        <table frame="box" border="1px">
            <xsl:apply-templates/>
        </table>
    </xsl:template>

    <xsl:template match="header">
        <tr>
            <xsl:apply-templates/>
        </tr>
    </xsl:template>
    <xsl:template match="tr">
        <tr>
            <xsl:apply-templates/>
        </tr>
    </xsl:template>

    <xsl:template match="td">
        <td>
            <xsl:apply-templates/>
        </td>
    </xsl:template>

    <xsl:template match="header/header/td">
        <th>
            <xsl:apply-templates/>
        </th>
    </xsl:template>

</xsl:stylesheet>
```

Author Component

The Author Component was designed as a separate product to provide the functionality of the standard Author mode. The component can be embedded either in a third-party standalone Java application or customized as a Java Web Applet to provide WYSIWYG-like XML editing directly in your web browser of choice.

The Author Component Startup Project for Java/Swing integrations is available online on the oXygen XML website: *http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-sample.zip*

Licensing

Just like the oXygen standard deployment, the Author component requires license information in order to run. Licensing information must follow the same models imposed for the standard oXygen application, namely the floating, named-user based or group licenses.

You can configure the component to:

- Inject floating license server details in the Java code. You can license an Author component using standard oXygen XML Editor/Author floating license keys or special Author Component license keys. This is the recommended method. Here are details to configure a floating license servlet or server:*http://www.oxygenxml.com/license_server.html*.
- Inject the licensing information key directly in the component's Java code. This method **MUST** be used only when you have a site license or a special evaluation license (for development purposes) obtained by contacting our technical support email address (support@oxygenxml.com).

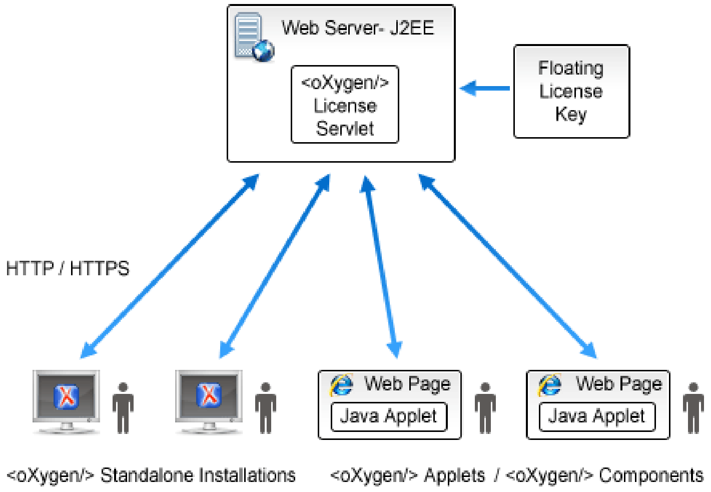
  ⚠ **Warning:** When the license key is injected set in the Author Component, only site and evaluation license keys can be used to license the component. Attempts to use other types of license will fail to initialize the component.

- Display the license registration dialog to the end user. This is the default behavior and transfers the licensing responsibility to the end-user. The standard licensing procedure applies. You can license an Author component using standard oXygen XML Editor/Author license keys or special Author Component license keys. The license key will be saved to the local user's disk and on subsequent runs the user will not be asked anymore.

The most common use-case is when you as a developer customize the component and then want to deliver it to end users (either embedded in a Java application or a Java Web applet). Your licensing options are:

- named-user based model, where users provide their own oXygen license keys and register the component (paste it in the license registration dialog);

- floating license model, where the component comes pre-configured to use one of the oXygen floating license servers (either the standalone or the servlet (recommended) version).


Web Server- J2EE
<oXygen/> License Servlet
Floating License Key
HTTP / HTTPS
Web Page
Java Applet
Web Page
Java Applet
<oXygen/> Standalone Installations
<oXygen/> Applets / <oXygen/> Components

Installation Requirements

Running the Author component as a Java applet requires:

- Oracle (Sun) Java JRE version 1.6 update 10 or newer;
- At least 100 MB disk space and 100MB free memory;
- The applet needs to be signed with a valid certificate and will request full access to the user machine, in order to store customization data (like options and framework files);
- A table of supported browsers can be found here:*Supported browsers and operating systems* on page 360.

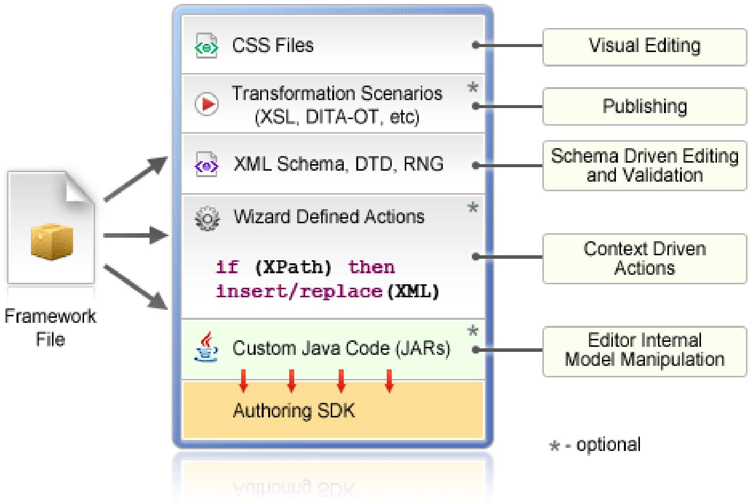
Running the Author component embedded in a third-party Java/Swing application requires:

- Oracle (Sun) Java JRE version 1.6 or newer;
- At least 100 MB disk space and 100MB free memory;

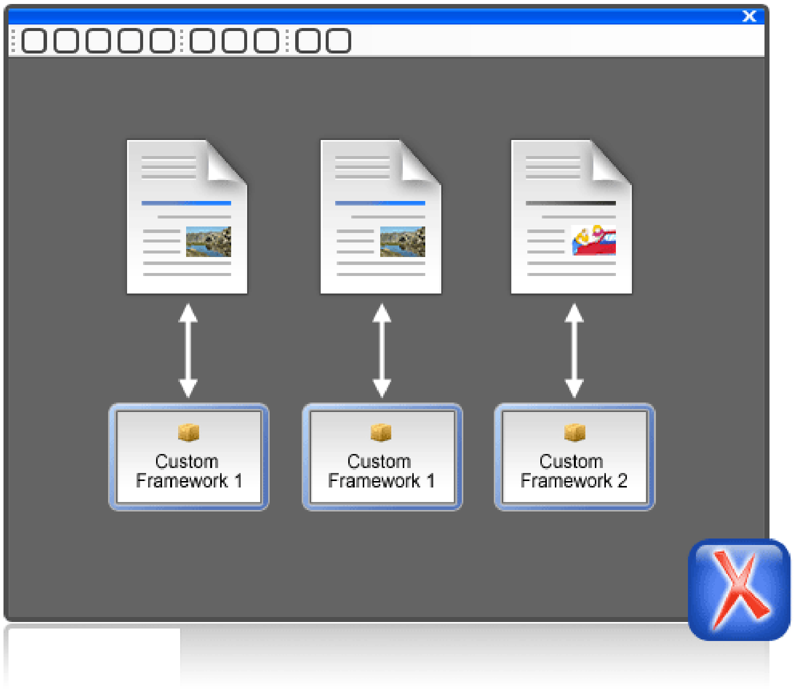
Customization

For a special type of XML, you can create a custom framework (which also works in an Oxygen standalone version). Oxygen XML Editor plugin already has frameworks for editing DocBook, DITA, TEI, and so on. Their sources are available in *the Author SDK*. This custom framework is then packed in a zip archive and used to deploy the component.

The following diagram shows the components of a custom framework.


Framework File
CSS Files
Transformation Scenarios * (XSL, DITA-OT, etc)
XML Schema, DTD, RNG
Wizard Defined Actions *
if (XPath) then insert/replace (XML)
Custom Java Code (JARs) *
Authoring SDK
Visual Editing
Publishing
Schema Driven Editing and Validation
Context Driven Actions
Editor Internal Model Manipulation
* - optional

More than one framework can coexist in the same component and can be used at the same time for editing XML documents.



You can add on your custom toolbar all actions available in the standalone Oxygen XML Editor plugin application for editing in the Author mode. You can also add custom actions defined in the framework customized for each XML type.

The Author component can also provide the *Outline*, *Model*, *Elements* and *Attributes* views which can be added to your own developed containers.

Deployment

The Author Component Java API allows you to use it in your Java application or as a Java applet. The JavaDoc for the API can be found in the *sample project* in the `lib/apiSrc.zip` archive. The sample project also comes with Java sources (`ro/sync/ecss/samples/AuthorComponentSample.java`) demonstrating how the component is created, licensed and used in a Java application.

Web Deployment

The Author Component can be deployed as a Java Applet using the new Applet with JNLP Java technology, available in Oracle (Sun) Java JRE version 1.6 update 10 or newer.

The *sample project* demonstrates how the Author component can be distributed as an applet.

Here are the main steps you need to follow in order to deploy the Author component as a Java Applet:

- Unpack the sample project archive and look for Java sources of the sample Applet implementation. They can be customized to fit your requirements.
- The `default.properties` configuration file must first be edited to specify your custom certificate information used to sign the applet libraries. You also have to specify the code base from where the applet will be downloaded.
- You can look inside the `author-component-dita.html` and `author-component-dita.js` sample Web resources to see how the applet is embedded in the page and how it can be controlled using Javascript (to set and get XML content from it).
- The sample Applet `author-component-dita.jnlp` JNLP file can be edited to add more libraries. The packed frameworks and options are delivered using the JNLP file as JAR archives:

```
<jar href="resources/frameworks.zip.jar"/>
<jar href="resources/options.zip.jar"/>
```

- The sample frameworks and options JAR archives can be found in the `resources` directory.
- Use the `build.xml` ANT build file to pack the component. The resulting applet distribution is copied in the `dist` directory. From this on, you can copy the applet files on your web server.

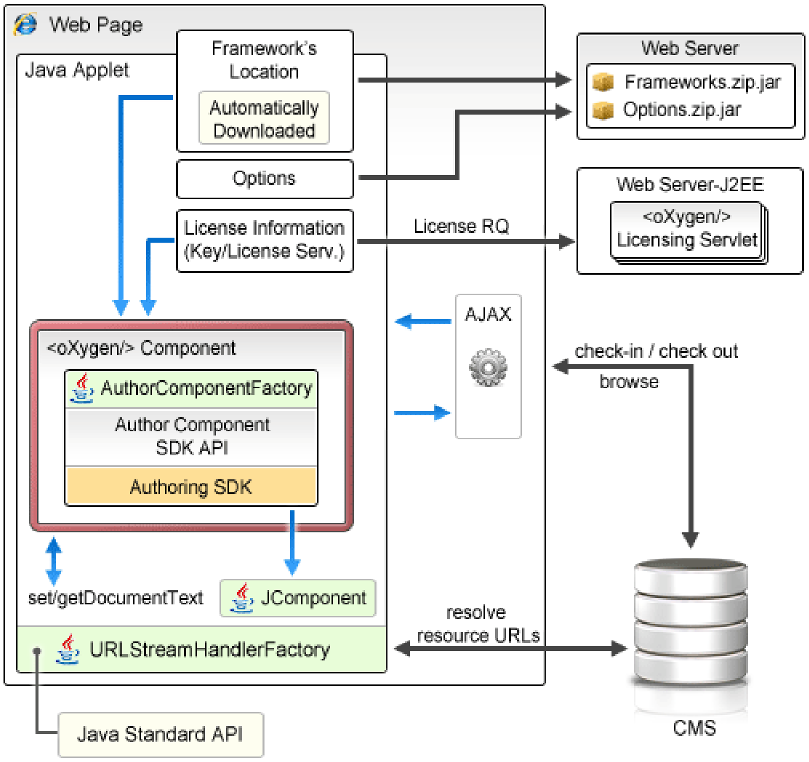


Figure 190: Oxygen XML Author Component deployed as a Java applet

Generate a Testing Certificate For Signing an Applet

All jar files of an applet deployed on a remote Web server must be signed with the same certificate before the applet is deployed. The following steps describe how to generate a test certificate for signing the jar files. We will use the tool called **keytool** which is included in the Oracle's Java Development Kit.

1. Create a keystore with a RSA encryption key.

   Invoke the following in a command line terminal:

   ```
   keytool -genkey -alias myAlias -keystore keystore.pkcs -storetype PKCS12
   -keyalg RSA -keysize 2048 -dname "cn=your name here, ou=organization unit
   name,  o=organization name, c=US"
   ```

   This command creates a keystore file called `keystore.pkcs`. The certificate attributes are specified in the *dname* parameter: common name of the certificate, organization unit name (for example *Purchasing* or *Sales Department*), organization name, country.

2. Generate a self-signed certificate.

   Invoke the following in a command line terminal:

   ```
   keytool -selfcert -alias myAlias -keystore keystore.pkcs -storetype PKCS12
   ```

3. Optionally display the certificate details in a human readable form.

   First, the certificate must be exported to a separate file with the following command:

   ```
   keytool -export -alias myAlias -keystore keystore.pkcs -storetype PKCS12 -file
    certfile.cer
   ```

The certificate details are displayed with the command:

```
keytool -printcert -file certfile.cer
```

4. Edit the `default.properties` file and fill-in the parameters that hold the path to `keystore.pkcs` file (`keystore` parameter), keystore type (`storetype` parameter, with `JSK` or `PKCS12` as possible values), alias (`alias` parameter) and password (`password` parameter).
5. Sign the jar files using the certificate by running the `sign` Ant task available in *the applet project*.

Supported browsers and operating systems

The applet was tested for compatibility with the following browsers:

	IE 7	IE 8 (32bit)	IE 9 (64bit)	Firefox 3	Firefox 4	Firefox 6	Safari 5	Chrome	Opera
Windows XP	Passed	Passed	-	Passed	Passed	Passed	-	Passed	Passed
Vista	Failed	Passed	Failed	Passed	Passed	Passed	Failed	Passed	Passed
Windows 7	-	-	Passed	Passed	Passed	Passed	-	Passed	Passed
Mac OS X 10.6	-	-	-	Failed	Passed	Passed	Passed	Failed	Passed
Mac OS X Lion	-	-	-	Failed	Passed	Passed	Passed	Failed	Passed
Linux Ubuntu 10	-	-	-	Failed	-	Passed	-	Failed	Passed

Troubleshooting

When the applet fails to start:

1. Make sure that your web browser really runs the next generation Java plug-in and not the legacy Java plug-in.

   For Windows and Mac OSX the procedure is straight forward. Some steps are given below for installing the Java plug-in on Linux:

   Installing the Java plugin in Ubuntu:

   ```
   sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"
   sudo apt-get update
   sudo apt-get install sun-java6-jdk
   sudo apt-get install sun-java6-plugin
   ```

   Manual Installation and Registration of Java Plugin for Linux: *http://www.oracle.com/technetwork/java/javase/manual-plugin-install-linux-136395.html*

2. Refresh the web page.
3. Remove the Java Webstart cache from the local drive and try again.

   - On Windows this folder is located in: `%APPDATA%\LocalLow\Sun\Java\Deployment\cache`;
   - On Mac OSX this folder is located in: `/Users/user_name/Library/Caches/Java/cache`;
   - On Linux this folder is located in: `/home/user/.java/deployment/cache`.

4. Remove the Author Applet Frameworks cache from the local drive and try again:

   - On Windows Vista or 7 this folder is located in: `%APPDATA%\Roaming\com.oxygenxml.author.component`;

- On Windows XP this folder is located in: `%APPDATA%\com.oxygenxml.author.component`;
- On Mac OSX this folder is located in: `/Users/user_name/Library/Preferences/com.oxygenxml.author.component`;
- On Linux this folder is located in: `/home/user/.com.oxygenxml.author.component`.

5. Problems sometimes occur after upgrading the web browser and/or the JavaTM runtime. Redeploy the applet on the server by running ANT in your Author Component project. However, doing this does not always fix the problem, which often lies in the web browser and/or in the Java plug-in itself.
6. Sometimes when the HTTP connection is slow on first time uses the JVM would simply shut down while the jars were being pushed to the local cache (i.e., first time uses). This shut down typically occurs while handling `oxygen.jar`. One of the reasons could be that some browsers (Firefox for example) implement some form of "Plugin hang detector" See *https://developer.mozilla.org/en/Plugins/Out_of_process_plugins/The_plugin_hang_detector*.

Enable JavaWebstart logging on your computer to get additional debug information:

1. Open a console and run `javaws -viewer`;
2. In the **Advanced** tab, expand the **Debugging** category and check all boxes.
3. Expand the **Java console** category and choose **Show console**.
4. Save settings.
5. After running the applet, you will find the log files in:
   - On Windows this folder is located in: `%APPDATA%\LocalLow\Sun\Java\Deployment\log`;
   - On Mac OSX this folder is located in: `/Users/user_name/Library/Caches/Java/log`;
   - On Linux this folder is located in: `/home/user/.java/deployment/log`.

Adding MathML support in the Author Component Web Applet

By default the Author Component Web Applet project does not come with the libraries necessary for viewing MathML equations in the Author page.

In the JNLP file `author-component-dita.jnlp` you must refer additional libraries necessary for the `JEuclid` library to parse MathML equations:

```
<jar href="lib/jcip-annotations.jar"/>
<jar href="lib/jeuclid-core.jar"/>
<jar href="lib/batik-all-1.7.jar"/>
<jar href="lib/commons-io-1.3.1.jar"/>
<jar href="lib/commons-logging-1.0.4.jar"/>
<jar href="lib/xmlgraphics-commons-1.4.jar"/>
```

These additional libraries can be copied to the component project `lib` directory from an `OXYGEN_INSTALLATION_DIRECTORY/lib` directory.

In order to have the DITA with MathML specialization fully operational to edit specialized DITA Composite with MathML content you also have to include in the frameworks bundled with the component `frameworks.zip.jar` the entire `OXYGEN_INSTALLATION_DIRECTORY/frameworks/mathml2` Mathml2 framework directory which is used to solve references to MathML DTDs.

Frequently asked questions

Installation and licensing

1. What hosting options are available for applet delivery and licensing services (i.e., Apache, IIS, etc.)?

   For applet delivery any web server. We currently use Apache to deploy the sample on our site. For the floating license server you would need a J2EE server, like Tomcat if you want to restrict the access to the licenses.

If you do not need the access restrictions that are possible with a J2EE server you can simplify the deployment of the floating license server by using the standalone version of this server. The standalone license server is a simple Java application that communicates with Author Component by TCP/IP connections.

2. Are there any client requirements beyond the Java VM and (browser) Java Plug-In Technology?

   Oracle (formerly Sun) Java JRE version 1.6 update 10 or newer. At least 200 MB disk space and 200MB free memory would be necessary for the Author Applet component.

3. Are there any other client requirements or concerns that could make deployment troublesome (i.e., browser security settings, client-side firewalls and AV engines, etc.)?

   The applet is signed and will request access to the user machine, in order to store customization data (frameworks). The applet needs to be signed by you with a valid certificate.

4. How sensitive is the applet to the automatic Java VM updates, which are typically on by default (i.e., could automatic updates potentially "break" the run-time)?

   The component should work well with newer Java versions but we cannot guarantee this.

5. How and when are "project" related files deployed to the client (i.e., applet code, DTD, styling files, customizations, etc.)?

   Framework files are downloaded on the first load of the applet. Subsequent loads will re-use the cached customization files and will be much faster.

6. For on-line demo (http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-dita.html), noted a significant wait during initial startup. Any other mechanisms to enhance startup time?

   See explanation above.

7. Does the XML Author component support multiple documents being open simultaneously? What are the licensing ramifications?

   The current floating license model allows for now only two concurrent components from the same computer when using the license servlet. An additional started component will take an extra license seat.

   Another licensing technique would be to embed the license key in one of the jar libraries used by the applet. But you would need to implement your own way of determining how many users are currently editing using the Author applet.

8. Is there any internet traffic during an editing session (user actively working on the content, on the client side, in the XML Author component))?

   No.

Functionality

1. How and when are saves performed back to the hosting server?

   What you can see on our web site is just an example of the Author component (which is a Java Swing component) used in an Applet.

   This applet is just for demonstration purposes. It's source can be at most a starting point for a customization. You should implement, sign and deploy your custom applet implementation.

   The save operation could be implemented either in Javascript by requesting the XML content from the Applet or in Java directly working with the Author component. You would be responsible to send the content back to the CMS.

2. Is there a particular XML document size (or range) when the Author applet would start to exhibit performance problems?

   The applet has a total amount of used memory specified in the JNLP JavaWebstart configuration file which can be increased if necessary. By default it is 156 Mb. It should work comfortably with documents of 1-3 megabytes.

3. **3.** What graphic formats can be directly rendered in the XML Author component?

   GIF, JPEG, PNG, BMP and SVG.

4. Can links be embedded to retrieve (from the server) and "play" other types of digital assets, such as audio or video files?

   You could add listeners to intercept clicks and open the clicked links. This would require a good knowledge of the Author SDK. The Author component can only render static images (no GIF animations).

5. Does the XML Author component provide methods for uploading ancillary files (new graphics, for instance) to the hosting server?

   No.

6. Does the XML Author component provide any type of autosave functionality?

   By default no but you could customize the applet that contains the author component to save its content periodically to a file on disk.

7. Assuming multiple documents can be edited simultaneously, can content be copied, cut and pasted from one XML Author component "instance" to another?

   Yes.

8. Does the XML Author component support pasting content from external sources (such as a web page or a Microsoft Word document and, if so, to what extent?

   If no customizations are available the content is pasted as simple text. We provide customizations for the major frameworks (DITA, Docbook, TEI, etc) which use a conversion XSLT stylesheet to convert HTML content from clipboard to the target XML.

9. Can UTF-8 characters (such as Greeks, mathematical symbols, etc.) be inserted and rendered?

   Any UTF-8 character can be inserted and rendered as long as the font used for editing supports rendering the characters. The font can be changed by the developers but not by the users. When using a logical font (which by default is *Serif* for the Author component) the JVM will know how to map all characters to glyphs. There is no character map available but you could implement one

Customization

1. Please describe, in very general terms, the menus, toolbars, context menu options, "helper panes", etc. that are available for the XML Author component "out of the box".

   You can mount on your custom toolbar all actions available in the standalone Oxygen application for editing in the Author page. This includes custom actions defined in the framework customized for each XML type.

   The Author component also can provide the *Outline*, *Model*, *Elements* and *Attributes* views which can be added to your own panels (see sample applet).

2. Please describe, in general terms, the actions, project resources (e.g., DTD/Schema for validation purposes, CSS/XSL for styling, etc.) and typical level of effort that would be required to deploy a XML Author component solution for a customer with a proprietary DTD.

   The Author internal engine uses CSS to render XML.

   For a special type of XML you can create a custom framework (which also works in an Oxygen standalone version) which would also contain default schemas and custom actions. A simple framework would probably need 2-3 weeks development time. For a complex framework with many custom actions it could take a couple of months. Oxygen already has frameworks for editing Docbook, DITA, TEI, etc. Sources for them are available in *the Author SDK*.

   More than one framework can coexist in the same Oxygen instance (the desktop standalone version or the applet version) and can be used at the same time for editing XML documents.

3. Many customers desire a very simplistic interface for contributors (with little or no XML expertise) but a more robust XML editing environment for editors (or other users with more advanced XML savviness). How well does the XML Author component support varying degrees of user interface complexity and capability?

   - *Showing/hiding menus, toolbars, helpers, etc.*

     All the UI parts from the Author component are assembled by you. You could provide two applet implementations: one for advanced/power users and one for technical writers.

   - *Forcing behaviors (i.e., ensuring change tracking is on and preventing it from being shut down)*

     You could avoid placing the change tracking toolbar actions in the custom applet. You could also use API to turn change tracking ON when the content has been loaded.

   - *Preventing access to "privileged" editor processes (i.e., accept/reject changes)*

     You can remove the change tracking actions completely in a custom applet implementation. Including the ones from the contextual menu.

   - *Presenting and/or describing XML constructs (i.e., tags) in "plain-English"*

     Using our API you can customize what the Outline or Breadcrumb presents for each XML tag. You can also customize the in-place content completion list.

   - *Presenting a small subset of the overall XML tag set (rather than the full tag set) for use by contributors (i.e., allowing an author to only insert Heading, Para and inline emphasis) Could varying "interfaces", with different mixes these capabilities and customizations, be developed and pushed to the user based on a "role" or a similar construct?*

     The API allows for a content completion filter which also affects the *Elements* view.

4. Does the XML Author component's API provide access to the XML document, for manipulation purposes, using common XML syntax such as DOM, XPath, etc.?

   Yes, using the Author API.

5. Can custom dialogs be developed and launched to collect information in a "form" (with scripting behind to push tag the collection information and embed it in the XML document?

   Yes.

6. Can project resources, customizations, etc. be readily shared between the desktop and component versions of your XML Author product line?

   A framework developed for the Desktop Oxygen application can then be bundled with an Author component in a custom applet. For example the Author demo applet from our web site is DITA-aware using the same framework as the Oxygen standalone distribution.

   A custom version of the applet that includes one or more customized frameworks and user options can be built and deployed for non-technical authors by a technical savvy user using a built-in tool of Oxygen. All the authors that load the deployed applet from the same server location will share the same frameworks and options.

   A custom editing solution can deploy one or more frameworks that can be used at the same time.

Chapter 9

Predefined Document Types

The following are the short presentations of some document types that come bundled with Oxygen XML Editor plugin . For each document type there are presented built-in transformation scenarios, document templates and Author extension actions.

Chapter 10

The DITA Topics Document Type

The Darwin Information Typing Architecture (DITA) is an XML-based architecture for authoring, producing, and delivering technical information. It divides content into small, self-contained topics that can be reused in different deliverables. The extensibility of DITA permits organizations to define specific information structures and still use standard tools to work with them.

A file is considered to be a DITA topic document when either of the following occurs:

- the root element name is one of the following: `concept`, `task`, `reference`, `dita`, `topic`
- PUBLIC ID of the document is one of the PUBLIC ID's for the elements above
- the root element of the file has an attribute named `DITAArchVersion` attribute from the “http://dita.oasis-open.org/architecture/2005/” namespace. This enhanced case of matching is only applied when the **Enable DTD processing** option from the ***Document Type Detection*** *option page* is enabled.

The default schema used for DITA topic documents is located in `${frameworks}/dita/dtd/ditabase.dtd`, where *${frameworks}* is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering DITA content in Author mode is `${frameworks}/dita/css/dita.css`.

The default XML catalog is `${frameworks}/dita/catalog.xml`.

Chapter 11

The DocBook 4 Document Type

DocBook is a very popular set of tags for describing books, articles, and other prose documents, particularly technical documentation.

A file is considered to be a *DocBook 4* document when one of the following conditions are true:

- root element name is `book` or `article`
- the PUBLIC ID of the document contains the string `-//OASIS//DTD DocBook XML`

The schema of *DocBook 4* documents is `${frameworks}/docbook/dtd/docbookx.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering DocBook content is located in `${frameworks}/docbook/css/docbook.css`.

The XML catalog is stored in `${frameworks}/docbook/catalog.xml`.

Chapter 12

The DocBook 5 Document Type

A file is considered to be a DocBook 5 document when the namespace is *http://docbook.org/ns/docbook*.

DocBook 5 documents use a Relax NG and Schematron schema located in `${frameworks}/docbook/5.0/rng/docbookxi.rng`, where *${frameworks}* is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering DocBook content is located in `${frameworks}/docbook/css/docbook.css`.

The XML catalog is stored in `${frameworks}/docbook/5.0/catalog.xml`.

Chapter 13

The TEI P4 Document Type

The **Text Encoding Initiative (TEI) Guidelines** is an international and interdisciplinary standard that enables libraries, museums, publishers, and individual scholars to represent a variety of literary and linguistic texts for online research, teaching, and preservation.

A file is considered to be a TEI P4 document when either of the following occurs:

- the root's local name is `TEI.2`
- the document's public id is *-//TEI P4*

The DTD schema used for these documents is located in `${frameworks}/tei/tei2xml.dtd`, where `${frameworks}` is a subdirectory of the Oxygen XML Editor plugin install directory.

The CSS file used for rendering TEI P4 content is located in `${frameworks}/tei/xml/tei/css/tei_oxygen.css`.

There are two default catalogs for TEI P4 document type:

- `${frameworks}/tei/xml/teip4/schema/dtd/catalog.xml`
- `${frameworks}/tei/xml/teip4/custom/schema/dtd/catalog.xml`

Chapter 14

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 Editor plugin install directory.
- A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `graphic` DITA element with the `url` attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

Chapter 15

The XHTML Document Type

The Extensible HyperText Markup Language, or XHTML, is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax.

A file is considered to be a XHTML document when the root element name is a `html`.

The schema used for these documents is located in `${frameworks}/xhtml/dtd/xhtml1-strict.dtd`, where *${frameworks}* is a subdirectory of the Oxygen XML Editor plugin install directory.

The default CSS options for the XHTML document type are set to merge the CSSs specified in the document with the CSSs defined in the XHTML document type.

The CSS file used for rendering XHTML content is located in `${frameworks}/xhtml/css/xhtml.css`.

There are three default catalogs for XHTML document type:

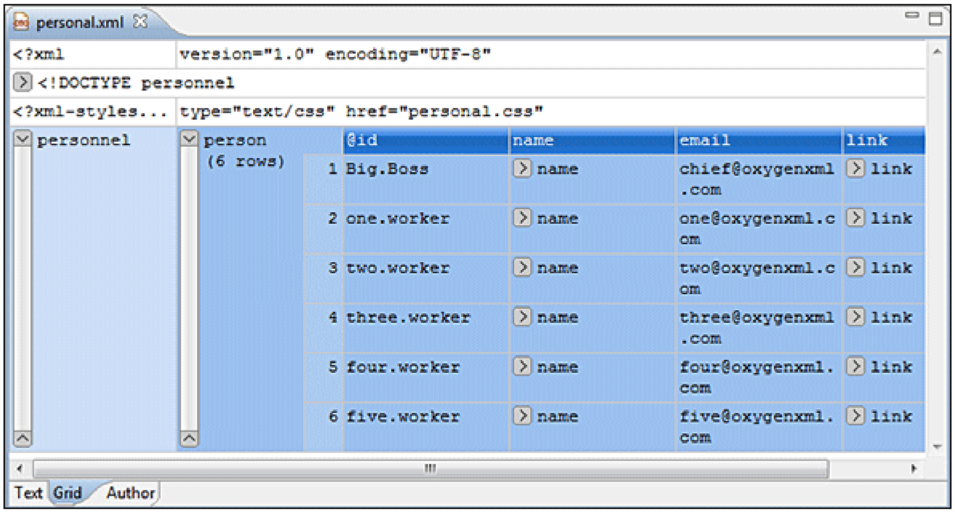
- `${frameworks}/xhtml/dtd/xhtmlcatalog.xml`
- `${frameworks}/xhtml11/dtd/xhtmlcatalog.xml`
- `${frameworks}/xhtml11/schema/xhtmlcatalog.xml`

Chapter 16

Grid Editor

Topics:

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)**.



<?xml	version="1.0" encoding="UTF-8"
<!DOCTYPE personnel	
<?xml-styles...	type="text/css" href="personal.css"

personnel — person (6 rows)

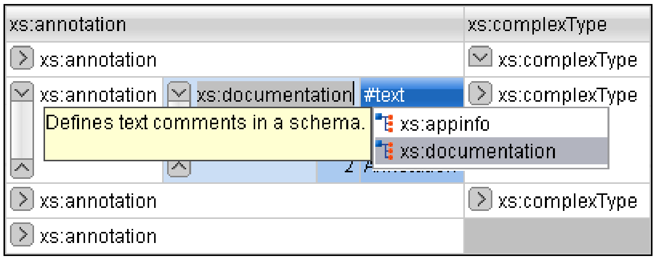
	@id	name	email	link
1	Big.Boss	name	chief@oxygenxml.com	link
2	one.worker	name	one@oxygenxml.com	link
3	two.worker	name	two@oxygenxml.com	link
4	three.worker	name	three@oxygenxml.com	link
5	four.worker	name	four@oxygenxml.com	link
6	five.worker	name	five@oxygenxml.com	link

Figure 191: 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.



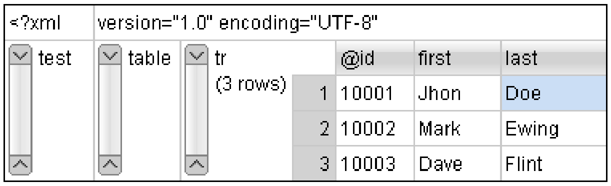
xs:annotation	xs:complexType
xs:annotation	xs:complexType
xs:annotation — xs:documentation — #text	xs:complexType
xs:annotation	xs:complexType
xs:annotation	

Defines text comments in a schema.

Figure 192: Content Completion in Grid Editor

Layouts: Grid and Tree

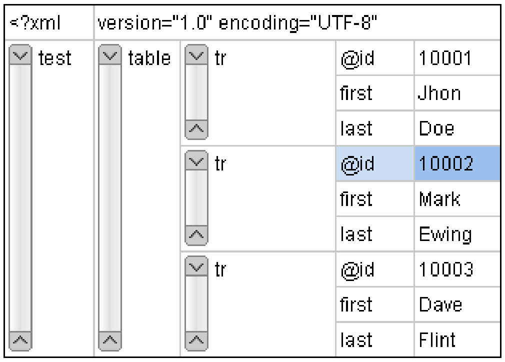
The grid editor has two modes for the layout. The default one is the grid layout. This smart layout detects the recurring elements in the XML document and creates tables having as columns the children (including the attributes) of these elements. This way it is possible to have tables nested in other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



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

Figure 193: Grid Layout

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



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

Figure 194: 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 eement 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 involvespressing 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: A↓Z Z↓A

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.

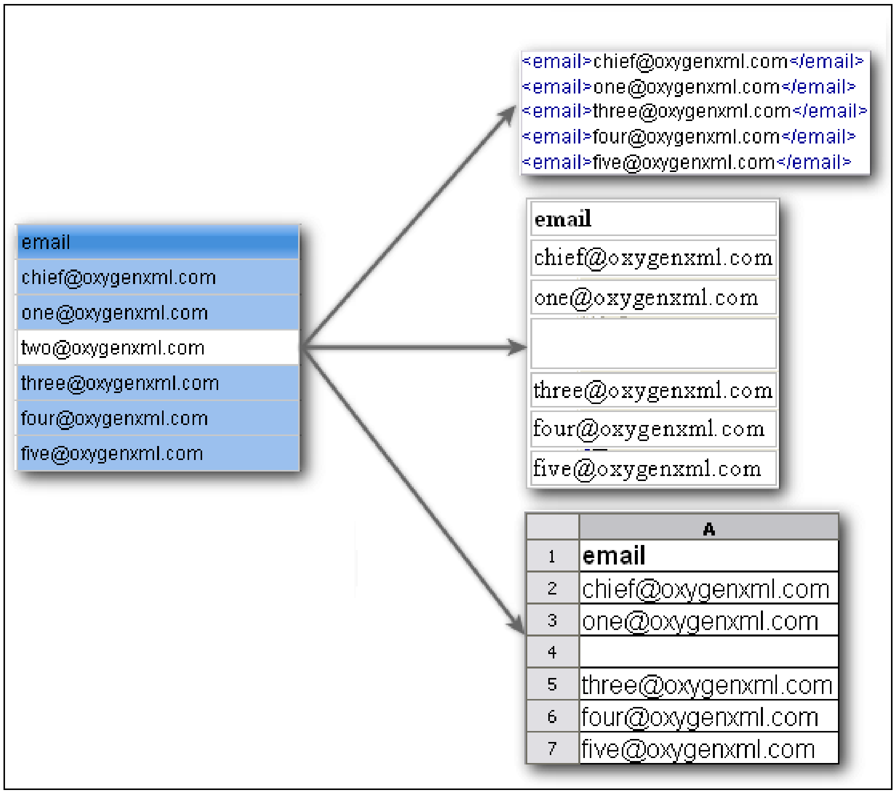
Copy and Paste in the Grid Editor

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.



email
chief@oxygenxml.com
one@oxygenxml.com
two@oxygenxml.com
three@oxygenxml.com
four@oxygenxml.com
five@oxygenxml.com

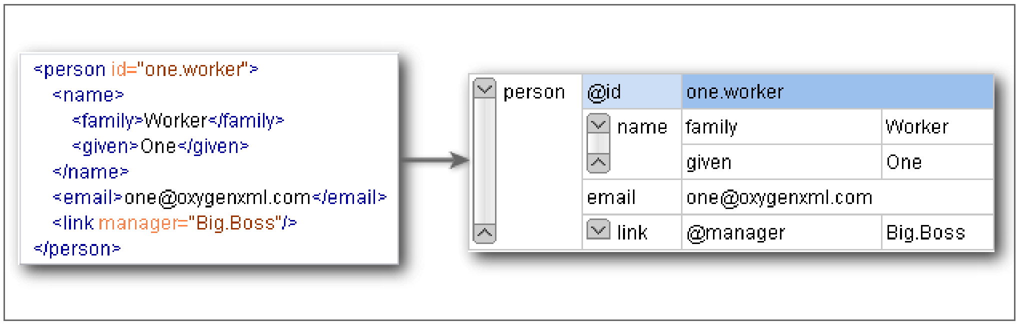
```
<email>chief@oxygenxml.com</email>
<email>one@oxygenxml.com</email>
<email>three@oxygenxml.com</email>
<email>four@oxygenxml.com</email>
<email>five@oxygenxml.com</email>
```

email
chief@oxygenxml.com
one@oxygenxml.com
three@oxygenxml.com
four@oxygenxml.com
five@oxygenxml.com

	A
1	email
2	chief@oxygenxml.com
3	one@oxygenxml.com
4	
5	three@oxygenxml.com
6	four@oxygenxml.com
7	five@oxygenxml.com

Figure 195: 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.

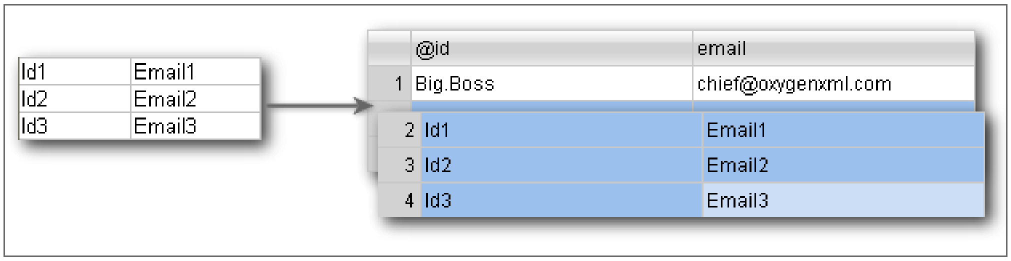


```
<person id="one.worker">
  <name>
    <family>Worker</family>
    <given>One</given>
  </name>
  <email>one@oxygenxml.com</email>
  <link manager="Big.Boss"/>
</person>
```

person	@id	one.worker	
	name	family	Worker
		given	One
	email	one@oxygenxml.com	
	link	@manager	Big.Boss

Figure 196: 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.



Id1	Email1
Id2	Email2
Id3	Email3

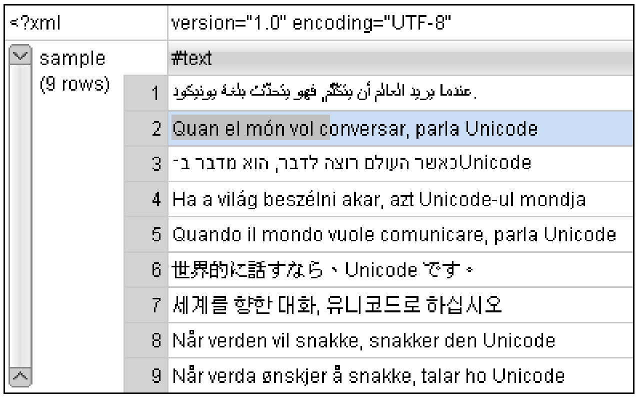
	@id	email
1	Big.Boss	chief@oxygenxml.com
2	Id1	Email1
3	Id2	Email2
4	Id3	Email3

Figure 197: Copying tab separated values into grid

Bidirectional Text Support in the Grid Editor

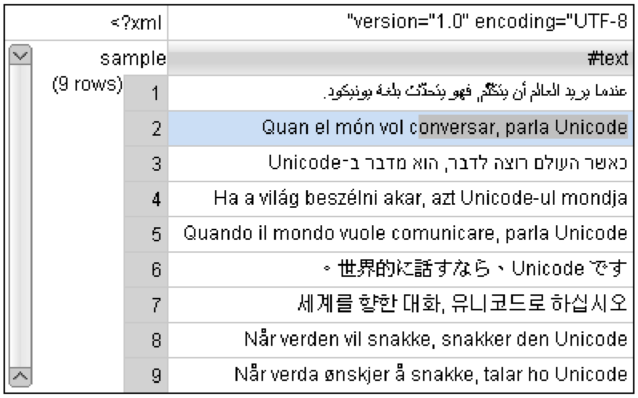
If you are editing documents employing a different text orientation you can change the way the text is rendered and edited in the grid cells by using the **(Ctrl-Shift-O)** shortcut to switch from the default left to right text orientation to the right to left orientation.

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 (9 rows)		#text
	1	عندما يريد العالم أن يتكلّم، فهو يتحدّث 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 198: Default left to right text orientation



<?xml	"version="1.0" encoding="UTF-8	
sample (9 rows)		#text
	1	عندما يريد العالم أن يتكلّم، فهو يتحدّث 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 199: Right to left text orientation

Chapter 17

Transforming Documents

Topics:

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 Editor plugin you can transform your XML documents to the following formats without having to exit from the application:

- **PDF** - Adobe Portable Document Format (PDF) is a compact binary file format that can be viewed and printed by anyone, anywhere across a broad range of hardware and software using the free PDF Viewer from *Adobe*.
- **PS** - PostScript is the leading printing technology from *Adobe* for high-quality, best-in-class printing solutions ranging from desktop devices to the most advanced digital presses, platemakers, and large format image setters in the world. PostScript files can be viewed using viewers such as GhostScript, but are more commonly created as a prepress format.
- **TXT** - Text files are Plain ASCII Text and can be opened in any text editor or word processor.
- **XML** - XML stands for eXtensible Markup Language and is a *W3C* standard markup language, much like HTML, which was designed to describe data. XML tags are not predefined in XML. You must define your own tags. XML uses a Document Type Definition (DTD), an XML Schema or a Relax NG schema to describe the data. XML with a DTD, XML Schema or Relax NG schema is designed to be self-descriptive. XML is not a replacement for HTML. XML and HTML were designed with different goals:

  - XML was designed to describe data and to focus on what data is.
  - HTML was designed to display data and to focus on how data looks.
  - HTML is about displaying information, XML is about describing information.

- **XHTML** - XHTML stands for eXtensible HyperText Markup Language, a *W3C* standard. XHTML is aimed to replace HTML. While almost identical to HTML 4.01, XHTML is a stricter and cleaner version of HTML. XHTML is HTML defined as an XML application.

For transformation to formats that are not listed above simply install the tool chain required to perform the transformation and process the xml files created with Oxygen XML Editor plugin in accordance with the processor instructions.

All formatting during a transformation is provided under the control of an Extensible Stylesheet (XSLT). Specifying the appropriate XSLT enables transformation to the above formats and preparation of output files for specific user agent viewing applications, including:

- **HTML** - HTML stands for Hyper Text Markup Language and is a *W3C Standard* for the World Wide Web. HTML is a text file containing small markup tags. The markup tags tell the Web browser how to display the page. An HTML file must have an `htm` or `html` file extension. An HTML file can be created using a simple text editor.
- **HTML Help** - *Microsoft HTML Help* is the standard help system for the Windows platform. Authors can use HTML Help to create online help for a software application or to create content for a multimedia title or Web site. Developers can use the HTML Help API to program a host application or hook up context-sensitive help to an application.
- **JavaHelp** - JavaHelp software is a full-featured, platform-independent, extensible help system from *Sun Microsystems/Oracle* that enables developers and authors to incorporate online help in applets, components, applications, operating systems, and devices. JavaHelp is a free product and the binaries for JavaHelp can be redistributed.
- **Eclipse Help** - Eclipse Help is the help system incorporated in the *Eclipse platform* that enables Eclipse plugin developers to incorporate online help in their plugins.

Many other target formats are possible, these are the most popular. The basic condition for transformation to any format is that your source document is well-formed. Always, make sure that the XSL used for the transformation is the right one according to the desired output format and with the input source definition. For example, if you want to transform to HTML format using a DocBook html stylesheet, your source xml document should respect the DocBook DTD.

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an output document by using special formatting vocabulary.

XSL consists of three parts:

- **XSL Transformations (XSLT)** - XSLT is a language for transforming XML documents.
- **XML Path (XPath) Language** - XPath is an expression language used by XSLT to access or refer parts of an XML document. XPath is also used by the XML Linking specification.

- **XSL Formatting Objects (XSL:FO)** - XSL:FO is an XML vocabulary for specifying formatting semantics.

Oxygen XML Editor plugin supports XSLT/XPath version 1.0 using Saxon 6.5.5, Xalan, Xsltproc, MSXML (3.0, 4.0, .NET) and XSLT/XPath 2.0 by using Saxon 9.3.0.5 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 Editor 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 Editor plugin comes with a built-in version of Ant and a built-in version of DITA-OT but different versions can be set in the scenario.

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 Editor 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 Editor 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))**.

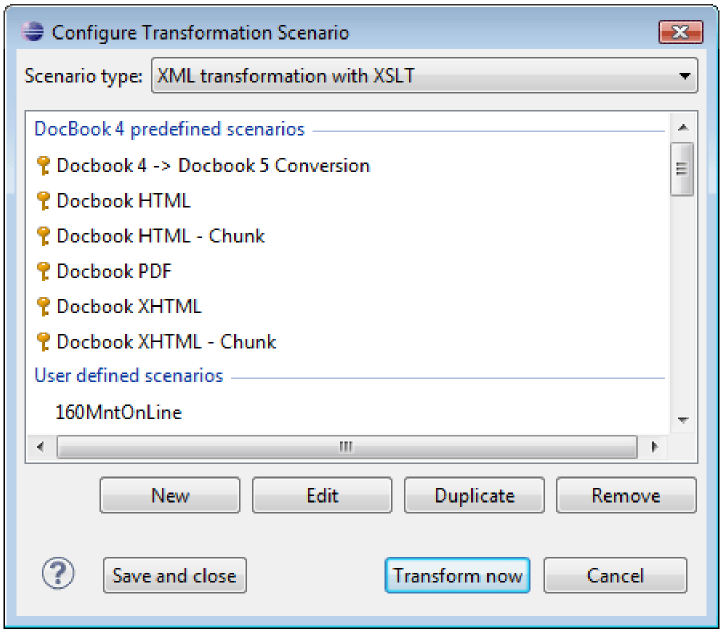
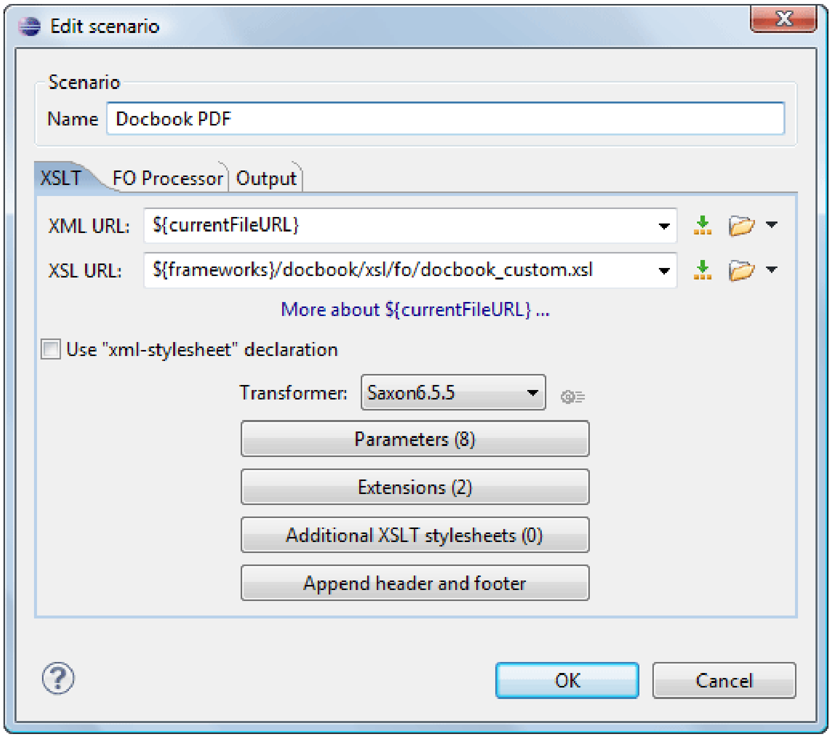


Figure 200: Configure Transformation Scenario Dialog

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:



Edit scenario

Scenario

Name: Docbook PDF

XML URL: ${currentFileURL}

XSL URL: ${frameworks}/docbook/xsl/fo/docbook_custom.xsl

☐ Use "xml-stylesheet" declaration

Transformer: Saxon6.5.5

Figure 201: 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 Editor 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, thenthe 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.

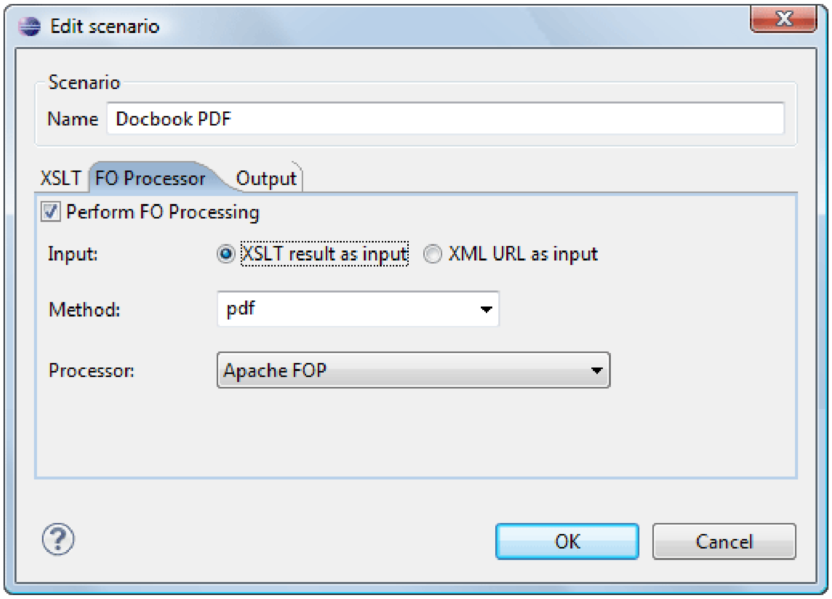
The advanced options available only in Saxon EE are:

- **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 Editor 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*.



Scenario

Name: Docbook PDF

☑ Perform FO Processing

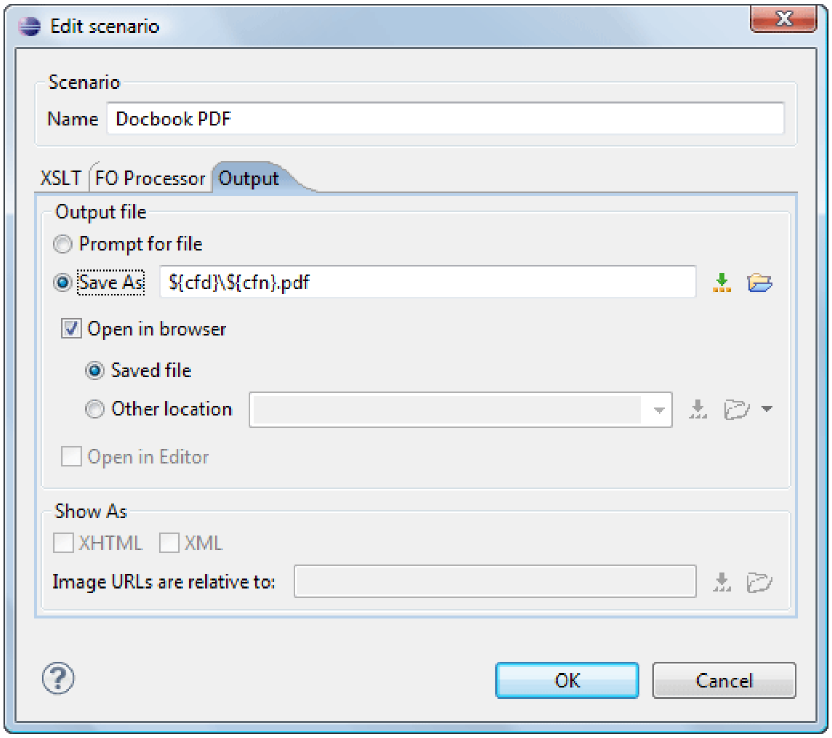
Input: ● XSLT result as input ○ XML URL as input

Method: pdf

Processor: Apache FOP

Figure 202: 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.*



Scenario

Name: Docbook PDF

Output file

○ Prompt for file

● Save As ${cfd}\${cfn}.pdf

☑ Open in browser

● Saved file

○ Other location

□ Open in Editor

Show As

□ XHTML □ XML

Image URLs are relative to:

Figure 203: 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 Editor plugin editor variables* or *custom editor variables*.
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor 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.

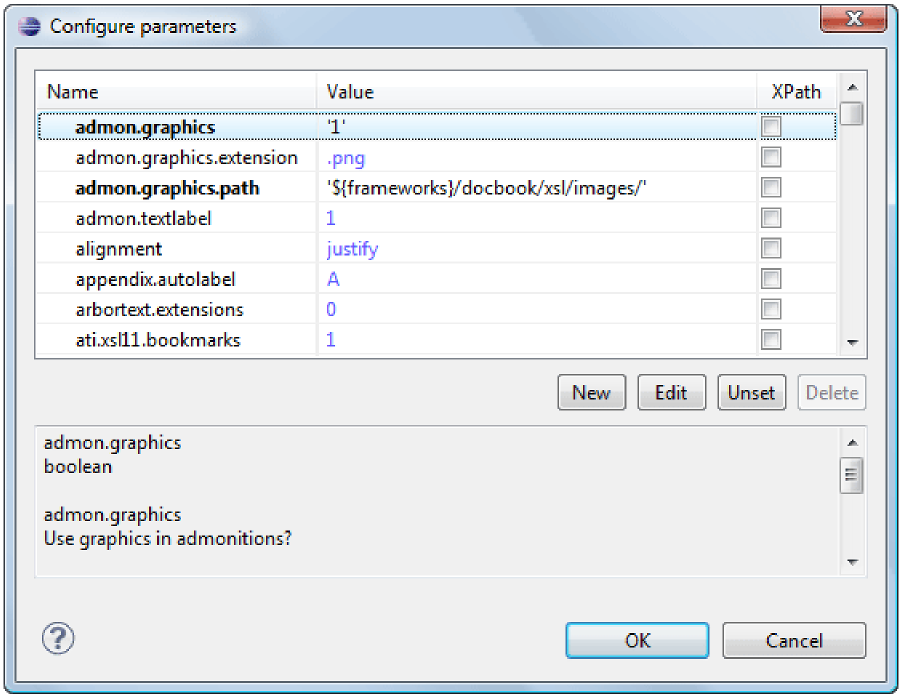
- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Editor plugin should open automatically at the end of the transformation the file specified in the **Save As** text field.
- **Other location** - When **Open in System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin should not open the file specified in the **Save As** text field, it should open the file specified in the text field of the **Other location** radio button. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variable*.
- **Open in editor** - When this is 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 Editor 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 Editor 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:



Name	Value	XPath
admon.graphics	'1'	☐
admon.graphics.extension	.png	☐
admon.graphics.path	'${frameworks}/docbook/xsl/images/'	☐
admon.textlabel	1	☐
alignment	justify	☐
appendix.autolabel	A	☐
arbortext.extensions	0	☐
ati.xsl11.bookmarks	1	☐

admon.graphics
boolean

admon.graphics
Use graphics in admonitions?

Figure 204: 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}*, *${cfd}*, 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 Editor 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.

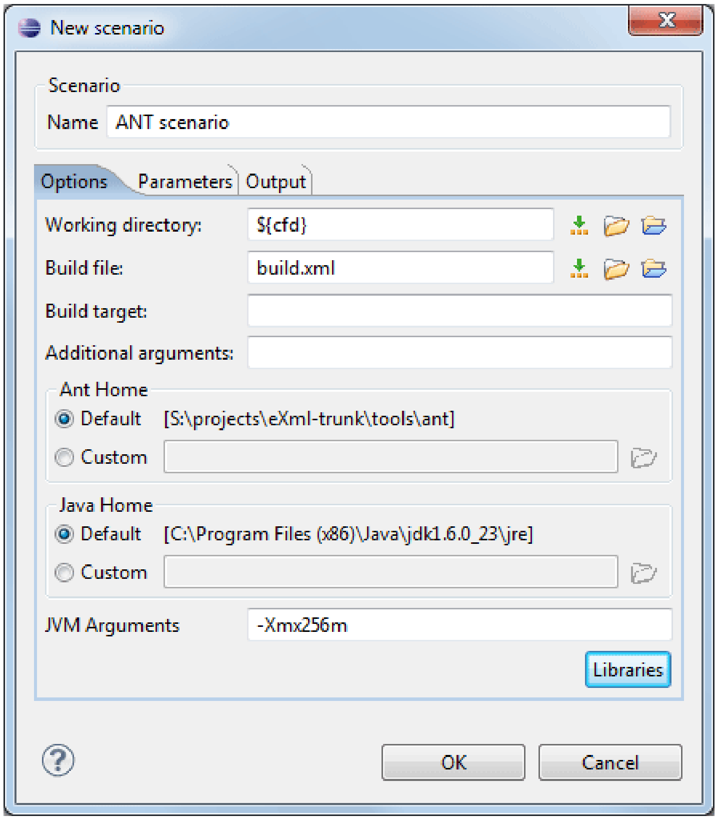
Duplicate a Transformation Scenario

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 Editor 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.



New scenario

Scenario

Name: ANT scenario

Working directory: ${cfd}

Build file: build.xml

Build target:

Additional arguments:

Ant Home

● Default [S:\projects\eXml-trunk\tools\ant]

○ Custom

Java Home

● Default [C:\Program Files (x86)\Java\jdk1.6.0_23\jre]

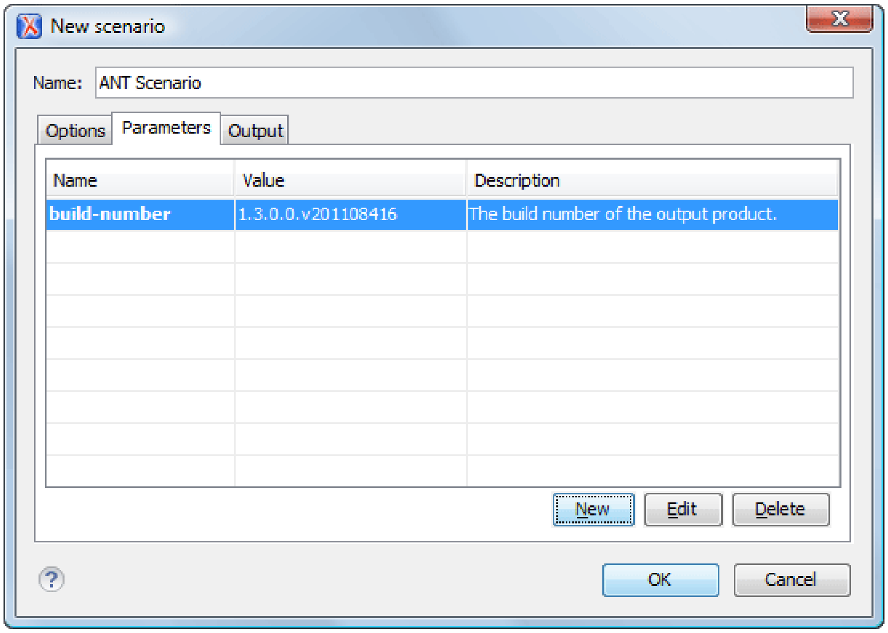
○ Custom

JVM Arguments: -Xmx256m

Figure 205: 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 Editor plugin . A custom Ant installation can also be set.
- **Java Home** - The path to the Java Virtual Machine that runs the Ant transformation. By default it is the Java Virtual Machine that is bundled with Oxygen XML Editor plugin . A custom Java virtual machine can also be set.
- **JVM Arguments** - This parameter allows you to set specific parameters to the Java Virtual Machine used by Ant. By default it is set to -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).



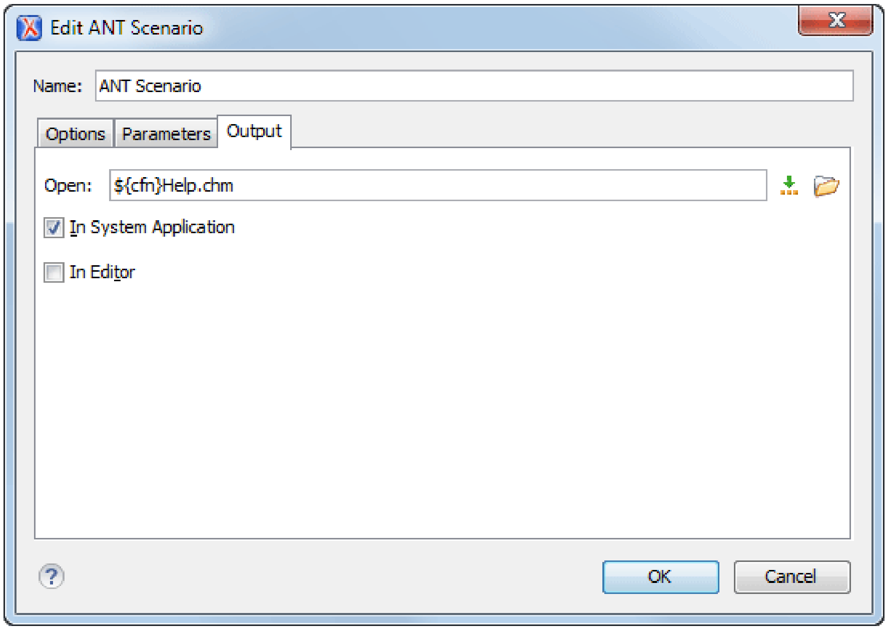
New scenario

Name: ANT Scenario

Name	Value	Description
build-number	1.3.0.0.v201108416	The build number of the output product.

Figure 206: 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.



Edit ANT Scenario

Name: ANT Scenario

Open: ${cfn}Help.chm

☑ In System Application

☐ In Editor

Figure 207: 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 Editor 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**.

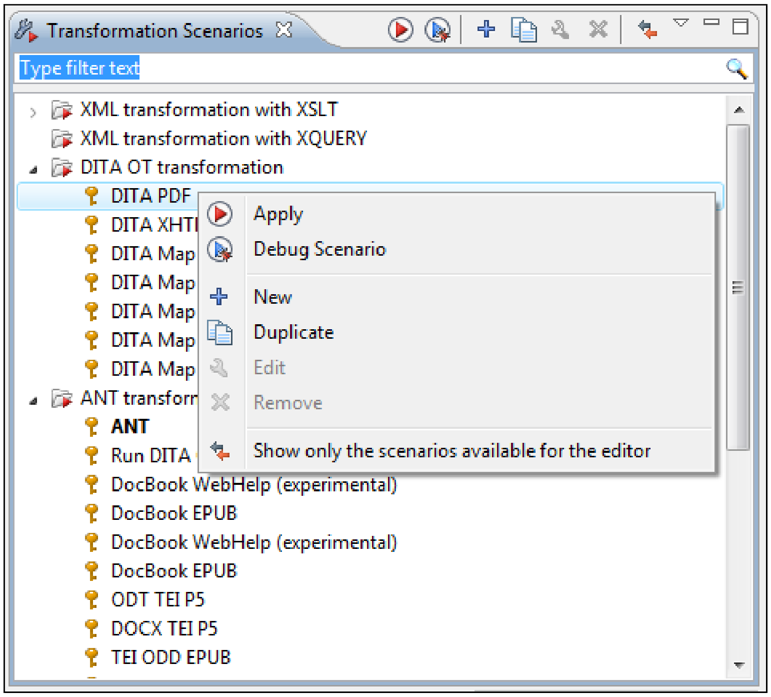


Figure 208: The Scenarios view

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 Editor plugin .

Supported XSLT Processors

Oxygen XML Editor plugin comes with the following XSLT processors:

- **Xalan 2.7.1** - *Xalan-Java* is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements XSL Transformations (XSLT) Version 1.0 and XML Path Language (XPath) Version 1.0.
- **Saxon 6.5.5** - *Saxon 6.5.5* is an XSLT processor, which implements the Version 1.0 XSLT and XPath with a number of powerful extensions. This version of Saxon also includes many of the new features that were first defined in the XSLT 1.1 working draft, but for conformance and portability reasons these are not available if the stylesheet header specifies `version="1.0"`.
- **Saxon 9.3.0.5 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 Editor plugin . Saxon EE includes an XML Schema processor, and schema-aware XSLT, XQuery, and XPath processors.

  The validation in schema aware transformations is done according to the W3C XML Schema 1.0 specification or according to the W3C XML Schema 1.1 one. This can be *configured in Preferences*.

Besides the above list Oxygen XML Editor plugin supports the following processors:

- **Xsltproc (libxslt)** - *Libxslt* is the XSLT C library developed for the Gnome project. `Libxslt` is based on libxml2 the XML C library developed for the Gnome project. It also implements most of the EXSLT set of processor-portable extensions functions and some of Saxon's evaluate and expressions extensions. The libxml2 version included in Oxygen XML Editor plugin is 2.7.6 and the libxslt version is 1.1.26

  Oxygen XML Editor plugin uses Libxslt through its command line tool (`Xsltproc`). The XSLT processor is included into the distribution kit of the stand-alone version for Windows and Mac OS X. Because there are differences between different Linux distributions, on Linux you must install `Libxslt` on your machine as a separate application and set the PATH variable to contain the `Xsltproc` executable.

  If you do not have the `Libxslt` library already installed, you should copy the following files from Oxygen XML Editor plugin stand-alone installation directory to the root of the `com.oxygenxml.editor_` 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`,`libxsml2.so.2`
  - on Mac OS X: `xsltproc.mac`, `libexslt`, `libxslt`, `libxml`

  The Xsltproc processor can be configured from the ***XSLTPROC** options page*.

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

- **MSXML 3.0/4.0** - *MSXML 3.0/4.0* is available only on Windows 2000, Windows NT and Windows XP platforms. It can be used for *transformation* and *validation of XSLT stylesheets* .

  Oxygen XML Editor plugin uses the Microsoft XML parser through its command line tool *`msxsl.exe`*.

Because `msxsl.exe` is only a wrapper, Microsoft Core XML Services (MSXML) must be installed on the computer otherwise you will get a corresponding warning. You can get the latest Microsoft XML parser from *Microsoft web-site*

- **MSXML .NET** - *MSXML .NET* is available only on Windows NT4, Windows 2000 and Windows XP platforms. It can be used for *transformation* and *validation of XSLT stylesheets* .

  Oxygen XML Editor plugin performs XSLT transformations and validations using .NET Framework's XSLT implementation (`System.Xml.Xsl.XslTransform` class) through the **nxslt** command line utility. The nxslt version included in Oxygen XML Editor plugin is 1.6.

  You should have the .NET Framework version 1.0 already installed on your system otherwise you will get the following warning: `MSXML.NET requires .NET Framework version 1.0 to be installed. Exit code: 128`

  You can get the .NET Framework version 1.0 from the *Microsoft website*
- **.NET 1.0** - A transformer based on the `System.Xml` 1.0 library available in the .NET 1.0 and .NET 1.1 frameworks from Microsoft (*http://msdn.microsoft.com/xml/*). It is available only on Windows.

  You should have the .NET Framework version 1.0 or 1.1 already installed on your system otherwise you will get the following warning: `MSXML.NET requires .NET Framework version 1.0 to be installed. Exit code: 128`

  You can get the .NET Framework version 1.0 from the *Microsoft website*
- **.NET 2.0** - A transformer based on the `System.Xml` 2.0 library available in the .NET 2.0 framework from *Microsoft*. It is available only on Windows.

  You should have the .NET Framework version 2.0 already installed on your system otherwise you will get the following warning: `MSXML.NET requires .NET Framework version 2.0 to be installed. Exit code: 128`

  You can get the .NET Framework version 2.0 from the *Microsoft website*

Configuring Custom XSLT Processors

You can *configure other XSLT transformation engines* than *the ones which come with the Oxygen XML Editor 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 Editor plugin linked message* then a click on the message in the output view highlights the location of the message in an editor panel containing the file referred in the message.

Configuring the XSLT Processor Extensions Paths

The Xalan and Saxon processors support the use of extension elements and extension functions. Unlike a literal result element, which the stylesheet simply transfers to the result tree, an extension element performs an action. The extension is usually used because the XSLT stylesheet fails in providing adequate functions to the user for accomplishing a more complex task.

Samples on how to use extensions can be found at:

- for Xalan - *http://xml.apache.org/xalan-j/extensions.html*
- for Saxon 6.5.5 - *http://saxon.sourceforge.net/saxon6.5.5/extensions.html*
- for Saxon 9.3.0.5 - *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 Editor plugin .

The Built-in XSL-FO Processor

The Oxygen XML Editor plugin installation package is distributed with the *Apache FOP* that is a Formatting Objects processor for rendering your XML documents to PDF. *FOP* is a print and output independent formatter driven by XSL Formatting Objects. *FOP* is implemented as a Java application that reads a formatting object tree and renders the resulting pages to a specified output.

To include PNG images in the final PDF document you need the *JIMI* or *JAI* libraries. For 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 Editor plugin but using them is very easy. You need to download them and *create an external FO processor* based on the built-in FOP libraries and the extension library. The *external FO processor created in **Preferences*** will have a command line like:

```
java -cp "${oxygenInstallDir}/lib/xercesImpl.jar:
${oxygenInstallDir}/lib/fop.jar:${oxygenInstallDir}/lib/
avalon-framework-4.2.0.jar:
${oxygenInstallDir}/lib/batik-all-1.7.jar:${oxygenInstallDir}/lib/
commons-io-1.3.1.jar:
${oxygenInstallDir}/lib/xmlgraphics-commons-1.3.1.jar:
${oxygenInstallDir}/lib/commons-logging-1.0.4.jar:
${oxygenInstallDir}/lib/saxon9ee.jar:${oxygenInstallDir}/lib/
saxon9-dom.jar:
${oxygenInstallDir}/lib/xalan.jar:${oxygenInstallDir}/lib/
serializer.jar:
${oxygenInstallDir}/lib/resolver.jar:${oxygenInstallDir}/lib/
fop-pdf-images-1.3.jar:
${oxygenInstallDir}/lib/PDFBox-0.7.3.jar"
org.apache.fop.cli.Main -fo ${fo} -${method} ${out}
```

You need to add to the classpath `JimiProClasses.zip` for *JIMI* and `jai_core.jar`, `jai_codec.jar` and `mlibwrapper_jai.jar` for *JAI*. For the *JAI* package you can include the directory containing the native libraries (`mlib_jai.dll` and `mlib_jai_mmx.dll` on Windows) in the *PATH* system variable.

The Mac OS X version of the *JAI* library can be downloaded from *http://www.apple.com/downloads/macosx/apple/java3dandjavaadvancedimagingupdate.html*. In order to use it, install the downloaded package.

Other FO processors can be configured in *the **Preferences** dialog*.

Add a Font to the Built-in FOP - The Simple Version

If the font that must be set to Apache FOP is one of the fonts that are installed in the operating system you should follow the next steps for creating and setting a FOP configuration file that looks for the font that it needs in the system fonts. It is a simplified version of *the procedure for setting a custom font in Apache FOP*.

1. Register the font in FOP configuration. (not necessary in case of DITA PDF transformations, see next step)
   a) Create a FOP configuration file that specifies that FOP should look for fonts in the installed fonts of the operating system.

   ```
   <fop version="1.0">
     <renderers>
       <renderer mime="application/pdf">
         <fonts>
           <auto-detect/>
   ```

```
      </fonts>
    </renderer>
  </renderers>
</fop>
```

b) Set the FOP configuration file in **Preferences**.

   Go to menu **Options** > **Preferences** > **XML** > **XSLT/FO/XQuery** > **FO Processors** and enter the path of the FOP configuration file in the **Configuration file for the built-in FOP** text field.

2. Set the font on the document content.

This is done usually with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

- For DocBook documents you can start with the predefined scenario called **DocBook PDF**, *edit the XSLT parameters* and set the font name (in our example the font family name is **Arial Unicode MS**) to the parameters body.font.family and title.font.family.
- For TEI documents you can start with the predefined scenario called **TEI PDF**, *edit the XSLT parameters* and set the font name (in our example **Arial Unicode MS**) to the parameters bodyFont and sansFont.
- For DITA transformations using DITA-OT you should use an IDIOM FOP transformation and modify the following two files:
  - `${frameworks}/dita/DITA-OT/demo/fo/cfg/fo/font-mappings.xml` - the `font-face` element included in each element `physical-font` having the `attribute char-set="default"` must contain the name of the font (**Arial Unicode MS** in our example)
  - `${frameworks}/dita/DITA-OT/demo/fo/fop/conf/fop.xconf` - an element `auto-detect` must be inserted in the element `fonts` which is inside the element `renderer` having the attribute `mime="application/pdf"`:

```
<renderer mime="application/pdf">
  . . .
   <fonts>
       <auto-detect/>
   </fonts>
  . . .
</renderer>
```

Add a Font to the Built-in FOP

If an XML document is transformed to PDF using the built-in Apache FOP processor but it contains some Unicode characters that cannot be rendered by the default PDF fonts then a special font that is capable to render these characters must be configured and embedded in the PDF result.

Important: If the special font that must be set to Apache FOP is installed in the operating system there is a simple way of telling FOP to look for the font. See *the simplified procedure for adding a font to FOP*.

1. Locate the font.

First, you have to find out the name of a font that has the glyphs for the special characters you used. One font that covers the majority of characters, including Japanese, Cyrillic and Greek, is Arial Unicode MS.

On Windows the fonts are located into the `C:\Windows\Fonts` directory. On Mac they are placed in `/Library/Fonts`. To install a new font on your system is enough to copy it in the `Fonts` directory.

2. Generate a font metrics file from the font file.

a) Open a terminal.

b) Change the working directory to the Oxygen XML Editor plugin install directory.

c) Create the following script file in the Oxygen XML Editor plugin installation directory.

For Mac OS X and Linux create a file `ttfConvert.sh`:

```
#!/bin/sh
export LIB=lib
export 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 Editor 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">
  . . .
   <fonts>
       <font metrics-url="Arialuni.xml" kerning="yes"
           embed-url="file:/Library/Fonts/Arialuni.ttf">
           <font-triplet name="Arialuni" style="normal"
               weight="normal"/>
       </font>
   </fonts>
  . . .
</renderer>
```

XProc Transformations

This section explains how to configure and run XProc transformations in Oxygen XML Editor 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

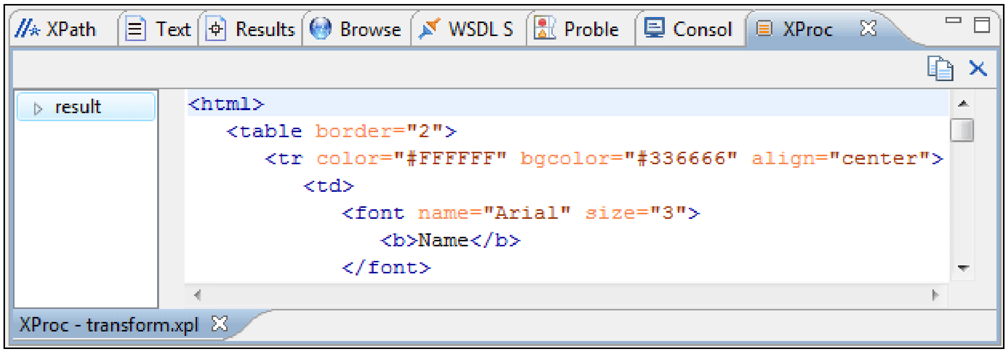
On the **XProc** tab of the scenario edit dialog you can select the URL of the XProc script and the XProc engine. The engine can be the built-in *Calabash* engine or a custom engine *configured in the **Preferences** dialog*.

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.



```
<html>
    <table border="2">
        <tr color="#FFFFFF" bgcolor="#336666" align="center">
            <td>
                <font name="Arial" size="3">
                    <b>Name</b>
                </font>
```

Figure 209: 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 18

Querying Documents

Topics:

This chapter shows how to query XML documents in Oxygen XML Editor plugin with XPath expressions and the XQuery language.

Running XPath Expressions

This section 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*.

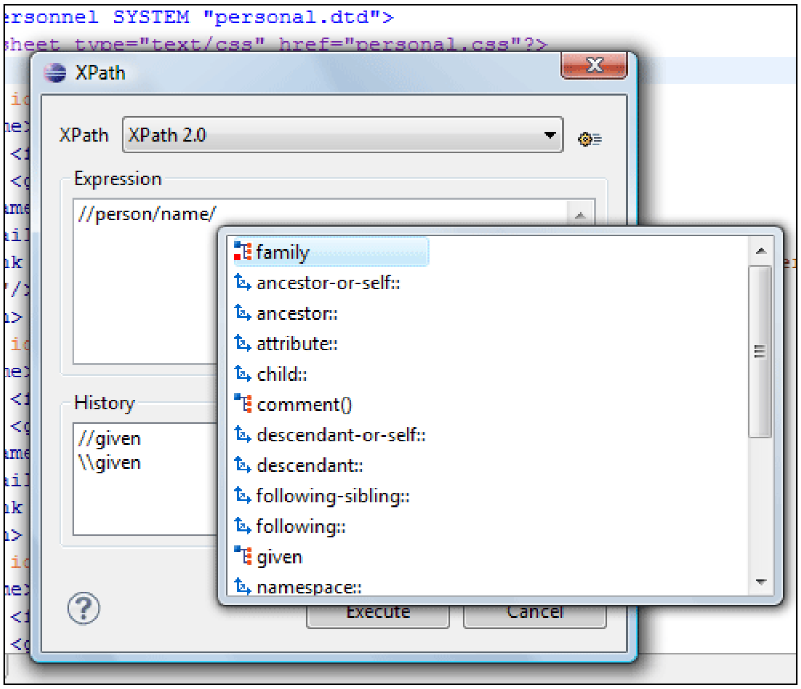
Oxygen's XPath Console

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:



XPath: XPath 2.0

Expression: //person/name/

History:
//given
\\given

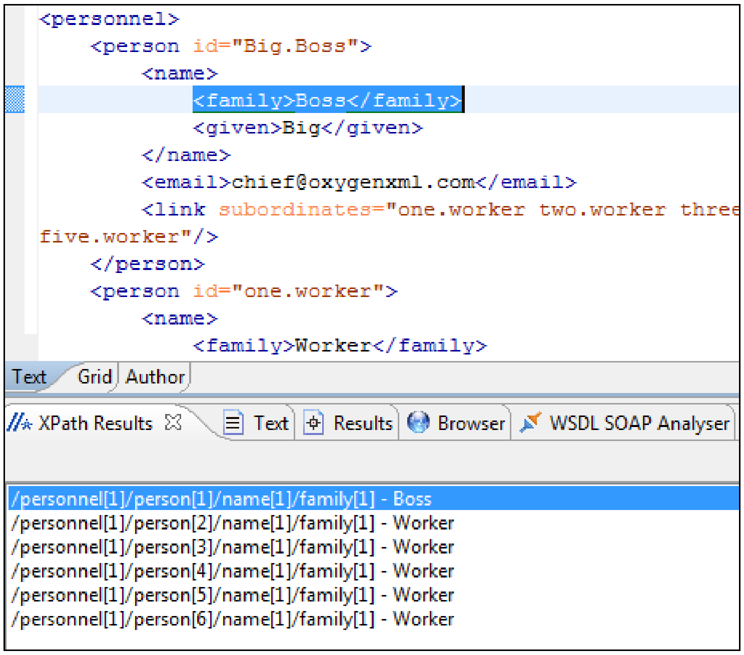
Figure 210: Content Completion in the XPath console

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 Editor 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/mydocsdir?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] -
```

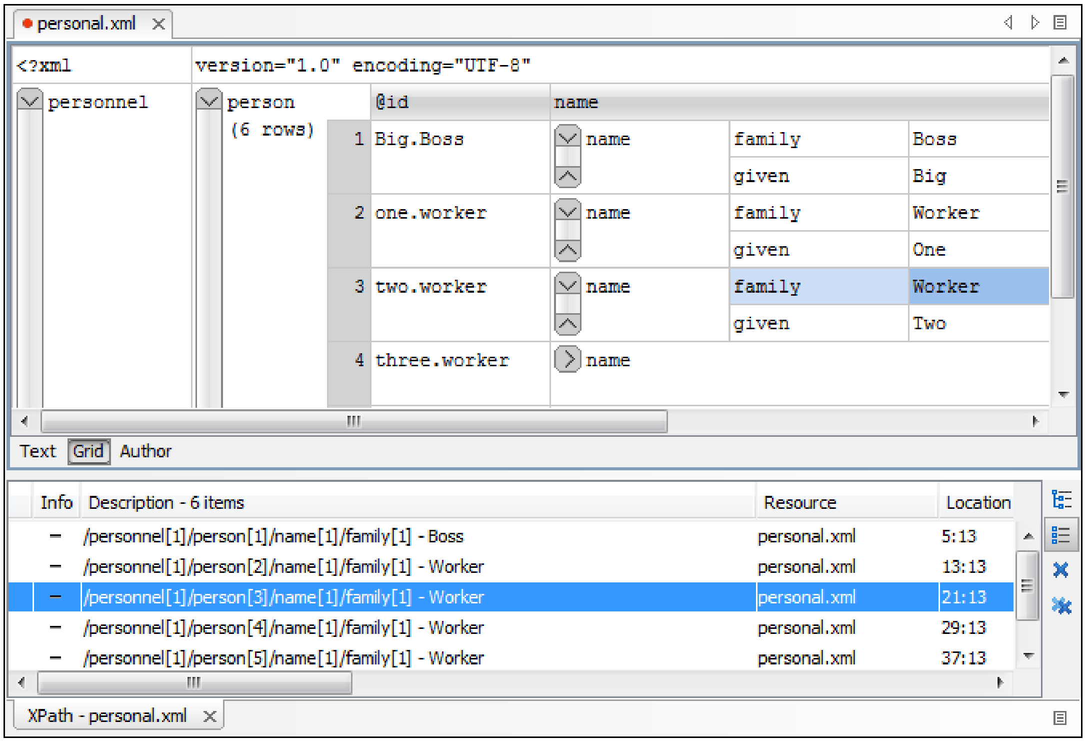


```
<personnel>
    <person id="Big.Boss">
        <name>
            <family>Boss</family>
            <given>Big</given>
        </name>
        <email>chief@oxygenxml.com</email>
        <link subordinates="one.worker two.worker three
five.worker"/>
    </person>
    <person id="one.worker">
    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

```
/personnel[1]/person[1]/name[1]/family[1] - Boss
/personnel[1]/person[2]/name[1]/family[1] - Worker
/personnel[1]/person[3]/name[1]/family[1] - Worker
/personnel[1]/person[4]/name[1]/family[1] - Worker
/personnel[1]/person[5]/name[1]/family[1] - Worker
/personnel[1]/person[6]/name[1]/family[1] - Worker
```

Figure 211: XPath results highlighted in editor panel with character precision

When using the *grid editor*, clicking a result record will highlight the entire node.



```
<?xml      version="1.0" encoding="UTF-8"
personnel  person      @id          name
           (6 rows)  1 Big.Boss     name  family  Boss
                                          given   Big
                     2 one.worker   name  family  Worker
                                          given   One
                     3 two.worker   name  family  Worker
                                          given   Two
                     4 three.worker name
```

Info	Description - 6 items	Resource	Location
–	/personnel[1]/person[1]/name[1]/family[1] - Boss	personal.xml	5:13
–	/personnel[1]/person[2]/name[1]/family[1] - Worker	personal.xml	13:13
–	/personnel[1]/person[3]/name[1]/family[1] - Worker	personal.xml	21:13
–	/personnel[1]/person[4]/name[1]/family[1] - Worker	personal.xml	29:13
–	/personnel[1]/person[5]/name[1]/family[1] - Worker	personal.xml	37:13

Figure 212: 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 Editor 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 Editor plugin .

What is XQuery

XQuery is the query language for XML and is officially defined by *a W3C Recommendation document*. The many benefits of XQuery include:

- XQuery allows you to work in one common model no matter what type of data you're working with: relational, XML, or object data.
- XQuery is ideal for queries that must represent results as XML, to query XML stored inside or outside the database, and to span relational and XML sources.
- XQuery allows you to create many different types of XML representations of the same data.
- XQuery allows you to query both relational sources and XML sources, and create one XML result.

Syntax Highlight and Content Completion

To *create a new XQuery document* select **File** > **New** **(Ctrl+N)** and when the **New** dialog appears select XQuery entry.

Oxygen XML Editor plugin provides syntax highlight for keywords and all known XQuery functions and operators. A content completion 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.

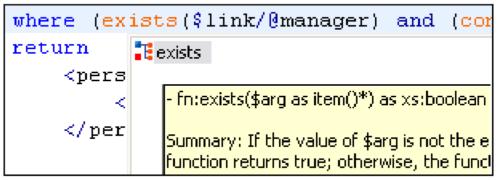
The extension functions built in the Saxon product are available on content completion if one of the following conditions are true:

- 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.



```
where (exists($link/@manager) and (cor
return
      <pers
          <
      </per
```

- fn:exists($arg as item()*) as xs:boolean

Summary: If the value of $arg is not the e
function returns true; otherwise, the funcl

Figure 213: 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.

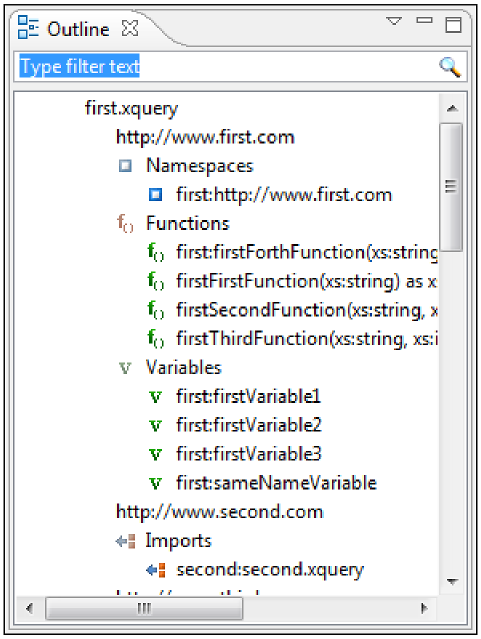


Figure 214: XQuery Outline View

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.

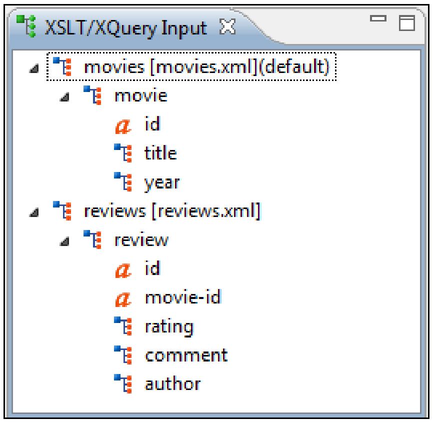


Figure 215: XQuery Input view

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>
```

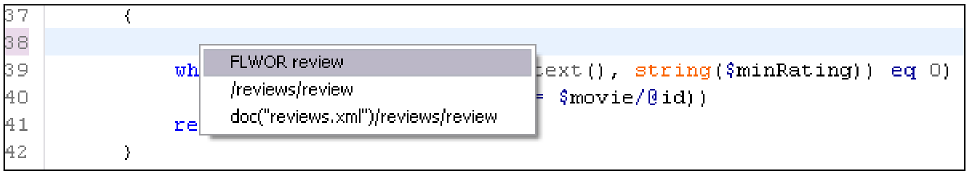
and

```
                    <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>
```

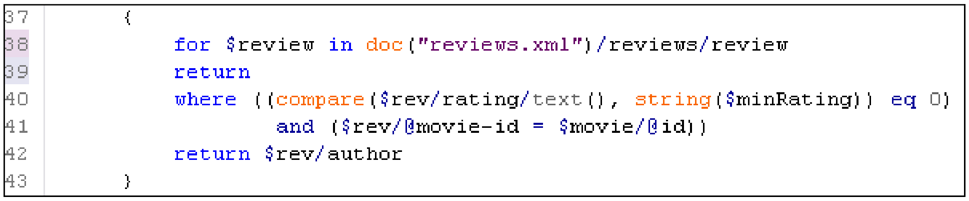
if you drag the **review** element and drop it between the braces a popup menu will be displayed.



```
        {

            wh                      text(), string($minRating)) eq 0)
                                    = $movie/@id))
            re
        }
```

Figure 216: XQuery Input drag and drop popup menu

Select **FLWOR rating** and the result document will be:

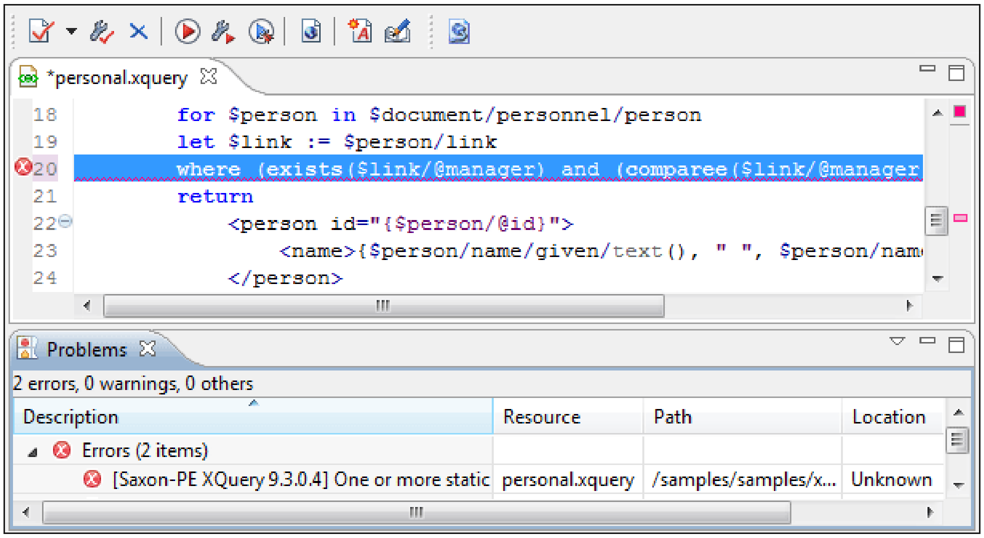


```
        {
            for $review in doc("reviews.xml")/reviews/review
            return
            where ((compare($rev/rating/text(), string($minRating)) eq 0)
                    and ($rev/@movie-id = $movie/@id))
            return $rev/author
        }
```

Figure 217: XQuery Input drag and drop result

XQuery Validation

With Oxygen XML Editor plugin you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.3.0.5 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.



```
        for $person in $document/personnel/person
        let $link := $person/link
        where (exists($link/@manager) and (comparee($link/@manager
        return
            <person id="{$person/@id}">
                <name>{$person/name/given/text(), " ", $person/nam
            </person>
```

2 errors, 0 warnings, 0 others

Description	Resource	Path	Location
Errors (2 items)			
[Saxon-PE XQuery 9.3.0.4] One or more static	personal.xquery	/samples/samples/x...	Unknown

Figure 218: 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* togheter 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 Editor 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:

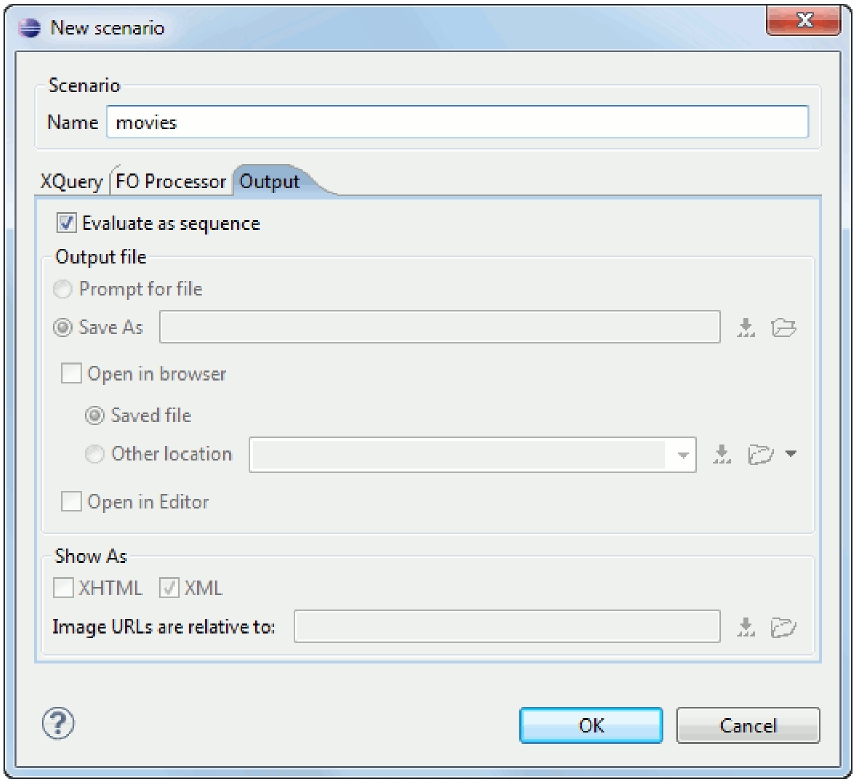
1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for this connection.
4. Select one of the previously configured *XQJ data sources* in the **Data Source** combo box.
5. Fill-in the connection details.

   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.



Scenario

Name: movies

☑ Evaluate as sequence

Output file

○ Prompt for file

● Save As

□ Open in browser

● Saved file

○ Other location

□ Open in Editor

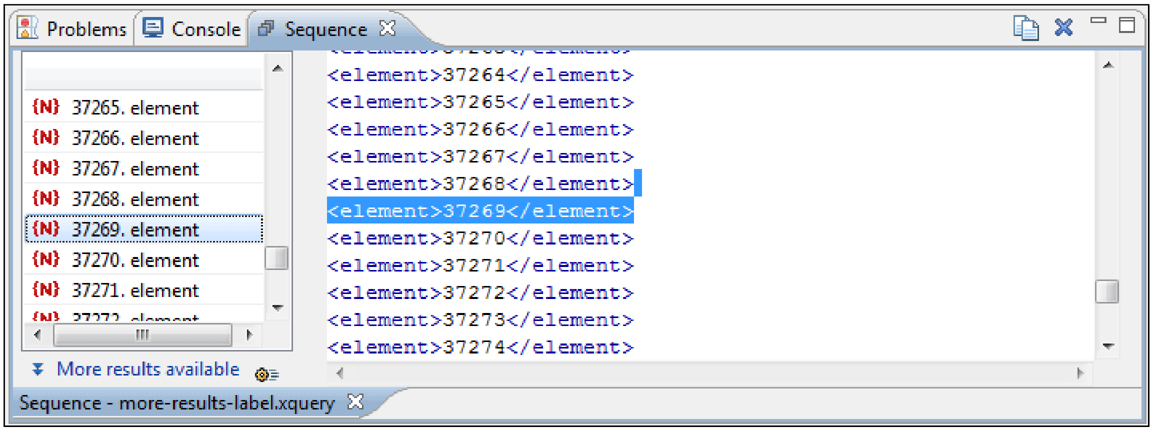
Show As

□ XHTML ☑ XML

Image URLs are relative to:

Figure 219: The XQuery transformation result displayed in Sequence view

A chunk of the XQuery transformation result is displayed in the **Sequence** view.



```
{N} 37265. element
{N} 37266. element
{N} 37267. element
{N} 37268. element
{N} 37269. element
{N} 37270. element
{N} 37271. element
```

```
<element>37264</element>
<element>37265</element>
<element>37266</element>
<element>37267</element>
<element>37268</element>
<element>37269</element>
<element>37270</element>
<element>37271</element>
<element>37272</element>
<element>37273</element>
<element>37274</element>
```

Figure 220: 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 Editor plugin offers support for XQuery Update 1.0. The XQuery Update Facility provides expressions that can be used to make persistent changes to instances of the XQuery 1.0 and XPath 2.0 Data Model. Thus, besides querying XML documents, you can modify them using the various insert/delete/modify/create methods available in the *XQuery Update 1.0* standard.

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 Editor plugin will notify you if the update was successful.

Using XQuery Update to modify a tag name in an XML file

```
rename node doc("books.xml")//publisher[1]//book[1] as "firstBook"
```

Chapter 19

Debugging XSLT Stylesheets and XQuery Documents

Topics:

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:

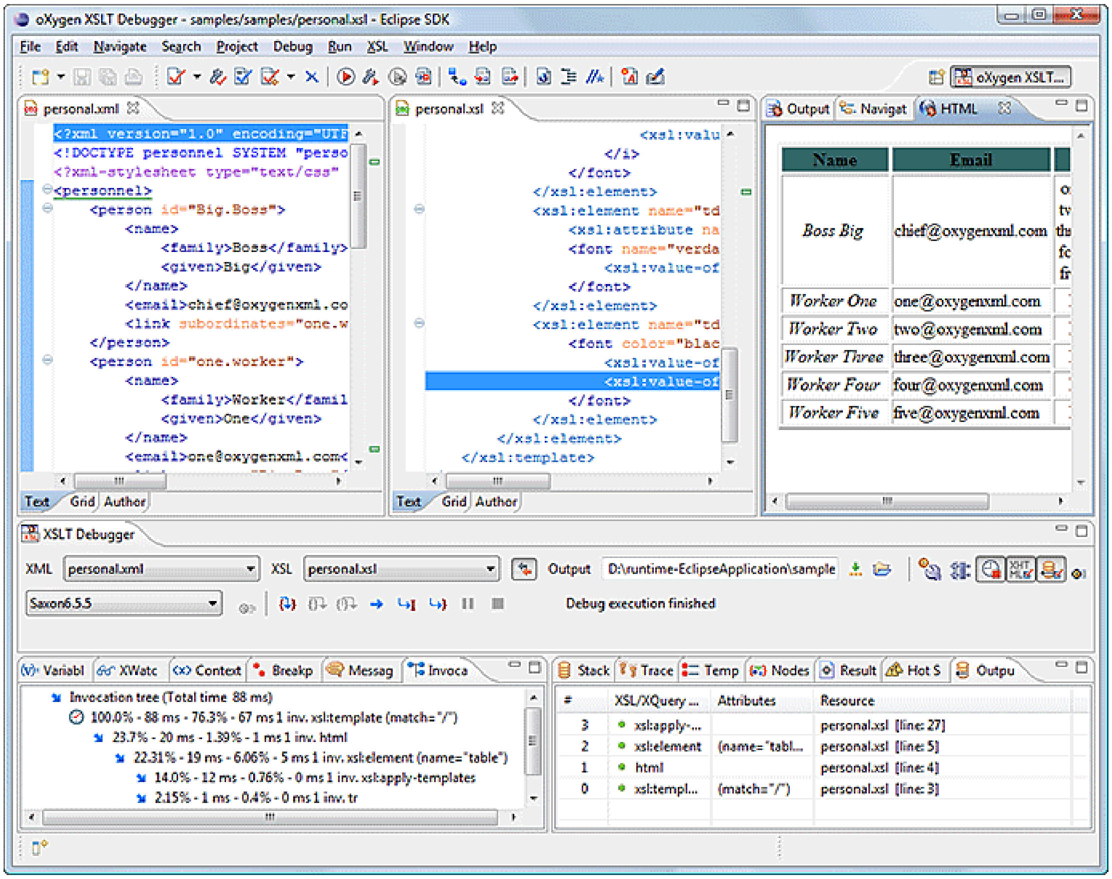
- Support for XSLT 1.0 stylesheets (using Saxon 6.5.5 and Xalan XSLT engines), XSLT 2.0 stylesheets and XPath 2.0 expressions that are included in the stylesheets (using Saxon 9.3.0.5 XSLT engine) and XQuery 1.0 (using Saxon 9.3.0.5 XQuery engine).
- Stepping capabilities: step in, step over, step out, 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 other modes (*Author mode*, *Grid mode*) are available only in *the Editor perspective*.

The debugger perspective contains four sections:

- **Source document view (XML)** - Displays and allows the editing of XML files (documents).
- **XSLT/XQuery document view (XSLT/XQuery)** - Displays and allows the editing of XSL files(stylesheets) or XQuery documents.
- **Output document view** - Displays the output that results from inputing a document (XML) and astylesheet (XSL) or XQuery document in the transformer. The transformation result is written dynamically while the transformation is processed.
- **Control view** - The control view is used to configure and control the debugging operations. It also provides a set of *Information views* types. This pane has two sections:
    - *Control toolbar*
    - *Information views*



```
<?xml version="1.0" encoding="UTF
<!DOCTYPE personnel SYSTEM "perso
<?xml-stylesheet type="text/css"
<personnel>
    <person id="Big.Boss">
        <name>
            <family>Boss</family>
            <given>Big</given>
        </name>
        <email>chief@oxygenxml.co
        <link subordinates="one.w
    </person>
    <person id="one.worker">
        <name>
            <family>Worker</famil
            <given>One</given>
        </name>
        <email>one@oxygenxml.com<
```

```
                        <xsl:valu
                    </i>
                </font>
            </xsl:element>
            <xsl:element name="td
                <xsl:attribute na
                <font name="verda
                    <xsl:value-of
                </font>
            </xsl:element>
            <xsl:element name="td
                <font color="blac
                    <xsl:value-of
                    <xsl:value-of
                </font>
            </xsl:element>
        </xsl:element>
    </xsl:template>
```

Name	Email
Boss Big	chief@oxygenxml.com
Worker One	one@oxygenxml.com
Worker Two	two@oxygenxml.com
Worker Three	three@oxygenxml.com
Worker Four	four@oxygenxml.com
Worker Five	five@oxygenxml.com

XML personal.xml XSL personal.xsl Output D:\runtime-EclipseApplication\sample

Saxon6.5.5 Debug execution finished

```
Invocation tree (Total time 88 ms)
  100.0% - 88 ms - 76.3% - 67 ms 1 inv. xsl:template (match="/")
    23.7% - 20 ms - 1.39% - 1 ms 1 inv. html
      22.31% - 19 ms - 6.06% - 5 ms 1 inv. xsl:element (name="table")
        14.0% - 12 ms - 0.76% - 0 ms 1 inv. xsl:apply-templates
        2.15% - 1 ms - 0.4% - 0 ms 1 inv. tr
```

#	XSL/XQuery ...	Attributes	Resource
3	xsl:apply-...		personal.xsl [line: 27]
2	xsl:element	(name="tabl...	personal.xsl [line: 5]
1	html		personal.xsl [line: 4]
0	xsl:templ...	(match="/")	personal.xsl [line: 3]

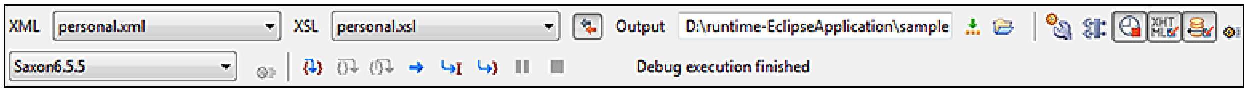
Figure 221: Debugger Mode Interface

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.



XML personal.xml XSL personal.xsl Output D:\runtime-EclipseApplication\sample

Saxon6.5.5 Debug execution finished

Figure 222: 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.

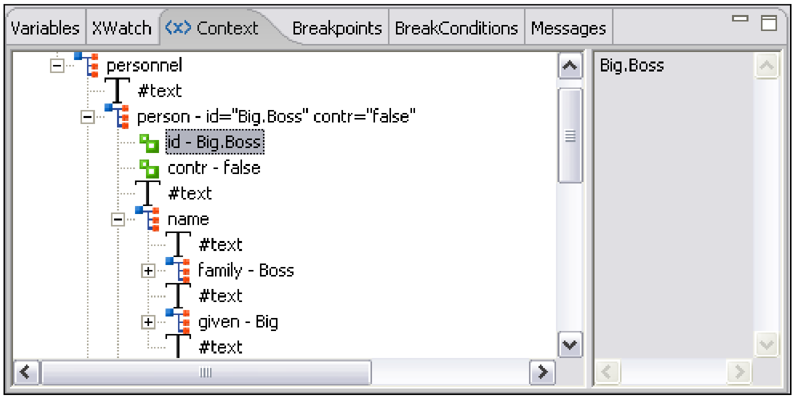


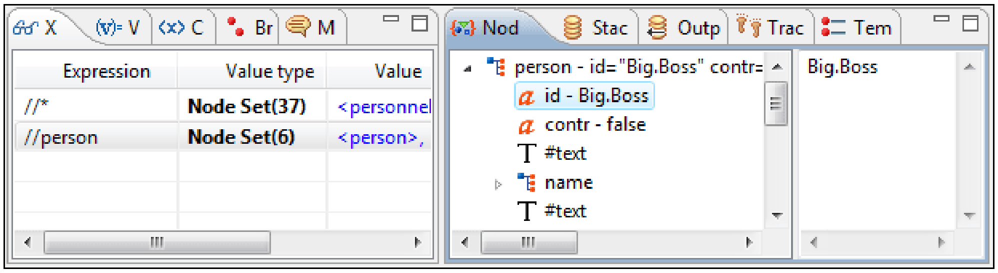
Figure 223: The Context node view

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*.



Expression	Value type	Value
//*	Node Set(37)	<personnel
//person	Node Set(6)	<person>,

Figure 224: The XPath watch view

Table 2: XWatch columns

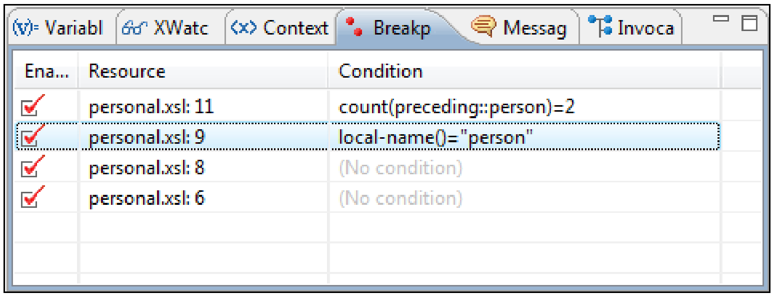
Column	Description
Expression	XPath expression to be evaluated (should be XPath 1.0 or 2.0 compliant).
Value	Result of XPath expression evaluation. Value has a type (see *the possible values* in the section *Variables View* on page 432). For Node Set results the number of nodes in the set is shown in 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
☑	personal.xsl: 11	count(preceding::person)=2
☑	personal.xsl: 9	local-name()="person"
☑	personal.xsl: 8	(No condition)
☑	personal.xsl: 6	(No condition)

Figure 225: The Breakpoints View

Table 3: 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:

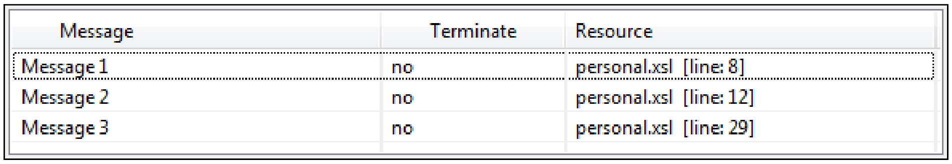
- For example if the breakpoint is set on an empty line or commented line or the line is not reached by the processor (no template to match it, line containing only an end tag), that breakpoint is invalid.
- Clicking a record highlights the breakpoint line into the document.
- The breakpoints list is 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.xsl [line: 8]
Message 2	no	personal.xsl [line: 12]
Message 3	no	personal.xsl [line: 29]

Figure 226: The Messages View

Table 4: Messages columns

Column	Description
Message	Message content.
Terminate	Signals if processor terminates the transformation or not once it encounters the message (yes/no respectively)
Resource	Resource file where `xsl:message` instruction is defined and the message line number.

The following actions are available in the contextual menu:

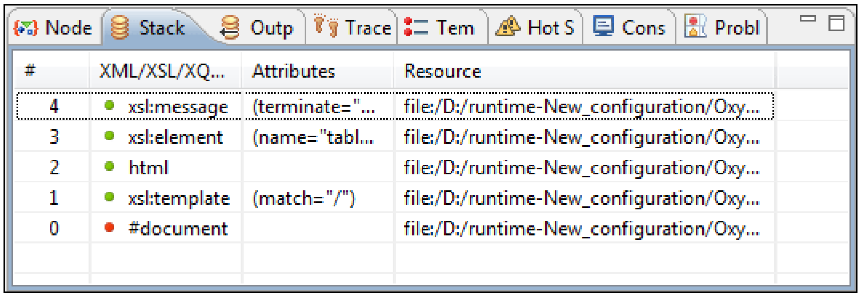
- **Go to** - Highlight the XSL fragment that generated the message.
- **Copy Value** - Copies to clipboard message details (system ID, severity info, description, start location, terminate state).

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 Editor plugin shows both node types into one common stack. The source (XML) nodes are preceded by a red color icon while XSLT/XQuery nodes are preceded by a green color icon. The advantage of this approach is that you can always see the source scope on which a XSLT/XQuery instruction is executed (the last red color node on the stack). The stack is oriented upside down.



#	XML/XSL/XQ...	Attributes	Resource
4	● xsl:message	(terminate="...	file:/D:/runtime-New_configuration/Oxy...
3	● xsl:element	(name="tabl...	file:/D:/runtime-New_configuration/Oxy...
2	● html		file:/D:/runtime-New_configuration/Oxy...
1	● xsl:template	(match="/")	file:/D:/runtime-New_configuration/Oxy...
0	● #document		file:/D:/runtime-New_configuration/Oxy...

Figure 227: 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 5: Stack columns

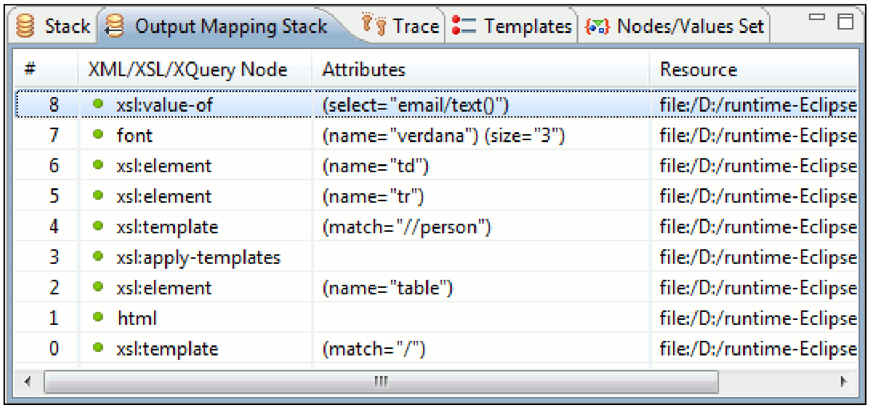
Column	Description
#	Order number, represents the depth of the node (0 is the stack base).
XML/XSLT/XQuery Node	Node from source or stylesheet document currently being processed. One particular stack node is the document root, noted as **#document**.
Attributes	Attributes of the node (a list of `id="value"` pairs).
Resource	Resource file where the node is located.

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 228: 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 6: 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.

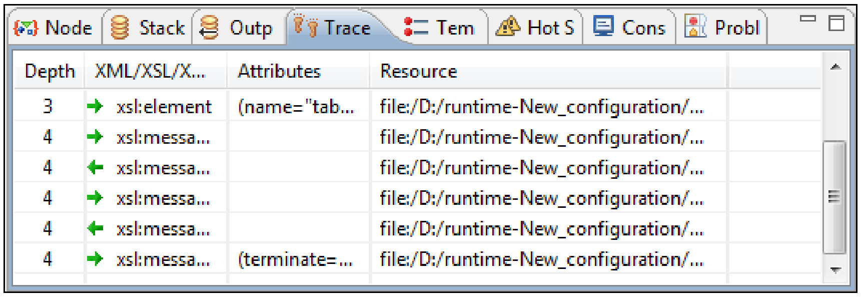
Trace History View

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.



Depth	XML/XSL/X...	Attributes	Resource
3	→ xsl:element	(name="tab...	file:/D:/runtime-New_configuration/...
4	→ xsl:messa...		file:/D:/runtime-New_configuration/...
4	← xsl:messa...		file:/D:/runtime-New_configuration/...
4	→ xsl:messa...		file:/D:/runtime-New_configuration/...
4	← xsl:messa...		file:/D:/runtime-New_configuration/...
4	→ xsl:messa...	(terminate=...	file:/D:/runtime-New_configuration/...

Figure 229: 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 7: Trace History columns

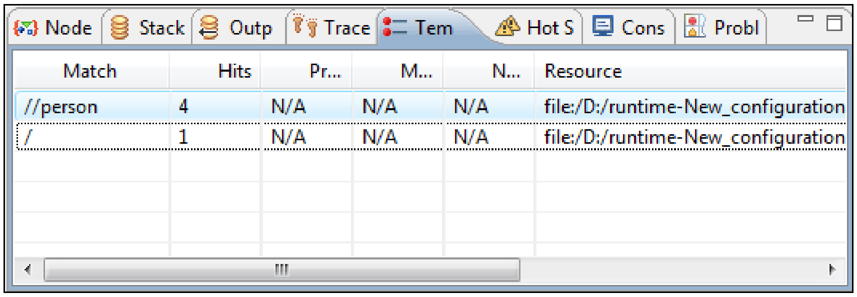
Column	Description
Depth	Shows you how deep the node is nested in the XML or stylesheet structure. The bigger the number, the more nested the node is. A depth 0 node is the document root.
XML/XSLT/XQuery Node	Represents the node from the processed source or stylesheet document. One particular node is the document root, noted as `#document`. Every node is preceded by an arrow that represents what action was performed on it (entering or leaving the node).
Attributes	Attributes of the node (a list of `id="value"` pairs).
Resource	Resource file where the node is located.

Important: Remarks:

- Clicking a record highlights that node's location inside the resource.
- Only the Saxon processor shows the element attributes.
- The Xalan processor shows also the built-in rules.

Templates View

The `xsl:template` is the basic element for stylesheets transformation. This view is only available during XSLT debugging sessions and shows all `xsl:template` instructions used by the transformation. By seeing the number of hits for each of the templates you get an idea of the stylesheet coverage by template rules with respect to the input source.



Match	Hits	Pr...	M...	N...	Resource
//person	4	N/A	N/A	N/A	file:/D:/runtime-New_configuration
/	1	N/A	N/A	N/A	file:/D:/runtime-New_configuration

Figure 230: 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 8: Templates columns

Column	Description
Match	The `match` attribute of the `xsl:template`.
Hits	The number of hits for the `xsl:template`. Shows how many times the XSLT processor used this particular template.
Priority	The template priority as established by XSLT processor.
Mode	The `mode` attribute of the `xsl:template`.
Name	The `name` attribute of the `xsl:template`.
Resource	The resource file where the template is located.

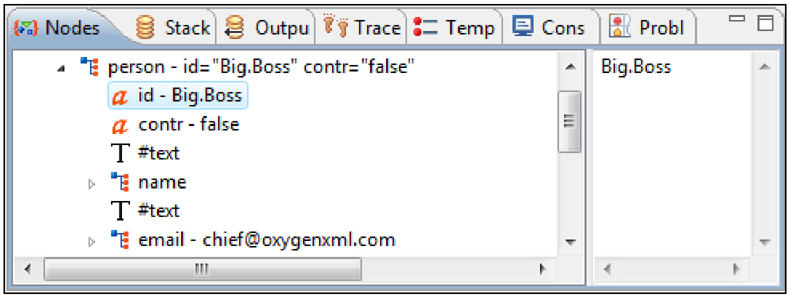
Important: Remarks:

- Clicking a record highlights that template definition inside the resource.
- Saxon only shows the applied templates having at least one hit from the processor. Xalan shows all defined templates, with or without hits.
- Template table values can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in rules.

Node Set View

This view is always used in relation with *The **Variables** view* and *the **XWatch** view*l. It shows an XSLT node set value in a tree form. The node set view is updated as response to the following events:

- You click a variable having a node set value in one of the above 2 views.
- You click a tree fragment in one of the above 2 views.
- You click an XPath expression evaluated to a node set in one of the above 2 views.



person - id="Big.Boss" contr="false"
id - Big.Boss
contr - false
#text
name
#text
email - chief@oxygenxml.com

Big.Boss

Figure 231: The Node Set view

The nodes / values set is presented in a tree-like fashion. Nodes from a defined namespace bound to a prefix are displayed using the qualified name. If the namespace is not bound to a prefix the namespace URI is presented before the node name. The value of the selected attribute or node is shown in the right side panel.

Important: Remarks:

- In case of longer values in the right side panel the interface shows three suspension points (...) at the end. A more detailed value is available as tooltip.
- Clicking a record highlights the location of that node into the source or stylesheet view.

Variables View

Variables and parameters play an important role during an XSLT/XQuery transformation. Oxygen XML Editor plugin uses the following icons to differentiate variables and parameters:

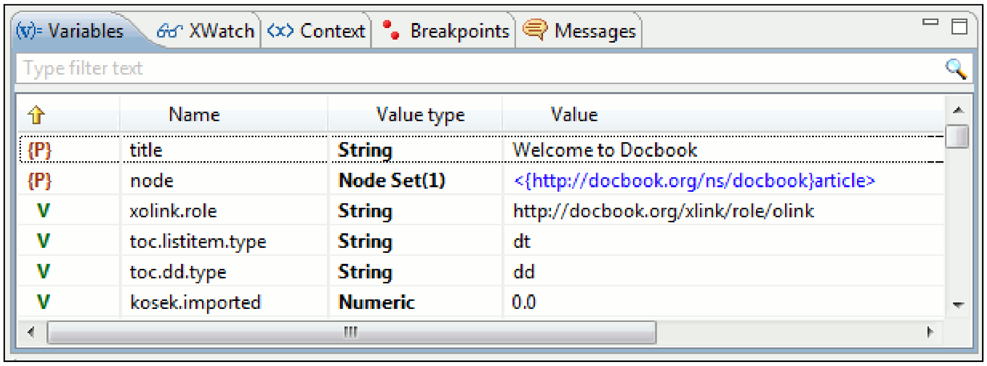
- V - Global variable.
- {V} - Local variable.
- P - Global parameter.
- {P} - Local parameter.

The following value types are available:

- **Boolean**
- **String**
- **Date** - XSLT 2.0 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 232: The Variables View

Table 9: 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 Editor plugin XML perspective switch to the Oxygen XML Editor plugin XSLT Debugger perspective or the Oxygen XML Editor plugin XQuery Debugger perspective with one of the actions (here explained for XSLT):

   - Menu **Window** > **Open Perspective** > **Other ...** > **Oxygen XSLT Debugger**
   - The toolbar button **Debug scenario** - This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer engine, the XSLT parameters, the transformer extensions, etc) will be saved back in the scenario when exiting from the Debugger perspective.

3. Select the source XML document in the XML source selector of *the Control toolbar*. In case of XQuery debugging if your XQuery document has no implicit source set the source selector value to **NONE**.
4. Select the XSLT/XQuery document in the XSLT/XQuery selector of *the Control toolbar*.
5. Set XSLT/XQuery parameters from the button available on *the Control toolbar*.
6. *Set one or more breakpoints*.
7. Step through the stylesheet using the buttons available on *the Control toolbar*:

- **Step into**
- **Step over**
- **Step out**
- **Run**
- **Run to cursor**
- **Run to end**
- **Pause**
- **Stop**

8. Examine the information in the Information views to find the bug in the transformation process.

   You may find *the procedure for determining the XSLT template/XQuery element that generated an output section* useful for fixing bugs in the transformation.

Using Breakpoints

The Oxygen XML Editor plugin XSLT/XQuery Debugger allows you to interrupt XSLT/XQuery processing to gather information about variables and processor execution at particular points. To ensure breakpoints persistence between work sessions, they are saved at project level. You can set maximum 100 breakpoints per project.

Inserting Breakpoints

To insert a breakpoint, follow these steps:

1. 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.

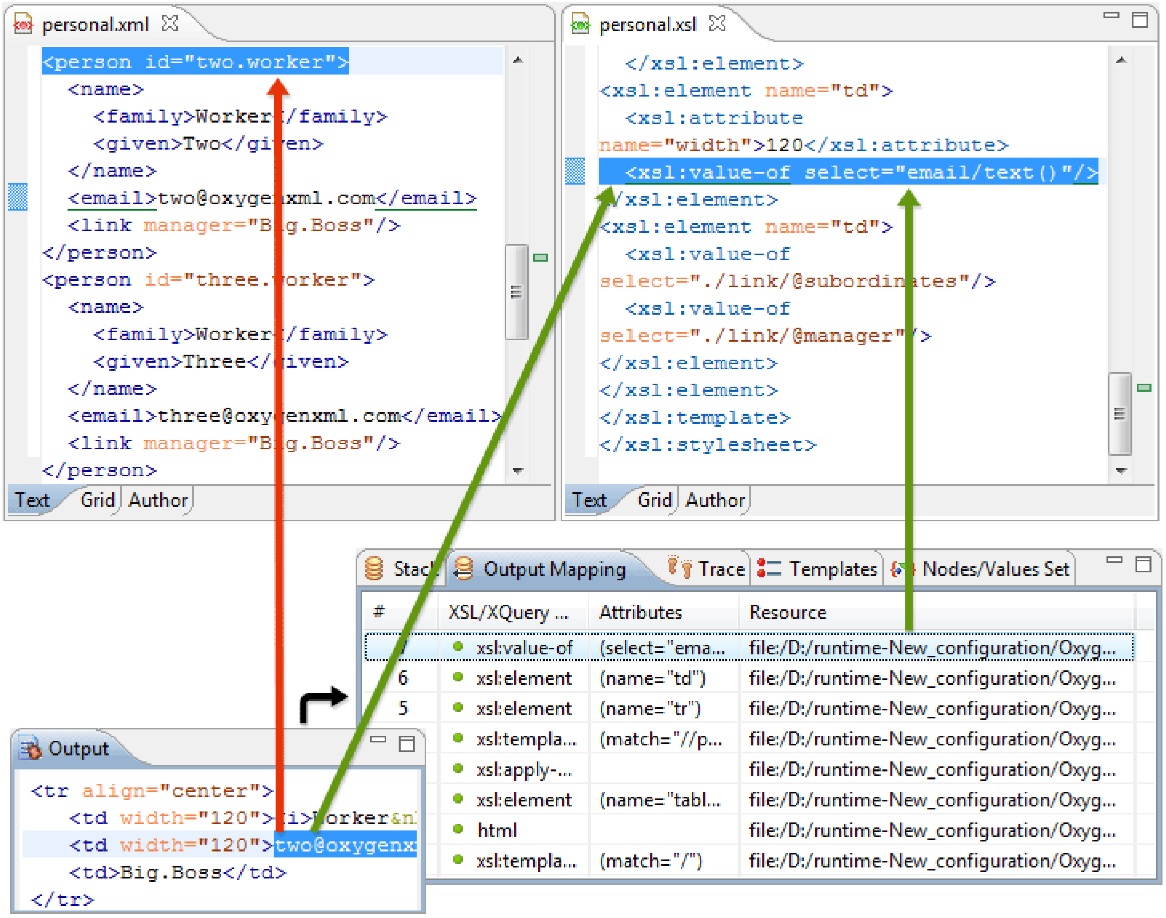
Output to source mapping is a powerful feature that makes this output to source mapping persistent. You can click on the text from the **Text** output view or **XHTML** output view and the editor will select the XML source context and the XSLT template/XQuery element that generated the text. Also inspecting 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 Editor plugin XSLT Debugger perspective or Oxygen XML Editor plugin XQuery Debugger perspective with one of the actions (here explained for XSLT):

   - Go to menu **Window** > **Open Perspective** > **Other ...** > **Oxygen XSLT Debugger**
   - Go to menu . The toolbar button **Debug scenario** . This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer

engine, the XSLT parameters, the transformer extensions, etc) will be saved back in the scenario when exiting from the Debugger perspective.

2. Select the source XML document in the XML source selector of *the Control toolbar*. In case of XQuery debugging without an implicit source choose the NONE value.
3. Select the XSLT / XQuery document in the XSLT / XQuery selector of *the Control toolbar*.
4. Select the XSLT / XQuery engine in the XSLT / XQuery engine selector of *the Control toolbar*.
5. Set XSLT / XQuery parameters from the button available on *the Control toolbar*.
6. Apply the XSLT stylesheet or XQuery transformation using the button ↪ **Run to end** available on *the Control toolbar*.
7. Inspect the mapping by clicking a section of the output from the **Output** view of the Oxygen XML Editor plugin XSLT Debugger or Oxygen XML Editor plugin XQuery Debugger perspectives.



```
<person id="two.worker">
  <name>
    <family>Worker</family>
    <given>Two</given>
  </name>
  <email>two@oxygenxml.com</email>
  <link manager="Big.Boss"/>
</person>
<person id="three.worker">
  <name>
    <family>Worker</family>
    <given>Three</given>
  </name>
  <email>three@oxygenxml.com</email>
  <link manager="Big.Boss"/>
</person>
```

```
  </xsl:element>
<xsl:element name="td">
  <xsl:attribute
name="width">120</xsl:attribute>
  <xsl:value-of select="email/text()"/>
</xsl:element>
<xsl:element name="td">
  <xsl:value-of
select="./link/@subordinates"/>
  <xsl:value-of
select="./link/@manager"/>
</xsl:element>
</xsl:element>
</xsl:template>
</xsl:stylesheet>
```

#	XSL/XQuery ...	Attributes	Resource
7	● xsl:value-of	(select="ema...	file:/D:/runtime-New_configuration/Oxyg...
6	● xsl:element	(name="td")	file:/D:/runtime-New_configuration/Oxyg...
5	● xsl:element	(name="tr")	file:/D:/runtime-New_configuration/Oxyg...
	● xsl:templa...	(match="//p...	file:/D:/runtime-New_configuration/Oxyg...
	● xsl:apply-...		file:/D:/runtime-New_configuration/Oxyg...
	● xsl:element	(name="tabl...	file:/D:/runtime-New_configuration/Oxyg...
	● html		file:/D:/runtime-New_configuration/Oxyg...
	● xsl:templa...	(match="/")	file:/D:/runtime-New_configuration/Oxyg...

```
<tr align="center">
   <td width="120"><i>Worker&n
   <td width="120">two@oxygenx
   <td>Big.Boss</td>
</tr>
```

Figure 233: 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 20

Profiling XSLT Stylesheets and XQuery Documents

Topics:

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.

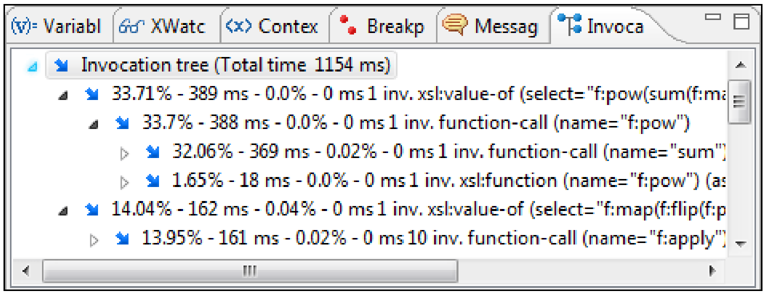
Enabling and disabling the profiler is controlled by the ⏲ *Profiler button* from the *debugger control toolbar*. The XSLT/XQuery profiler is off by default. This option is not available during a debugger session so you should set it before starting the transformation.

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.



```
Invocation tree (Total time 1154 ms)
    33.71% - 389 ms - 0.0% - 0 ms 1 inv. xsl:value-of (select="f:pow(sum(f:ma
        33.7% - 388 ms - 0.0% - 0 ms 1 inv. function-call (name="f:pow")
            32.06% - 369 ms - 0.02% - 0 ms 1 inv. function-call (name="sum")
            1.65% - 18 ms - 0.0% - 0 ms 1 inv. xsl:function (name="f:pow") (as
    14.04% - 162 ms - 0.04% - 0 ms 1 inv. xsl:value-of (select="f:map(f:flip(f:p
        13.95% - 161 ms - 0.02% - 0 ms 10 inv. function-call (name="f:apply")
```

Figure 234: Invocation tree view

The entries in the invocation tree have different meanings which are indicated by the displayed icons:

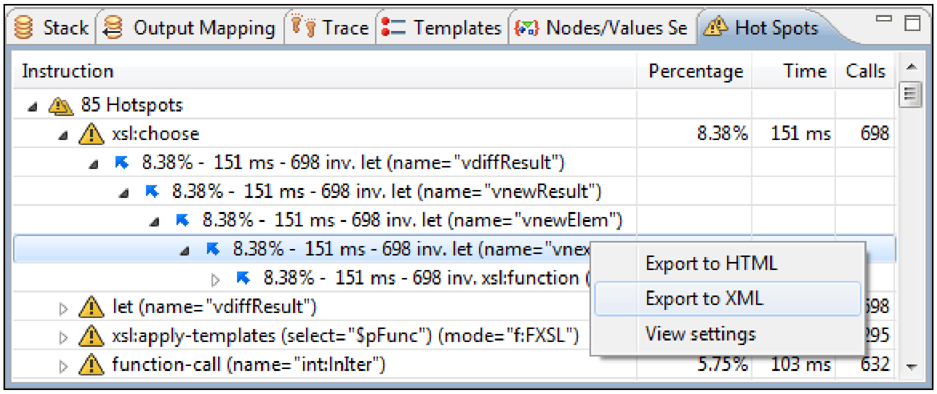
- ↘ - Points to a call whose inherent time is insignificant compared to its call tree time.
- ⏲ - Points to a call whose inherent time is significant compared to its call tree time (greater than 1/3rd of its call tree time).

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*** .



Instruction	Percentage	Time	Calls
85 Hotspots			
xsl:choose	8.38%	151 ms	698
8.38% - 151 ms - 698 inv. let (name="vdiffResult")			
8.38% - 151 ms - 698 inv. let (name="vnewResult")			
8.38% - 151 ms - 698 inv. let (name="vnewElem")			
8.38% - 151 ms - 698 inv. let (name="vnex			
8.38% - 151 ms - 698 inv. xsl:function (			
let (name="vdiffResult")			98
xsl:apply-templates (select="$pFunc") (mode="f:FXSL")			95
function-call (name="int:lnIter")	5.75%	103 ms	632

Figure 235: 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*).

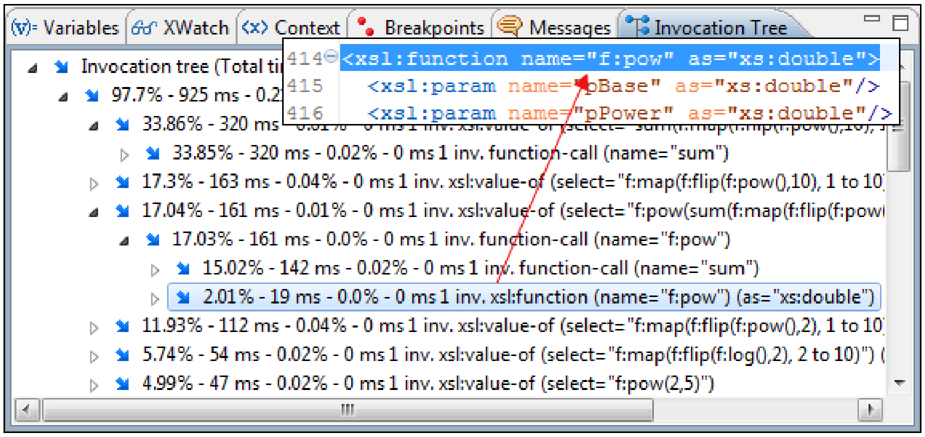
Immediately after turning the profiler on two new information views are added to the current debugger *information views*:

- ***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.



```
<xsl:function name="f:pow" as="xs:double">
   <xsl:param name="pBase" as="xs:double"/>
   <xsl:param name="pPower" as="xs:double"/>
```

```
Invocation tree (Total ti
  97.7% - 925 ms - 0.2
    33.86% - 320 ms
      33.85% - 320 ms - 0.02% - 0 ms 1 inv. function-call (name="sum")
    17.3% - 163 ms - 0.04% - 0 ms 1 inv. xsl:value-of (select="f:map(f:flip(f:pow(),10), 1 to 10
    17.04% - 161 ms - 0.01% - 0 ms 1 inv. xsl:value-of (select="f:pow(sum(f:map(f:flip(f:pow
      17.03% - 161 ms - 0.0% - 0 ms 1 inv. function-call (name="f:pow")
        15.02% - 142 ms - 0.02% - 0 ms 1 inv. function-call (name="sum")
        2.01% - 19 ms - 0.0% - 0 ms 1 inv. xsl:function (name="f:pow") (as="xs:double")
    11.93% - 112 ms - 0.04% - 0 ms 1 inv. xsl:value-of (select="f:map(f:flip(f:pow(),2), 1 to 10
    5.74% - 54 ms - 0.02% - 0 ms 1 inv. xsl:value-of (select="f:map(f:flip(f:log(),2), 2 to 10)") (
    4.99% - 47 ms - 0.02% - 0 ms 1 inv. xsl:value-of (select="f:pow(2,5)")
```

Figure 236: Source backmapping

In any of the above views you can use the backmapping feature in order to find the XSLT stylesheet or XQuery expression definition. Clicking on the selected item cause Oxygen XML Editor plugin to highlight the XSLT stylesheet or XQuery expression source line where the instruction is defined.

When navigating through the trees by opening instruction calls, Oxygen XML Editor plugin automatically expands instructions which are only called by one other instruction themselves.

The profiling data can be saved into XML and HTML format. On any view you should right click , use the pop-up menu and select the corresponding choice. Basically saving HTML means saving XML and applying an XSLT stylesheet to render the report as XML. These stylesheets are included in the Oxygen XML Editor plugin distribution (see the subfolder `frameworks/profiler/` of the Oxygen XML Editor plugin installation folder) so you can make your own report based on the profiling raw data.

If you like to change the *XSLT/XQuery profiler settings* you should right click on view, use the pop-up menu and choose the corresponding **View settings** entry.

Caution: Profiling exhaustive transformation may run into an OutOfMemory error due to the large amount of information being collected. If this is the case you can close unused projects when running the profiling or use high values for Java VM options -Xms and -Xmx. If this does not help you can shorten your source xml file and try again.

Chapter 21

Working with Archives

Topics:

Oxygen XML Editor plugin offers the means to manipulate files directly from ZIP type archives. By manipulation one should understand opening and saving files directly in archives, browsing and modifying archive structures. The archive support is available for all ZIP-type archives, which includes:

- ZIP archives
- EPUB books
- JAR archives
- Office Open XML (OOXML) files
- Open Document Format (ODF) files
- IDML files

This means that you can modify, transform, validate files directly from OOXML or ODF packages. The structure and content of an EPUB book, OOXML file or ODF file *can be opened, edited and saved* as for any other ZIP archive.

You can transform, validate and perform many other operations on files directly from an archive. When selecting an URL for a specific operation like transformation or validation you can click the **Browse for archived file** button to navigate and choose the file from a certain archive.

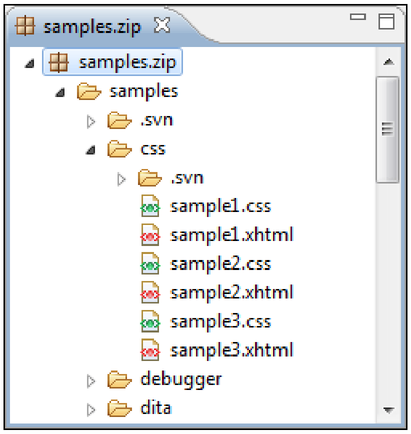
Browsing and Modifying Archive Structure

You can navigate archives in the **Archives Browser** either by opening them from the **Navigator** or by using the integration with the Eclipse File System. For the EFS (Eclipse File System) integration you must right click the archive in the **Navigator** and choose **Expand Zip Archive**. All the standard Eclipse **Navigator** actions are available on the mounted archive. If you decide to close the archive you can use the **Collapse ZIP Archive** action located in the contextual menu for the expanded archive. Any file opened from the archive expanded in the EFS will be closed when the archive in 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 Editor plugin as a supported archive type, you can add it from the *Archive preferences page*.



```
samples.zip
  samples.zip
    samples
      .svn
      css
        .svn
        sample1.css
        sample1.xhtml
        sample2.css
        sample2.xhtml
        sample3.css
        sample3.xhtml
      debugger
      dita
```

Figure 237: Browsing an Archive

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 Editor 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.

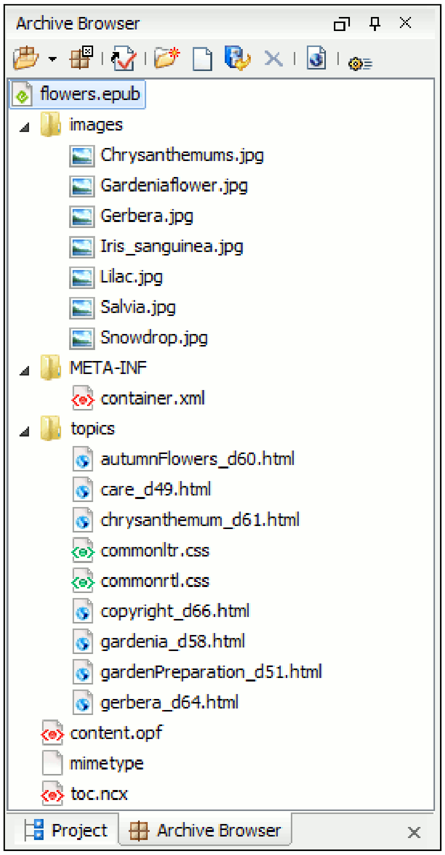


Figure 238: EPUB file displayed in the Archive Browser view

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 Editor 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 Editor 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 22

Working with Databases

Topics:

XML is a storage and interchange format for structured data and it is supported by all major database systems. Oxygen XML Editor plugin offers the means of managing the interaction with some of the widely used databases, both relational ones and Native XML Databases. By interaction, one should understand browsing, querying, SQL execution support, content editing, importing from databases, generating XML Schema from database structure.

Relational Database Support

Relational databases use a relational model and are based on tables linked by a common key. Oxygen XML Editor plugin offers support for the following relational databases: IBM DB2, JDBC-ODBC Bridge, MySQL, Microsoft SQL Server, Oracle 11g:

- browsing the tables of these types of database in the **Data Source Explorer** view
- executing SQL queries against them
- calling stored procedures with input and output parameters

Configuring Database Data Sources

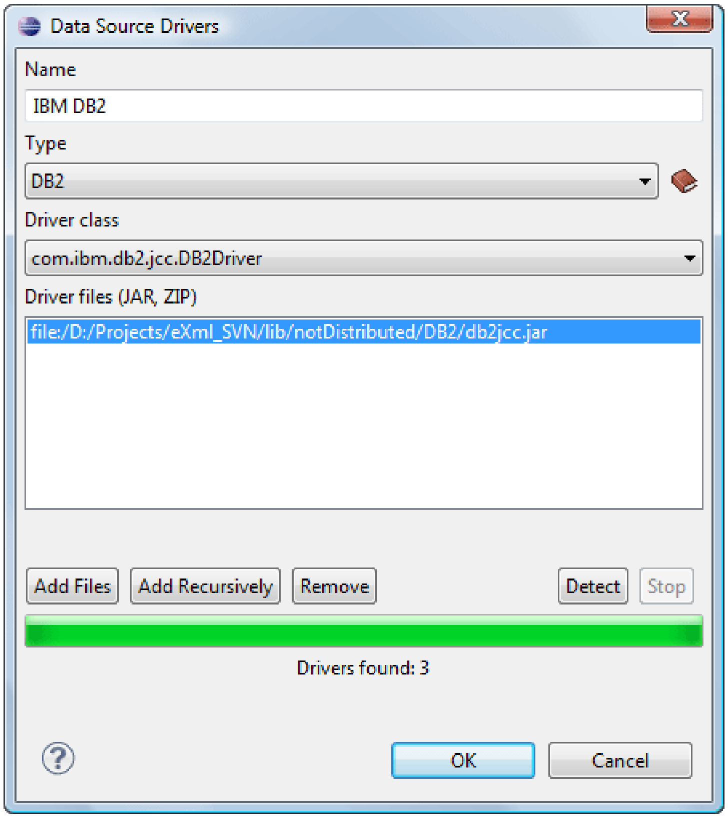
This section describes the procedures for configuring the data sources for relational databases.

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.



Data Source Drivers

Name

IBM DB2

Type

DB2

Driver class

com.ibm.db2.jcc.DB2Driver

Driver files (JAR, ZIP)

file:/D:/Projects/eXml_SVN/lib/notDistributed/DB2/db2jcc.jar

Drivers found: 3

Figure 239: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **DB2** in the driver type combo box.
5. Add the driver files for IBM DB2 using the **Add** button.

   The IBM DB2 driver files are:

   - db2jcc.jar

- db2jcc_license_cisuz.jar
- db2jcc_license_cu.jar

In the *Download links for database drivers* section there are listed the URLs from where to download the drivers necessary for accessing IBM DB2 databases in Oxygen XML Editor plugin .

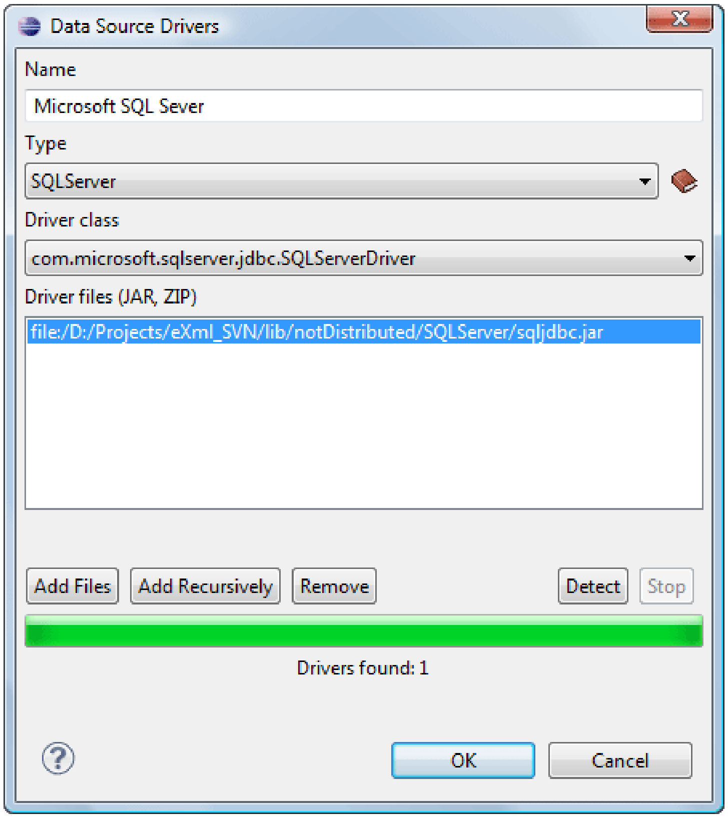
6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

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.



Data Source Drivers

Name: Microsoft SQL Sever

Type: SQLServer

Driver class: com.microsoft.sqlserver.jdbc.SQLServerDriver

Driver files (JAR, ZIP): file:/D:/Projects/eXml_SVN/lib/notDistributed/SQLServer/sqljdbc.jar

Drivers found: 1

Figure 240: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **SQLServer** in the driver type combo box.
5. Add the Microsoft SQL Server driver file using the **Add** button.

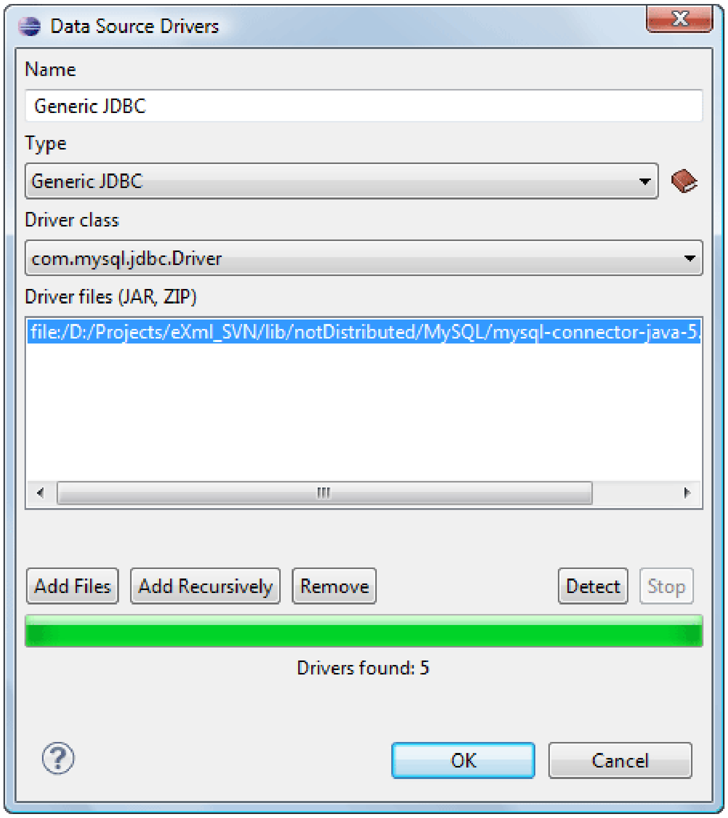
   The SQL Server driver file is called `sqljdbc.jar`. In the *Download links for database drivers* section there are listed the URLs from where to download the drivers necessary for accessing Microsoft SQL Server databases in Oxygen XML Editor plugin .
6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a Generic JDBC Data Source

Oxygen XML Editor plugin 's default configuration already contains a generic JDBC data source called **JDBC-ODBC Bridge**. Oxygen XML Editor plugin can display and edit XML data stored in PostgreSQL and Microsoft SQL Server databases accessible through a JDBC 4 driver. To do this, configure a **Generic JDBC** data source that uses a JDBC 4 driver. The following procedure shows you how to configure a generic JDBC data source:

1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

   The following dialog is displayed:



Name: Generic JDBC

Type: Generic JDBC

Driver class: com.mysql.jdbc.Driver

Driver files (JAR, ZIP): file:/D:/Projects/eXml_SVN/lib/notDistributed/MySQL/mysql-connector-java-5.

Drivers found: 5

Figure 241: Data Source Drivers Configuration Dialog

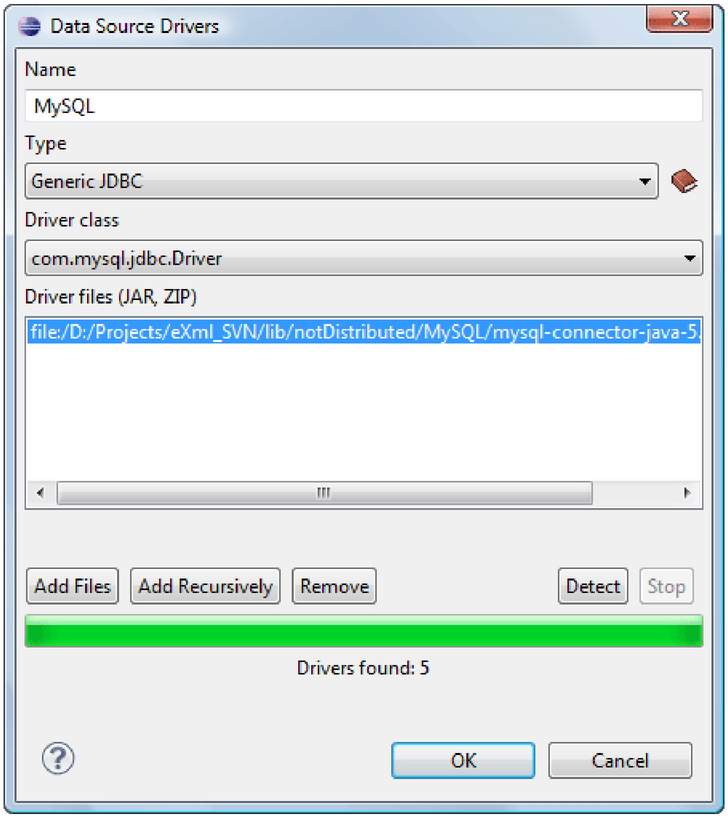
3. Enter a unique name for the data source.
4. Select **Generic JDBC** in the driver type combo box.
5. Add the driver file(s) using the **Add** button.
6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a MySQL Data Source

Previous versions of Oxygen XML Editor 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 Editor 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.



Data Source Drivers

Name

MySQL

Type

Generic JDBC

Driver class

com.mysql.jdbc.Driver

Driver files (JAR, ZIP)

file:/D:/Projects/eXml_SVN/lib/notDistributed/MySQL/mysql-connector-java-5.

Drivers found: 5

Figure 242: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **Generic JDBC** in the driver type combo box.
5. Add the MySQL 5 driver files using the **Add** button.

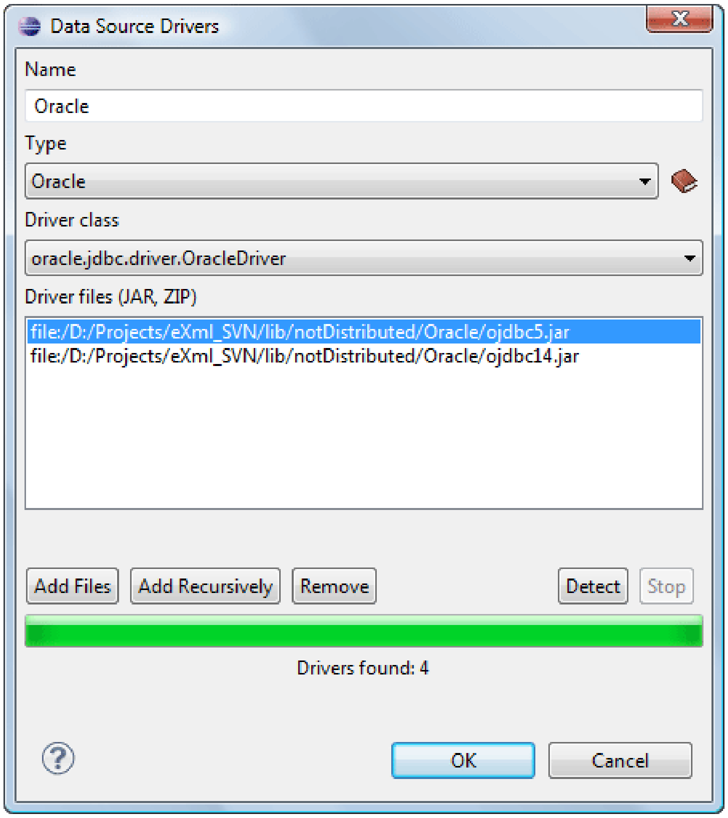
   The driver file for the MySQL server is called `mysql-com.jar`. In the *Download links for database drivers* section there are listed the URLs from where to download the drivers necessary for accessing MySQL databases in Oxygen XML Editor plugin .
6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure an Oracle 11g Data Source

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.



Name

Oracle

Type

Oracle

Driver class

oracle.jdbc.driver.OracleDriver

Driver files (JAR, ZIP)

```
file:/D:/Projects/eXml_SVN/lib/notDistributed/Oracle/ojdbc5.jar
file:/D:/Projects/eXml_SVN/lib/notDistributed/Oracle/ojdbc14.jar
```

Drivers found: 4

Figure 243: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.
4. Select **Oracle** in the driver type combo box.
5. Add the Oracle driver file using the **Add** button.

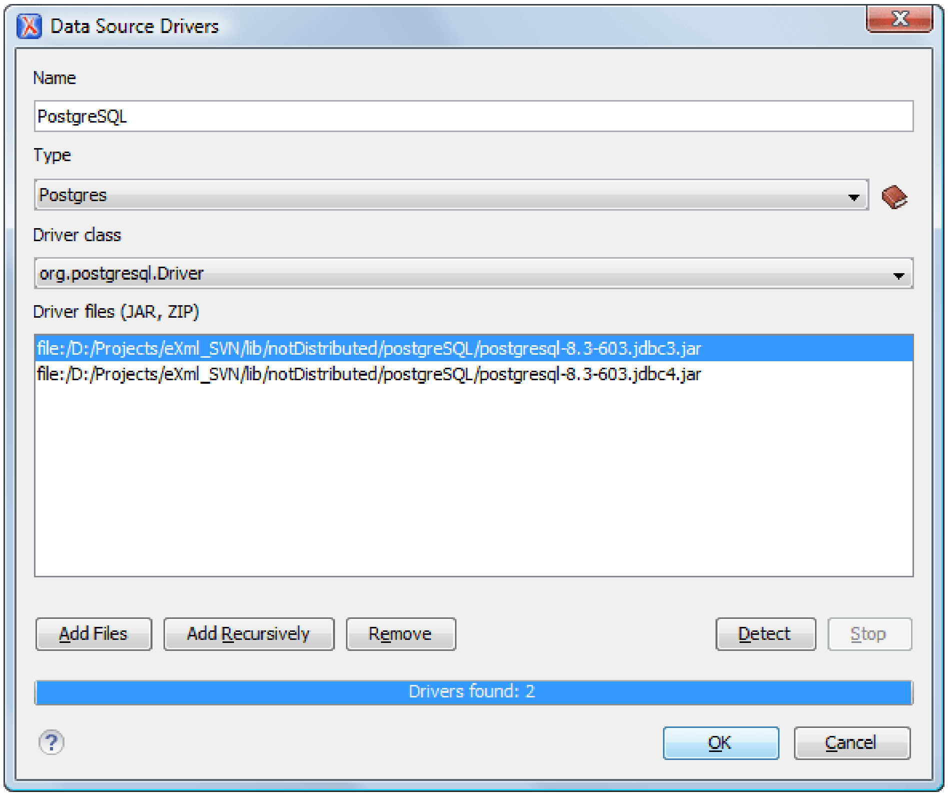
   The Oracle driver file is called `ojdbc5.jar`. In the *Download links for database drivers* section there are listed the URLs from where to download the drivers necessary for accessing Oracle databases in Oxygen XML Editor plugin .
6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a PostgreSQL 8.3 Data Source

The steps for configuring a data source for connecting to a PostgreSQL server are the following:

1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.



Name

PostgreSQL

Type

Postgres

Driver class

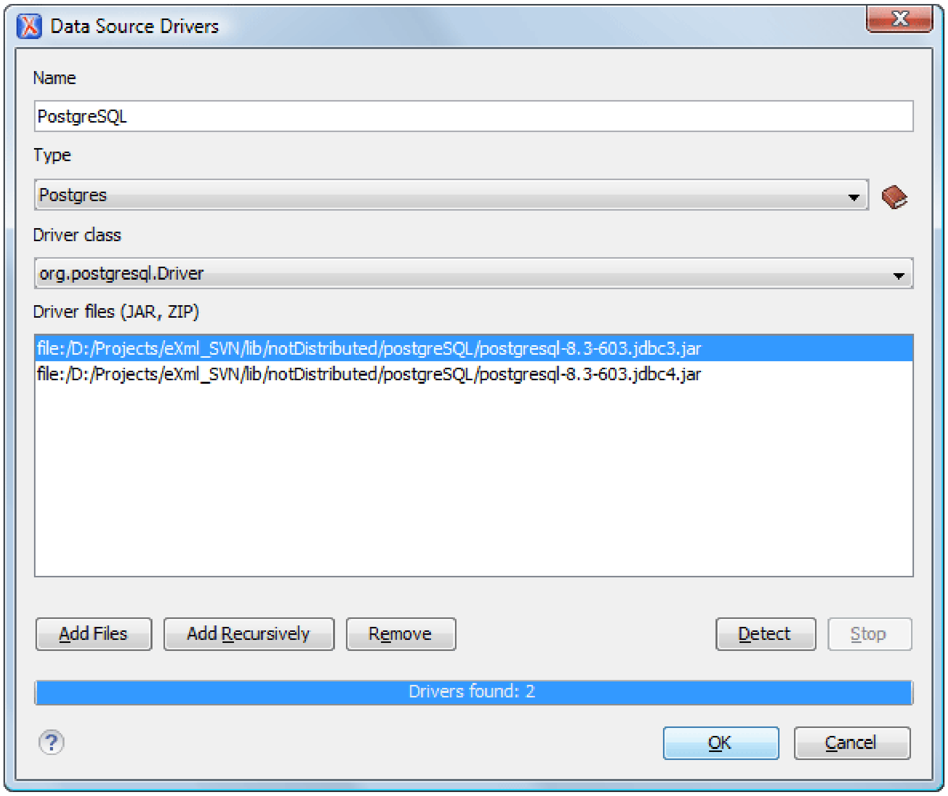
org.postgresql.Driver

Driver files (JAR, ZIP)

```
file:/D:/Projects/eXml_SVN/lib/notDistributed/postgreSQL/postgresql-8.3-603.jdbc3.jar
file:/D:/Projects/eXml_SVN/lib/notDistributed/postgreSQL/postgresql-8.3-603.jdbc4.jar
```

Drivers found: 2

Figure 244: Data Source Drivers Configuration Dialog



Name

PostgreSQL

Type

Postgres

Driver class

org.postgresql.Driver

Driver files (JAR, ZIP)

file:/D:/Projects/eXml_SVN/lib/notDistributed/postgreSQL/postgresql-8.3-603.jdbc3.jar
file:/D:/Projects/eXml_SVN/lib/notDistributed/postgreSQL/postgresql-8.3-603.jdbc4.jar

Drivers found: 2

Figure 245: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.

4. Select **PostgreSQL** in the driver type combo box.
5. Add the PostgreSQL driver file using the **Add** button.

   The PostgreSQL driver file is called `postgresql-8.3-603.jdbc3.jar`. In the *Download links for database drivers* section there are listed the URLs from where to download the drivers necessary for accessing PostgreSQL databases in Oxygen XML Editor plugin .

6. Select the most suited **Driver class**.
7. Click the **OK** button to finish the data source configuration.

Configuring Database Connections

This section describes the procedures for configuring the connections for relational databases:

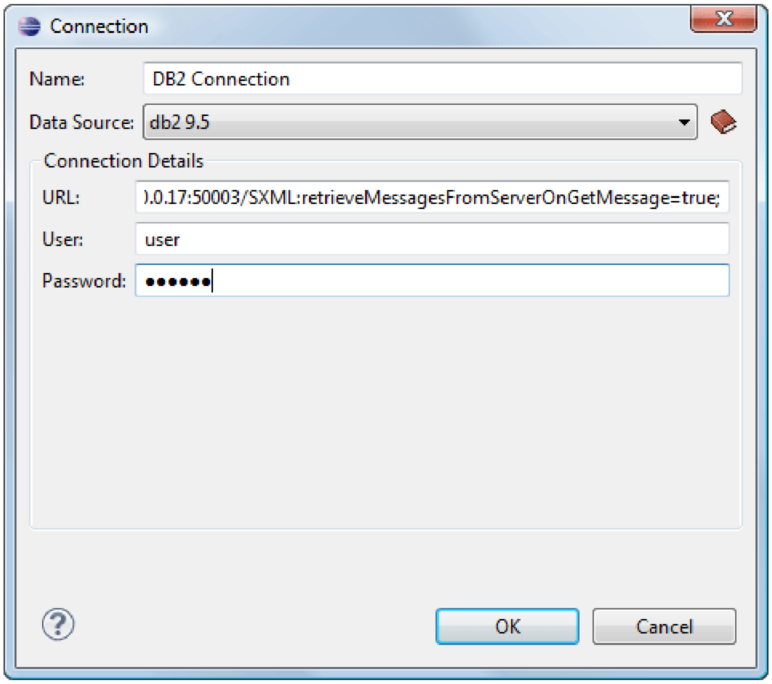
How to Configure an IBM DB2 Connection

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.



Name: DB2 Connection
Data Source: db2 9.5
URL:).0.17:50003/SXML:retrieveMessagesFromServerOnGetMessage=true;
User: user
Password: ••••••

Figure 246: The Connection Configuration Dialog

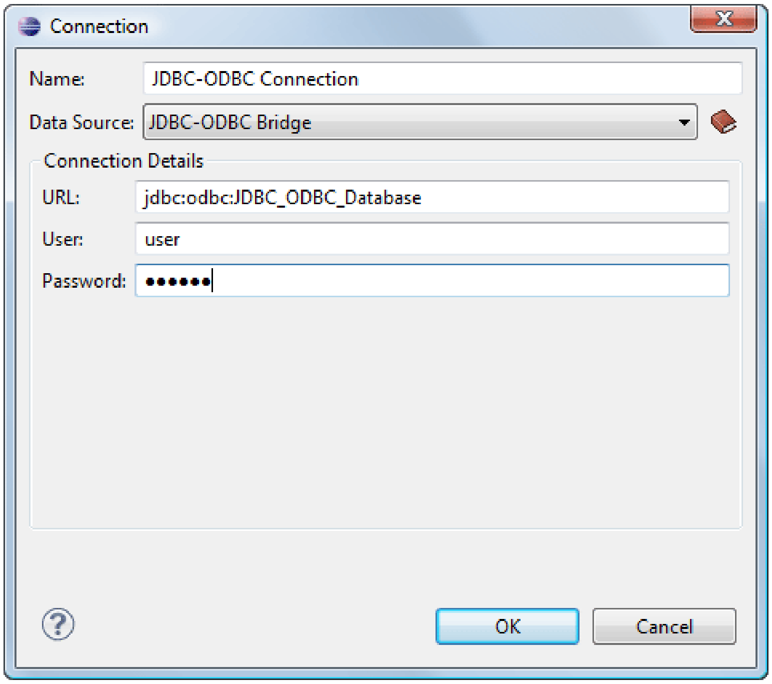
3. Enter a unique name for the connection.
4. Select an IBM DB2 data sources in the **Data Source** combo box.
5. Fill-in the connection details.
   a) Fill-in the URL to the installed IBM DB2 engine.
   b) Fill-in the user name to access the IBM DB2 engine.
   c) Fill-in the password to access the IBM 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.



Name: JDBC-ODBC Connection

Data Source: JDBC-ODBC Bridge

Connection Details

URL: jdbc:odbc:JDBC_ODBC_Database

User: user

Password: ••••••

Figure 247: 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.

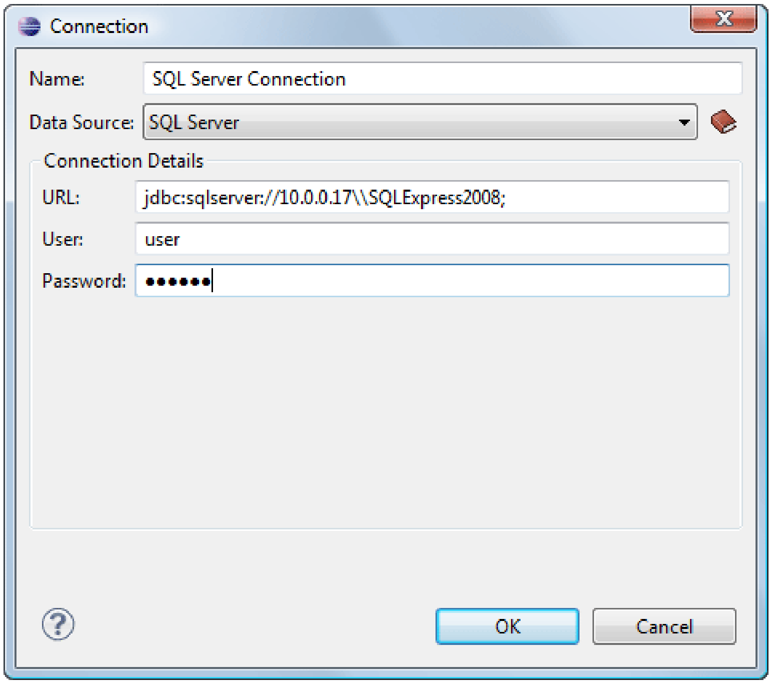
How to Configure a Microsoft SQL Server Connection

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.



Connection

Name: SQL Server Connection

Data Source: SQL Server

Connection Details

URL: jdbc:sqlserver://10.0.0.17\\SQLExpress2008;

User: user

Password: ●●●●●●

Figure 248: 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 and 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.

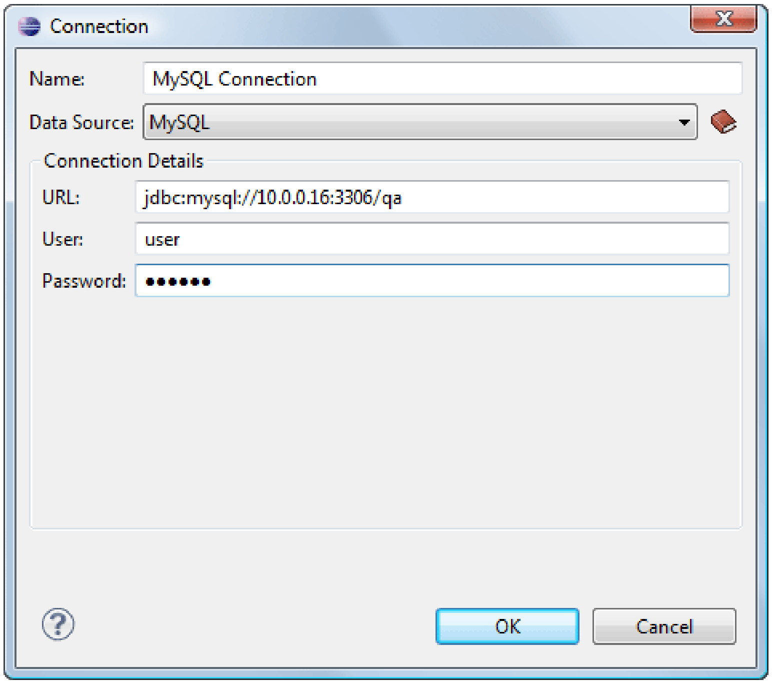
   c) Fill-in the password for the connection to the SQL Server.
6. Click the **OK** button to finish the configuration of the database connection.

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.



Name: MySQL Connection

Data Source: MySQL

Connection Details

URL: jdbc:mysql://10.0.0.16:3306/qa

User: user

Password: ••••••

Figure 249: 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.

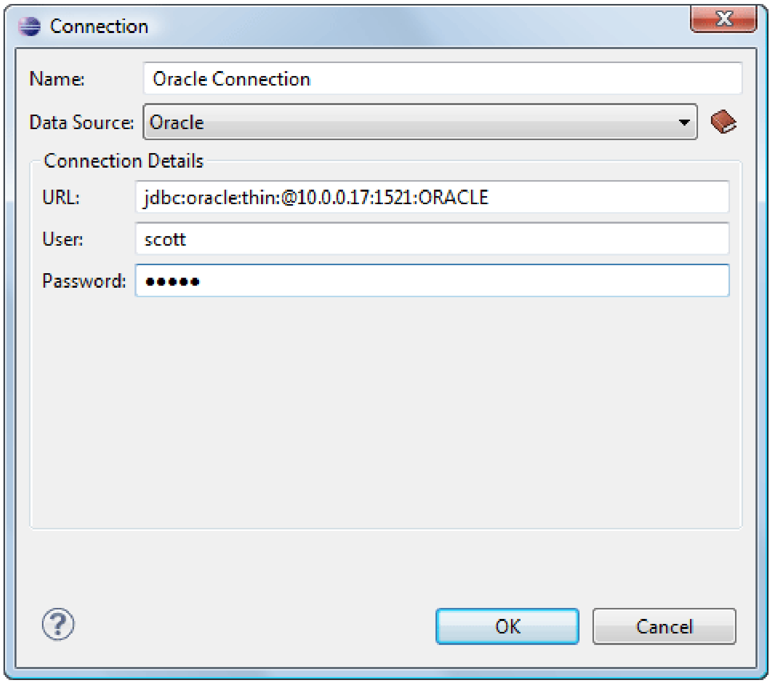
How to Configure an Oracle 11g Connection

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.



Name: Oracle Connection
Data Source: Oracle
URL: jdbc:oracle:thin:@10.0.0.17:1521:ORACLE
User: scott
Password: •••••

Figure 250: 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.

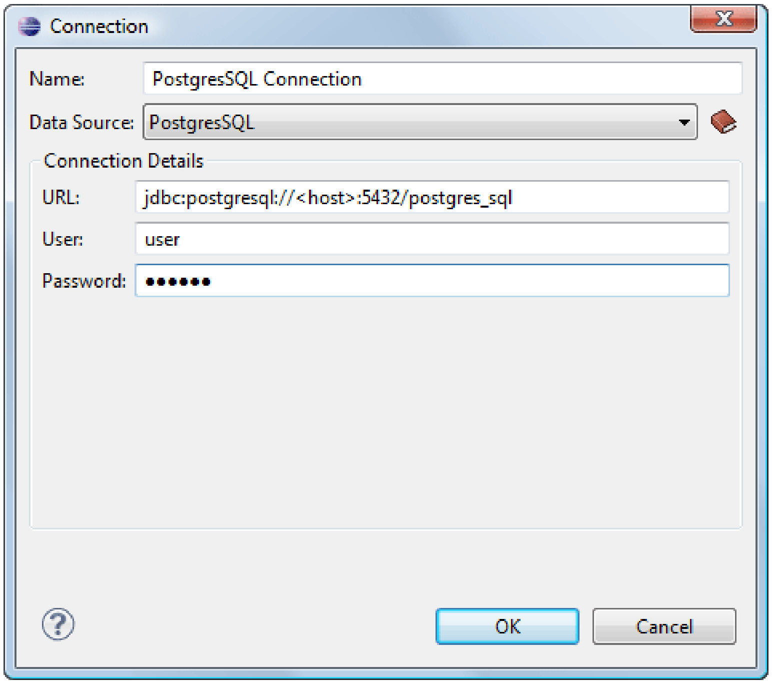
How to Configure a PostgreSQL 8.3 Connection

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.



Connection

Name: PostgresSQL Connection

Data Source: PostgresSQL

Connection Details

URL: jdbc:postgresql://<host>:5432/postgres_sql

User: user

Password: ••••••

Figure 251: The Connection Configuration Dialog

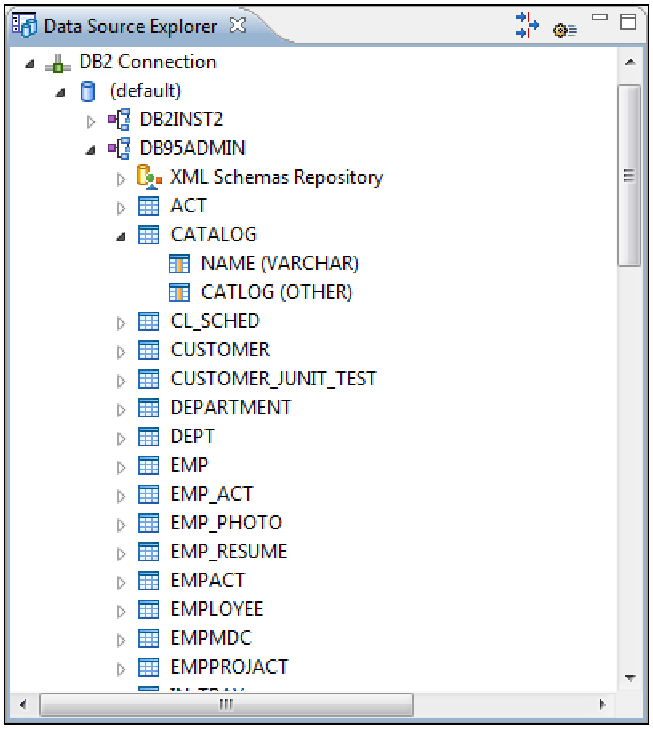
3. Enter a unique name for the connection.
4. Select a PostgreSQL 8.3 data source in the **Data Source** combo box.
5. Fill-in the connection details.
   a) Fill-in the URL of the PostgreSQL 8.3 server.
   b) Fill-in the user name for the connection to the PostgreSQL 8.3 server.
   c) Fill-in the password 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 Editor plugin supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.



- DB2 Connection
  - (default)
    - DB2INST2
    - DB95ADMIN
      - XML Schemas Repository
      - ACT
      - CATALOG
        - NAME (VARCHAR)
        - CATLOG (OTHER)
      - CL_SCHED
      - CUSTOMER
      - CUSTOMER_JUNIT_TEST
      - DEPARTMENT
      - DEPT
      - EMP
      - EMP_ACT
      - EMP_PHOTO
      - EMP_RESUME
      - EMPACT
      - EMPLOYEE
      - EMPMDC
      - EMPPROJACT

Figure 252: 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 Editor plugin .

Microsoft SQL Server's XML Schema Repository Level

The contextual menu of a **XML Schema Repository** node of the tree from the **Data Source Explorer** view contains the following actions:

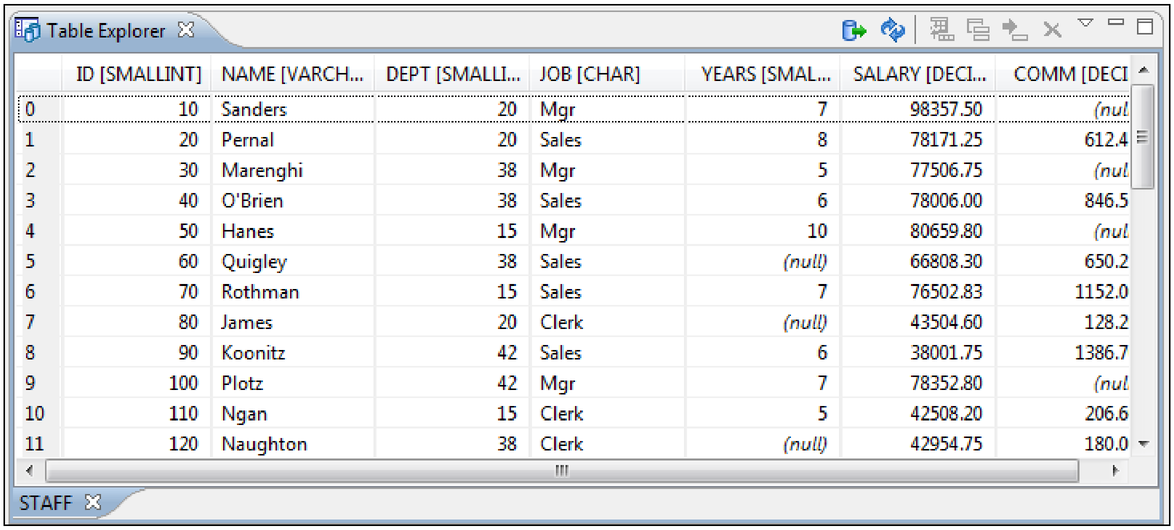
- **Refresh** - Performs a refresh of the selected node's subtree.
- **Register** - Opens a dialog for adding a new schema file in the DB XML repository. In this dialog you enter a collection name and the necessary schema files. XML Schema files management is done by using the **Add** and **Remove** buttons.

The actions available at **Schema** level are the following:

- **Refresh** - Performs a refresh of the selected node (and it's subtree).
- **Add** - Adds a new schema to the XML Schema files.
- **Unregister** - Removes the selected schema from the XML Schema Repository.
- **View** - Opens the selected schema in Oxygen XML Editor plugin .

Table Explorer View

Every table from the **Data Source Explorer** view can be displayed and edited in the **Table Explorer** view by pressing the **Edit** button from the contextual menu or by double-clicking one of its fields. To modify a cell's content, double click it and start typing. When editing is finished, Oxygen XML Editor plugin will try to update the database with the new cell content.



	ID [SMALLINT]	NAME [VARCH...	DEPT [SMALLI...	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 253: The Table Explorer View

You can sort the content of a table by one of its columns by clicking on its column header.

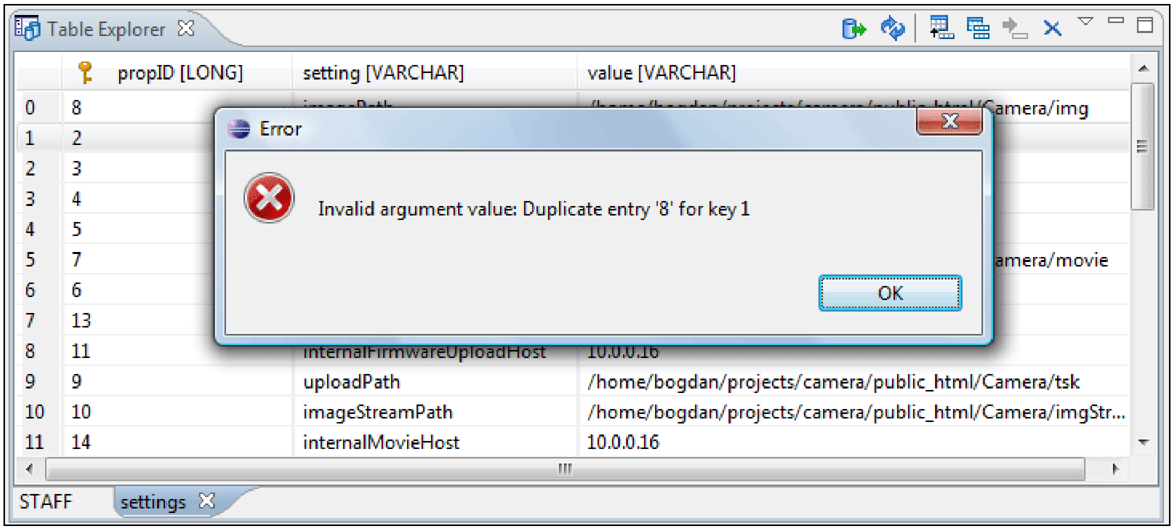
Note the following:

- The first column is an index (does not belong to the table structure).
- Every column header contains the field name and its data type.
- The primary key columns are marked with this symbol: 🔑 .
- Multiple tables are presented in a tabbed manner

For performance issues, you can set the maximum number of cells that will be displayed in the **Table Explorer** view (the **Limit the number of cells** field from the *Data Sources* Preferences page). If a table having more cells than the value set in Oxygen XML Editor plugin 's options is displayed in the **Table Explorer** view, a warning dialog will inform you that the table is only partially shown.

You will be notified if the value you have entered in a cell is not valid (and thus it cannot be updated).

- If the content of the edited cell does not belong to the data type of the column, an Information dialog will appear, notifying you that the value you have inserted cannot be converted to the SQL type of that field. For example, in the above figure `propID` contains `LONG` values. If a character or string was inserted, you would get the error message that a String value cannot be converted to the requested SQL type (NUMBER).
- If the constraints of the database are not met (like primary key constraints for example), an Information dialog will appear, notifying you of the reason the database has not been updated. For example, if you'd try to set the primary key `propID` for the second record in the table to 10 also, you would get the following message:



	propID [LONG]	setting [VARCHAR]	value [VARCHAR]
0	8	imagePath	/home/bogdan/projects/camera/public_html/Camera/img
1	2		
2	3		
3	4		
4	5		
5	7		amera/movie
6	6		
7	13		
8	11	internalFirmwareUploadHost	10.0.0.16
9	9	uploadPath	/home/bogdan/projects/camera/public_html/Camera/tsk
10	10	imageStreamPath	/home/bogdan/projects/camera/public_html/Camera/imgStr...
11	14	internalMovieHost	10.0.0.16

Invalid argument value: Duplicate entry '8' for key 1

Figure 254: Duplicate entry for primary key

The usual edit actions (**Cut**, **Copy**, **Paste**, **Select All**, **Undo**, **Redo**) are available in the popup menu of the edited cell.

The contextual menu available on every cell has the following actions:

- Set NULL - Sets the content of the cell to (null). This action is disabled for columns that cannot be null.
- **Insert row** - Inserts an empty row in the table.
- **Duplicate row** - Makes a copy of the selected row and adds it in the **Table Explorer** view. You should note that the new row will not be inserted in the database table until all conflicts are solved.
- **Commit row** - Commits the selected row.
- ✖ **Delete row** - Deletes the selected row.
- **Copy** - Copies the content of the cell.
- **Paste** - Performs paste in the selected cell.

Some of the above actions are also available on the **Table Explorer** toolbar:

- **Export to XML** - Opens the **Export Criteria** dialog (a thorough description of this dialog can be found in the *Import from database* chapter) .
- **Refresh** - Performs a refresh of the selected node's subtree.
- **Insert row** - Inserts an empty row in the table.
- **Duplicate row** - Makes a copy of the selected row and adds it in the **Table Explorer** view. You should note that the new row will not be inserted in the database table until all conflicts are solved.
- **Commit row** - Commits the selected row.
- ✖ **Delete row** - Deletes the selected row.

SQL Execution Support

Oxygen XML Editor plugin 's support for writing SQL statements includes syntax highlight, folding and drag&drop (DND) from the **Data Source Explorer** view. It also includes transformation scenarios for executing the statements and the results are displayed in the **Table Explorer** view.

Drag and Drop from Data Source Explorer View

Drag and drop(*DND*) from the **Data Source Explorer** view to the SQL editor allows creating SQL statements quickly by inserting the names of tables and columns in the SQL statements.

1. Configure a database connection (see the procedure specific for your database server).
2. Browse to the table you will use in your statement.

3. Drag the table or a column of the table into the editor where a SQL file is open.

   DND is available both on the table and on its fields. A popup menu is displayed in the SQL editor.

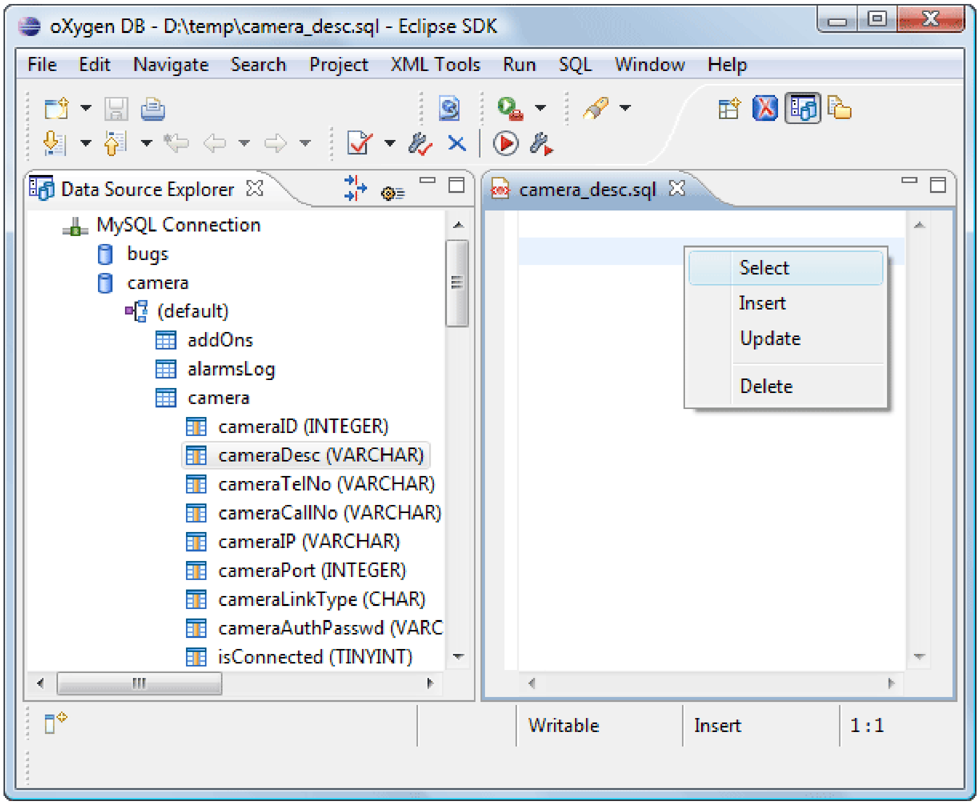


Figure 255: SQL statement editing with DND

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 Editor plugin window.
4. View more complex return values of the SQL transformation in a separate editor panel.

   A more complex value returned by the SQL query (for example an XMLTYPE value or a CLOB one) cannot be displayed entirely in the result table.

   a) Right click on the cell containing the complex value.
   b) Select the action **Copy cell** from the popup menu.
      The action will copy the value in the clipboard.
   c) Paste the value where you need it.
      For example you can paste the value in an opened XQuery editor panel of Oxygen XML Editor plugin .

Native XML Database (NXD) Support

Native XML databases have an XML-based internal model and their fundamental unit of storage is XML. Oxygen XML Editor plugin offers support for the following native XML databases:

- Berkeley DB XML
- eXist
- MarkLogic
- 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 Editor plugin can be found on our *website*.

Oxygen XML Editor plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The following directory definitions shall apply:

- OXY_DIR - Oxygen XML Editor 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 Editor plugin can be found on our *website*.

Oxygen XML Editor plugin supports eXist database server versions 1.3, 1.4 and 1.5.

1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *eXist* from the **Driver type** combo box.
5. Press the **Add** button to add the eXist driver files.

   The following driver files should be added in the dialog box for setting up the eXist datasource. They are found in the installation directory of the eXist database server. Please make sure you copy the files from the installation of the eXist server where you want to connect from Oxygen.

   - `exist.jar`
   - `lib/core/xmldb.jar`
   - `lib/core/xmlrpc-client-3.1.1.jar`
   - `lib/core/xmlrpc-common-3.1.1.jar`
   - `lib/core/ws-commons-util-1.0.2.jar`

   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 Editor plugin can be found on our *website*.

1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *MarkLogic* from the **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 Editor 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 Editor plugin can be found on our *website* .

1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *Tamino* from the **Driver type** combo box.
5. Press the **Add** button to add the Tamino driver files.

   The driver files for the Tamino database are the following:

   - `TaminoAPI4J.jar`
   - `TaminoAPI4J-l10n.jar`
   - `TaminoJCA.jar`

   Note: You must use the jar files from the version 4.4.1 of the Tamino database.

6. Click the **OK** button to finish the data source configuration.

How to Configure a Raining Data TigerLogic Data Source

Note: Support for TigerLogic database will be discontinued starting with version 14.

The latest instructions on how to configure TigerLogic support in Oxygen XML Editor plugin can be found on our *website* .

1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.

4. Select *TigerLogic* from the **Driver type** combo box.
5. Press the **Add** button to add the TigerLogic driver files.

   The driver files for the TigerLogic database are found in the TigerLogic JDK lib directory from the server side:

   - `connector.jar`
   - `jca-connector.jar`
   - `tlapi.jar`
   - `tlerror.jar`
   - `utility.jar`
   - `xmlparser.jar`
   - `xmltypes.jar`
6. Click the **OK** button to finish the data source configuration.

How to Configure a Documentum xDb (X-Hive/DB) 10 Data Source

The latest instructions on how to configure support for Documentum xDb (X-Hive/DB) 10 versions 8 and 9 in Oxygen XML Editor plugin can be found on our *website*.

1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *XHive* from the **Driver type** combo box.
5. Press the **Add** button to add the XHive driver files.

   The driver files for the Documentum xDb (X-Hive/DB) 10 database are found in the Documentum xDb (X-Hive/DB) 10 lib directory from the server installation folder:

   - `antlr-runtime.jar`
   - `aspectjrt.jar`
   - `icu4j.jar`
   - `xhive.jar`
   - `google-collect.jar`
6. Click the **OK** button to finish the data source configuration.

Configuring Database Connections

This section describes the procedures for configuring the connections for native databases.

How to Configure a Berkeley DB XML Connection

Oxygen XML Editor plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The steps for configuring a connection to a Berkeley DB XML database are the following:

1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
   a) Set the path to the Berkeley DB XML 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 Editor plugin uses XCC connector to interact with MarkLogic XDBC server and requires the basic authentication schema to be set. Starting with version MarkLogic 4.0 the default authentication method when you create a HTTP or WebDAV Server is digest, so make sure to change it to basic.

    b) Set the port number of the MarkLogic engine 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 Editor plugin . The following procedure explains the *xhive://* protocol connection type.

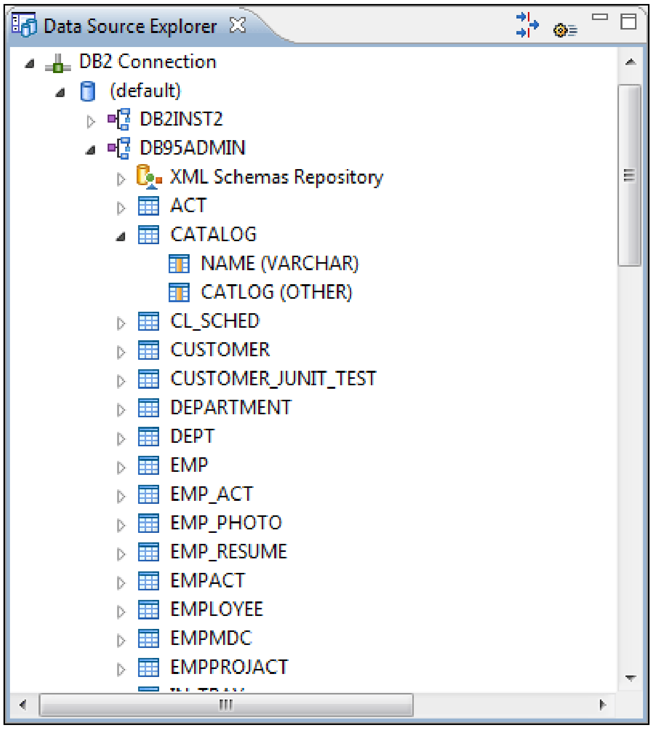
1. Go to menu **Preferences** > **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
   a) Set the URL property of the connection in the **URL** field.

      If the property is a URL of the form *xhive://host:port*, the Documentum xDb (X-Hive/DB) 10 connection will attempt to connect to a Documentum xDb (X-Hive/DB) 10 server running behind the specified TCP/IP port.

   b) Set the user name to access the Documentum xDb (X-Hive/DB) 10 engine in the **User** field.
   c) Set the password to access the Documentum xDb (X-Hive/DB) 10 engine in the **Password** field.
   d) Set the name of the database to access from the Documentum xDb (X-Hive/DB) 10 engine in the **Database** field.
   e) Check the checkbox **Run XQuery in read / write session (with committing)** if you want to end the session with a commit, otherwise the session ends with a rollback.
6. Click the **OK** button to finish the connection configuration.

Data Source Explorer View

This view presents in a tree-like fashion the database connections configured 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 Editor plugin supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.



Data Source Explorer

- DB2 Connection
  - (default)
    - DB2INST2
    - DB95ADMIN
      - XML Schemas Repository
      - ACT
      - CATALOG
        - NAME (VARCHAR)
        - CATLOG (OTHER)
      - CL_SCHED
      - CUSTOMER
      - CUSTOMER_JUNIT_TEST
      - DEPARTMENT
      - DEPT
      - EMP
      - EMP_ACT
      - EMP_PHOTO
      - EMP_RESUME
      - EMPACT
      - EMPLOYEE
      - EMPMDC
      - EMPPROJACT

Figure 256: 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 Editor plugin allows the user to browse the native Oracle XML Repository and perform various operations on the resources in the repository.

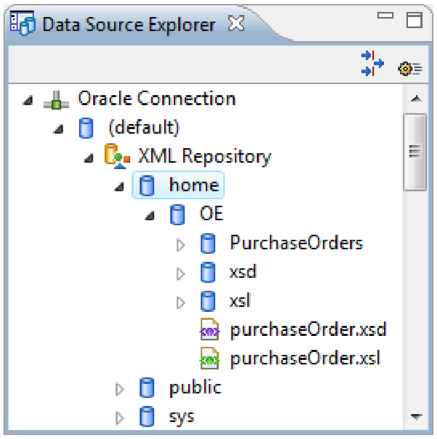


Figure 257: Browsing the Oracle XML DB Repository

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.

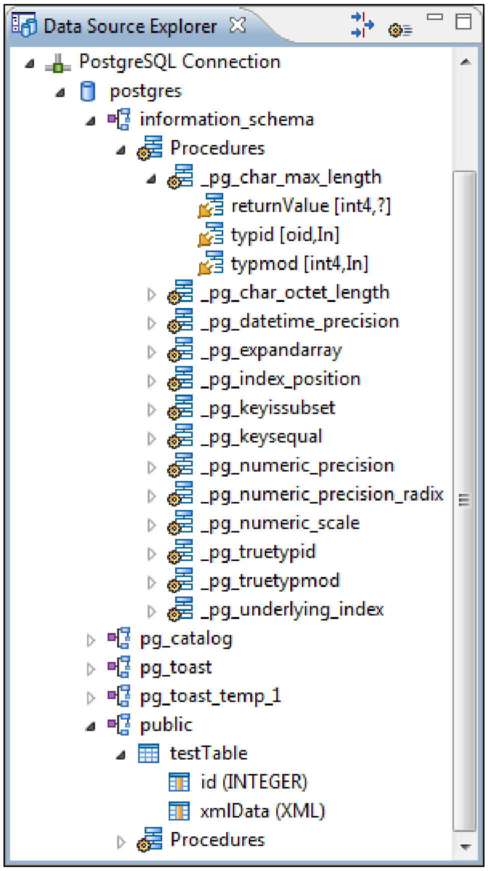
The actions available at resource level are the following:

- **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 Editor plugin allows the user to browse the structure of the PostgreSQL database in the **Data Source Explorer** view and open the tables in the **Table Explorer** view.



Data Source Explorer

- PostgreSQL Connection
  - postgres
    - information_schema
      - Procedures
        - _pg_char_max_length
          - returnValue [int4,?]
          - typid [oid,In]
          - typmod [int4,In]
        - _pg_char_octet_length
        - _pg_datetime_precision
        - _pg_expandarray
        - _pg_index_position
        - _pg_keyissubset
        - _pg_keysequal
        - _pg_numeric_precision
        - _pg_numeric_precision_radix
        - _pg_numeric_scale
        - _pg_truetypid
        - _pg_truetypmod
        - _pg_underlying_index
    - pg_catalog
    - pg_toast
    - pg_toast_temp_1
    - public
      - testTable
        - id (INTEGER)
        - xmlData (XML)
      - Procedures

Figure 258: 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.

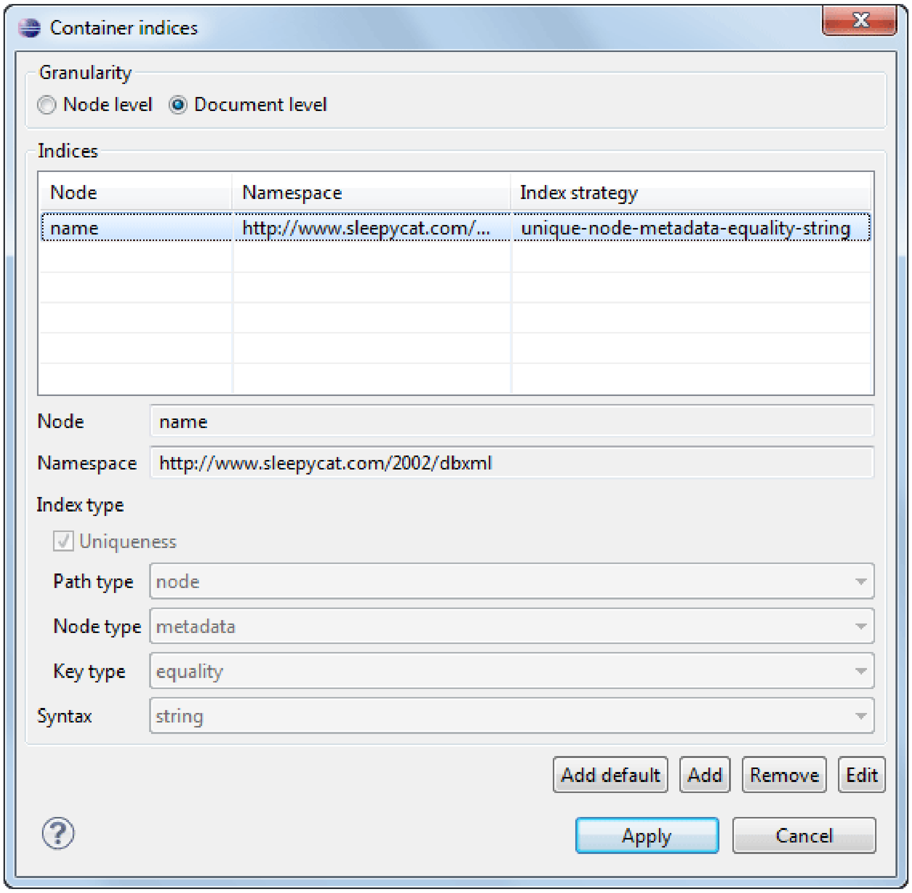
  - **Allow validation** - If checked it causes documents to be validated when they are loaded into the container. The default behavior is to not validate documents.
  - **Index nodes** - If checked it causes indices for the container to return nodes rather than documents. The default is to index at the 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.



Container indices

Granularity

○ Node level ● Document level

Indices

Node	Namespace	Index strategy
name	http://www.sleepycat.com/...	unique-node-metadata-equality-string

Node: name

Namespace: http://www.sleepycat.com/2002/dbxml

Index type

☑ Uniqueness

Path type: node

Node type: metadata

Key type: equality

Syntax: string

Figure 259: 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 ca be done through WebDAV. For this a WebDAV URL must be configured in *the MarkLogic connection*. The actions that can be performed on MarkLogic resources through WebDAV are the same used for a WebDAV connection (see more about this in *WebDAV Connection* section).

Software AG Tamino Connection

This section explains the actions that are available on a Tamino connection.

Actions Available at Connection Level

For a Tamino database the actions available at connection level in the **Data Source Explorer** view are the following:

- **Refresh** - Performs a refresh of the selected node's subtree.
- **Disconnect** - Closes the current database connection.
- **Configure Database Sources** - Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.
- **Add container** - Allows you to create a new collection in the database.

Actions Available at Collection Level

For every new Tamino collection created in the **Data Source Explorer** view, you can specify if a schema is *required*, *optional* or *prohibited*. The following actions are available at collection level:

- **Refresh** - Performs a refresh of the selected node's subtree.
- **Filter ...** - An XQuery expression can be specified for filtering the nodes displayed in the selected Tamino container. It is only possible to specify one predicate. In the XQuery syntax a predicate is enclosed in square brackets. The square brackets, however, must not be specified in the dialog box displayed by this action. Only the predicate must be specified and it will be applied on the selected document type. For example: `name/surname between 'B', 'C'`
- **Insert XML instance** - Allows you to load a new XML document.
- **Insert non XML instance** - Allows you to load a non XML document.
- **Modify Collection Properties** - Allows you to change the schema usage for the selected collection to optional. This action is available on collections with required and prohibited schema usage.
- **Define schema** - Allows you to add a new schema in the Schema Repository. This action is available on collections with optional and required schema usage.
- **Delete** - Removes the selected collection. If it is a Tamino doctype then the action removes all the XML instances contained in the document type.
- **Set default** - Sets this collection as the default collection for running queries with the `input()` function.

Actions Available at Schema Level

For a Tamino database the actions available at schema level in the **Data Source Explorer** view are the following:

- **Refresh** - Performs a refresh of the selected schema.
- **Open** - Opens the selected schema in the editor. There are supported schema changes that preserve the validity relative to the existent instances.
- **Delete** - Removes the selected schema from the Schema Repository.

Actions Available at Resource Level

For a Tamino database the actions available at resource level in the **Data Source Explorer** view are the following:

- **Refresh** - Performs a refresh of the selected resource.
- **Open** - Opens the selected resource in the editor.
- **Rename** - Allows you to change the name of the selected resource.
- **Delete** - Removes the selected resource.
- **Copy location** - Allows you to copy to clipboard an application specific URL for the resource which can then be used for various actions like opening or transforming the resources.
- **Properties** - Allows the user to view various useful properties associated with the resource.
- **Save As** - Allows you to save the name of the selected binary resource as a file on disk.

Validation of an XML resource stored in a Tamino database is done against the schema associated with the resource in the database.

Documentum xDb (X-Hive/DB) Connection

This section explains the actions that are available on a Documentum xDb (X-Hive/DB) 10 connection.

Actions Available at Connection Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at connection level in the **Data Source Explorer** view are the following:

- **Refresh** - Performs a refresh of the selected node's subtree.
- **Disconnect** - Closes the current database connection.
- **Configure Database Sources** - Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.
- **Add library** - Allows you to add a new library.
- **Insert XML Instance** - Allows you to add a new XML resource directly into the database root. See *Documentum xDb (X-Hive/DB) 10 Parser Configuration* for more details.
- **Insert non XML Instance** - Allows you to add a new non XML resource directly into the database root.
- **Properties** - Displays the connection properties.

Actions Available at Catalog Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at catalog level in the **Data Source Explorer** view are the following:

- **Refresh** - Performs a refresh of the selected catalog.
- **Add AS models** - Allows you to add a new abstract schema model to the selected catalog.
- **Set default schema** - Allows you to set a default DTD to be used for parsing. It is not possible to set a default XML Schema.
- **Clear default schema** - Allows you to clear the default DTD. The action is available only if there is a DTD set as default.
- **Properties** - Displays the catalog properties.

Actions Available at Schema Resource Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at schema resource level in the **Data Source Explorer** view are the following:

- **Refresh** - Performs a refresh of the selected schema resource.
- **Open** - Opens the selected schema resource in the editor.
- **Rename** - Allows you to change the name of the selected schema resource.
- **Save As** - Allows you to save the selected schema resource as a file on disk.
- **Delete** - Removes the selected schema resource from the catalog
- **Copy location** - Allows you to copy to clipboard the URL of the selected schema resource.
- **Set default schema** - Allows you to set the selected DTD to be used as default for parsing. The action is available only for DTD.
- **Clear default schema** - Allows you to unset the selected DTD. The action is available only if the selected DTD is the current default to be used for parsing.

Actions Available at Library Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at library level in the **Data Source Explorer** view are the following:

- **Refresh** - Performs a refresh of the selected library.
- **Add library** - Adds a new library as child of the selected library.
- **Add local catalog** - Adds a catalog to the selected library. By default, only the root-library has a catalog, and all models would be stored there.

- **Insert XML Instance** - Allows you to add a new XML resource to the selected library. See *Documentum xDb (X-Hive/DB) 10 Parser Configuration* for more details.
- **Insert non XML Instance** - Allows you to add a new non XML resource to the selected library.
- **Rename** - Allows you to specify a new name for the selected library.
- **Move** - Allows you to move the selected library to a different one (also available through drag and drop).
- ✖ **Delete** - Removes the selected library.
- **Properties** - Displays the library properties.

Actions Available at Resource Level

When an XML instance document is added For a Documentum xDb (X-Hive/DB) 10 database the actions available at resource level in the **Data Source Explorer** view are the following:

- **Refresh** - Performs a refresh of the selected resource.
- **Open** - Opens the selected resource in the editor.
- **Rename** - Allows you to change the name of the selected resource.
- **Move** - Allows you to move the selected resource in a different library in the database tree (also available through drag and drop).

  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-shrinked.jar

              core/google-collect.jar core/icu4j.jar core/lucene-regex.jar
core/lucene.jar
              core/serializer.jar core/xalan.jar core/xercesImpl.jar
```

Because the driver was configured to use `xhive.jar` directly from the *xDB* installation(where many other jars are located), `core/xercesImpl.jar` from the *xDB* installation directory is loaded even though it is not specified in the list of jars from the data source driver configuration(it is in the classpath from `xhive.jar`'s *MANIFEST.MF*).A simple workaround for this issue is to copy **ONLY** the jar files used in the driver configuration to a separate folder and configure the data source driver to use them from there.

XQuery and Databases

XQuery is a native XML query language which is useful for querying XML views of relational data to create XML results. It provides the mechanism to efficiently and easily extract information from Native XML Databases (NXD) and relational data as well. The following database systems supported in Oxygen XML Editor plugin offer XQuery support:

- *Native XML Databases:*
  - Berkeley DB XML
  - eXist
  - MarkLogic (validation support 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.

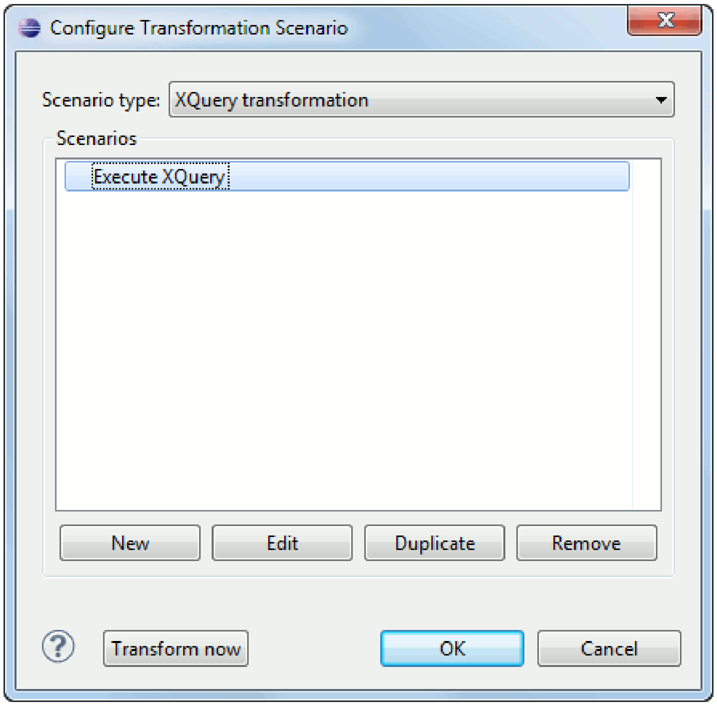
1. Configure a data source for the database.

   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**.

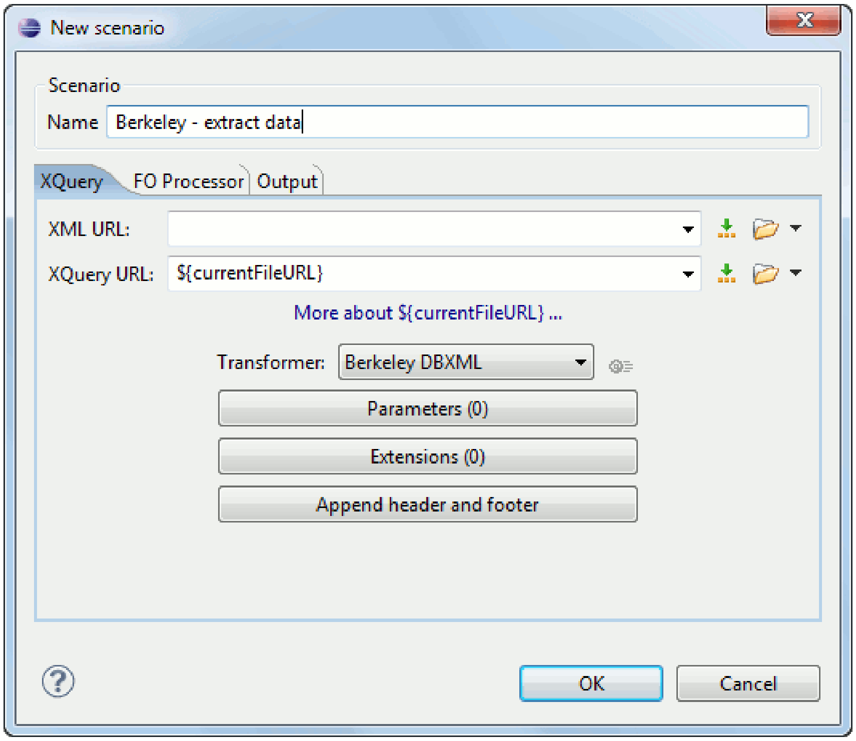
   The dialog for configuring a scenario will be opened.



Scenario type: XQuery transformation

   b) Click the **New** button of the dialog.

   The dialog for editing an XQuery scenario will be opened.



Name: Berkeley - extract data

XQuery URL: ${currentFileURL}

Transformer: Berkeley DBXML

Figure 260: Edit Scenario Dialog

c) Insert the scenario name in the dialog for editing the scenario.

d) Choose the database connection in the **Transformer** combo box.

e) Configure any other parameters if necessary.

For an XQuery transformation the output tab has an option called **Sequence** which allows you to execute an XQuery in lazy mode. The amount of data extracted from the database is controlled from option *Size limit on Sequence view*. If you choose **Perform FO Processing** in the **FO Processor** tab, the **Sequence** option is ignored.

f) Click the **OK** button to finish editing the scenario.

Once the scenario is associated with the XQuery file, the query can include calls to specific XQuery functions implemented by that engine. The available functions depend on the target database engine selected in the scenario. For example for eXist and Berkeley DB XML *the content completion assistant* lists the functions supported by that database engine. This is useful for inserting in the query only calls to the supported functions (standard XQuery functions or extension ones).

Note: An XQuery transformation is executed against a Berkeley DB XML server as a transaction using the query transaction support of the server.

3. Run the scenario.

To view a more complex value returned by the query that cannot be displayed entirely in the XQuery query result table at the bottom of the Oxygen XML Editor plugin window, for example an XMLTYPE value or a CLOB value, do the following actions:

- right click on that table cell
- select the 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 Editor plugin .

XQuery Database Debugging

This section describes the procedures for debugging XQuery transformations that are executed against MarkLogic databases and Berkeley DB XML ones.

Debugging with MarkLogic

To start a debug session against the MarkLogic engine 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 Editor 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 Editor plugin 's XQuery Debugger. *The same restrictions and peculiarities* apply for the Berkeley debugger as for the MarkLogic one.

WebDAV Connection

This section explains how to work with a WebDAV connection in the **Data Source Explorer** view.

How to Configure a WebDAV Connection

Oxygen XML Editor 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 23

Importing Data

Topics:

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 Editor plugin offers support for importing text files, MS Excel files, Database Data and HTML files into XML documents. The XML documents can be further converted into other formats using the Transform features.

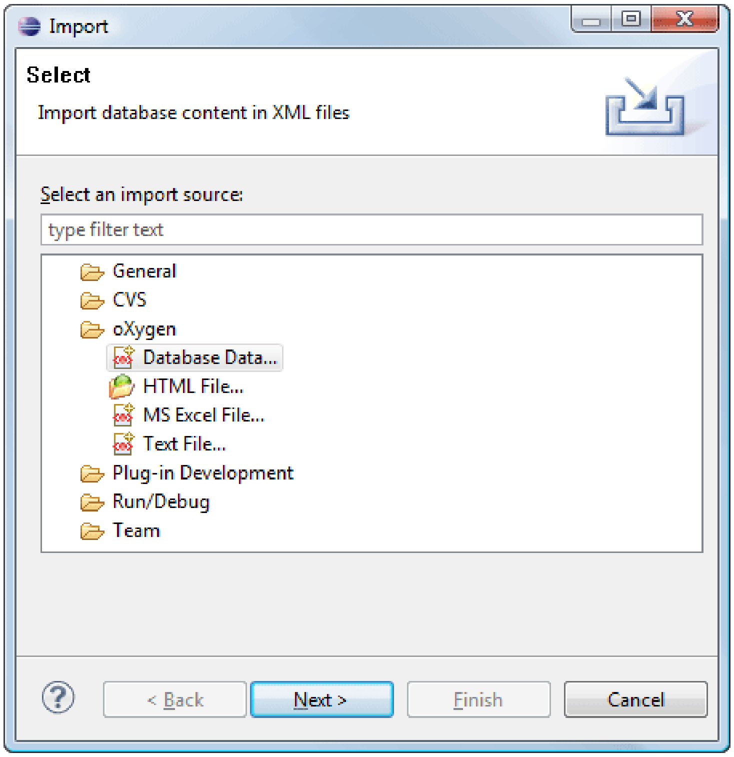


Figure 261: The Import Wizards of the Oxygen XML Editor plugin Plugin

Import from Database

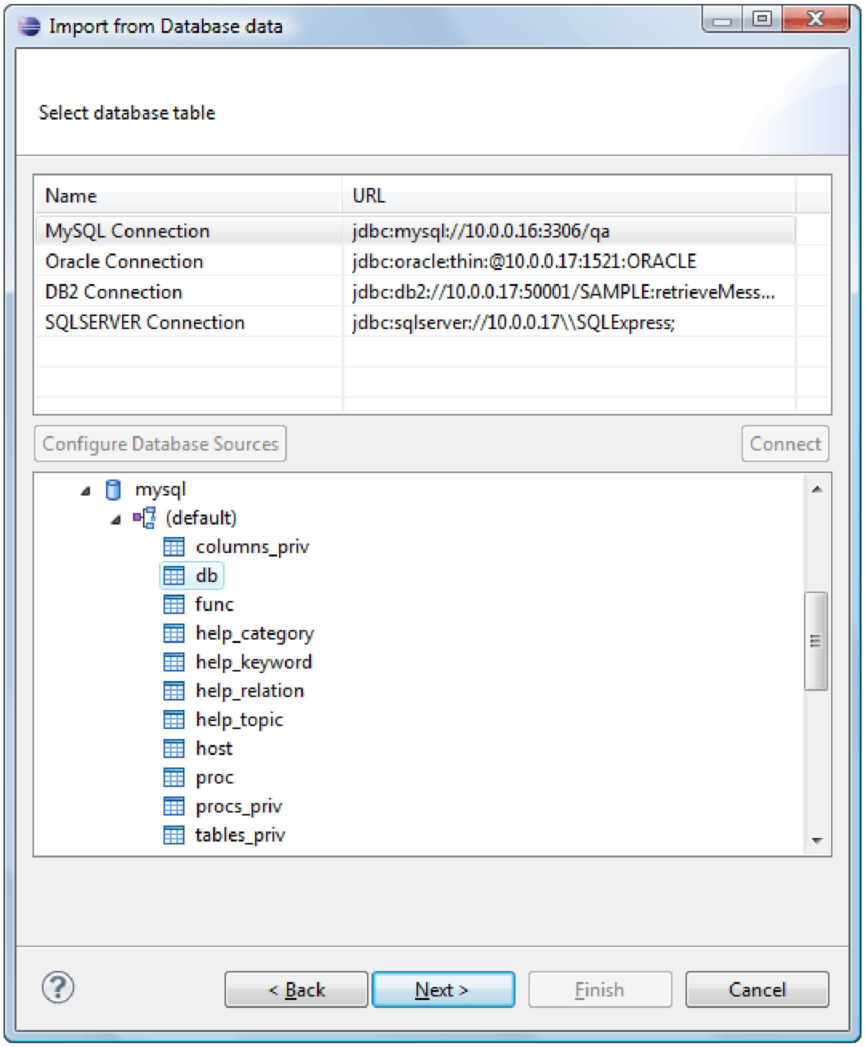
This section explains how to import data from a database into Oxygen XML Editor plugin .

Import Table Content as XML Document

The steps for importing the data from a relational database table are the following:

1. Go to menu **File** > **Import** > **oXygen / Database Data**.

   Clicking this action will open a dialog with all the defined database connections:



Name	URL
MySQL Connection	jdbc:mysql://10.0.0.16:3306/qa
Oracle Connection	jdbc:oracle:thin:@10.0.0.17:1521:ORACLE
DB2 Connection	jdbc:db2://10.0.0.17:50001/SAMPLE:retrieveMess...
SQLSERVER Connection	jdbc:sqlserver://10.0.0.17\\SQLExpress;

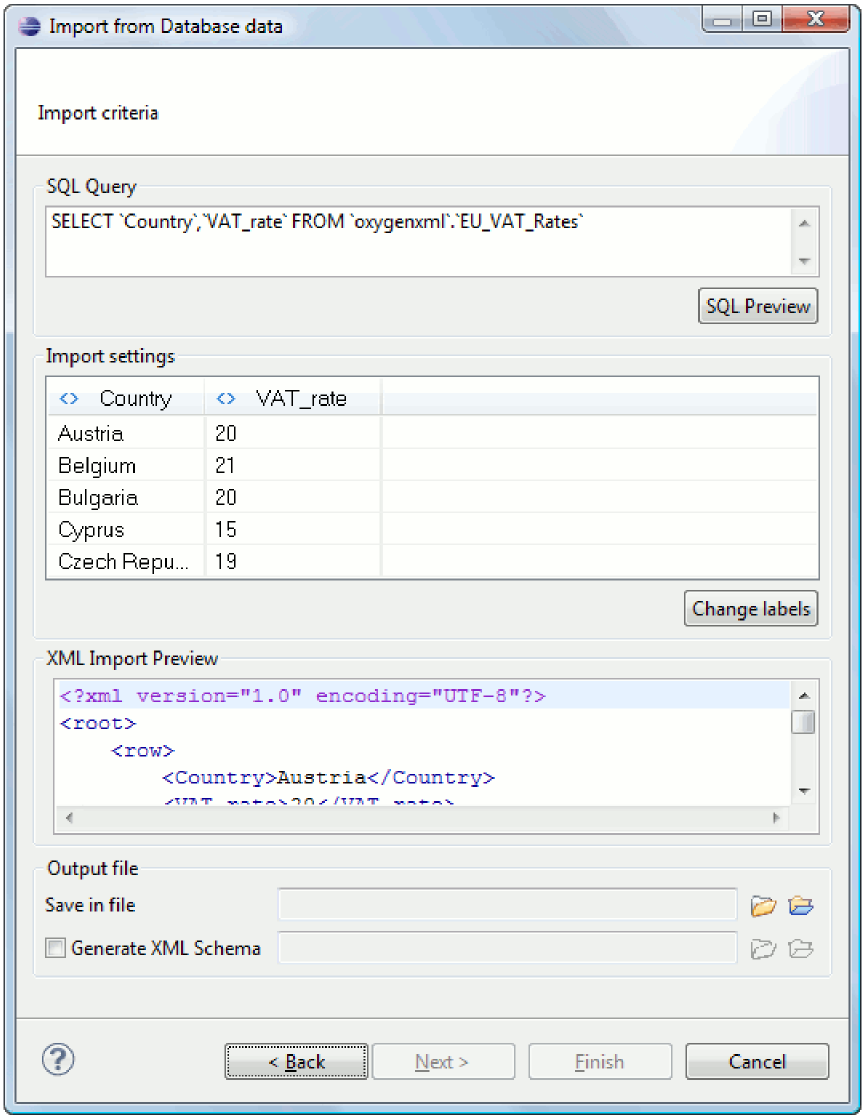
```
mysql
  (default)
    columns_priv
    db
    func
    help_category
    help_keyword
    help_relation
    help_topic
    host
    proc
    procs_priv
    tables_priv
```

Figure 262: 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:



SQL Query: SELECT `Country`,`VAT_rate` FROM `oxygenxml`.`EU_VAT_Rates`

<> Country	<> VAT_rate
Austria	20
Belgium	21
Bulgaria	20
Cyprus	15
Czech Repu...	19

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <row>
        <Country>Austria</Country>
        <VAT_rate>20</VAT_rate>
```

Figure 263: 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 Editor 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 Editor 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.

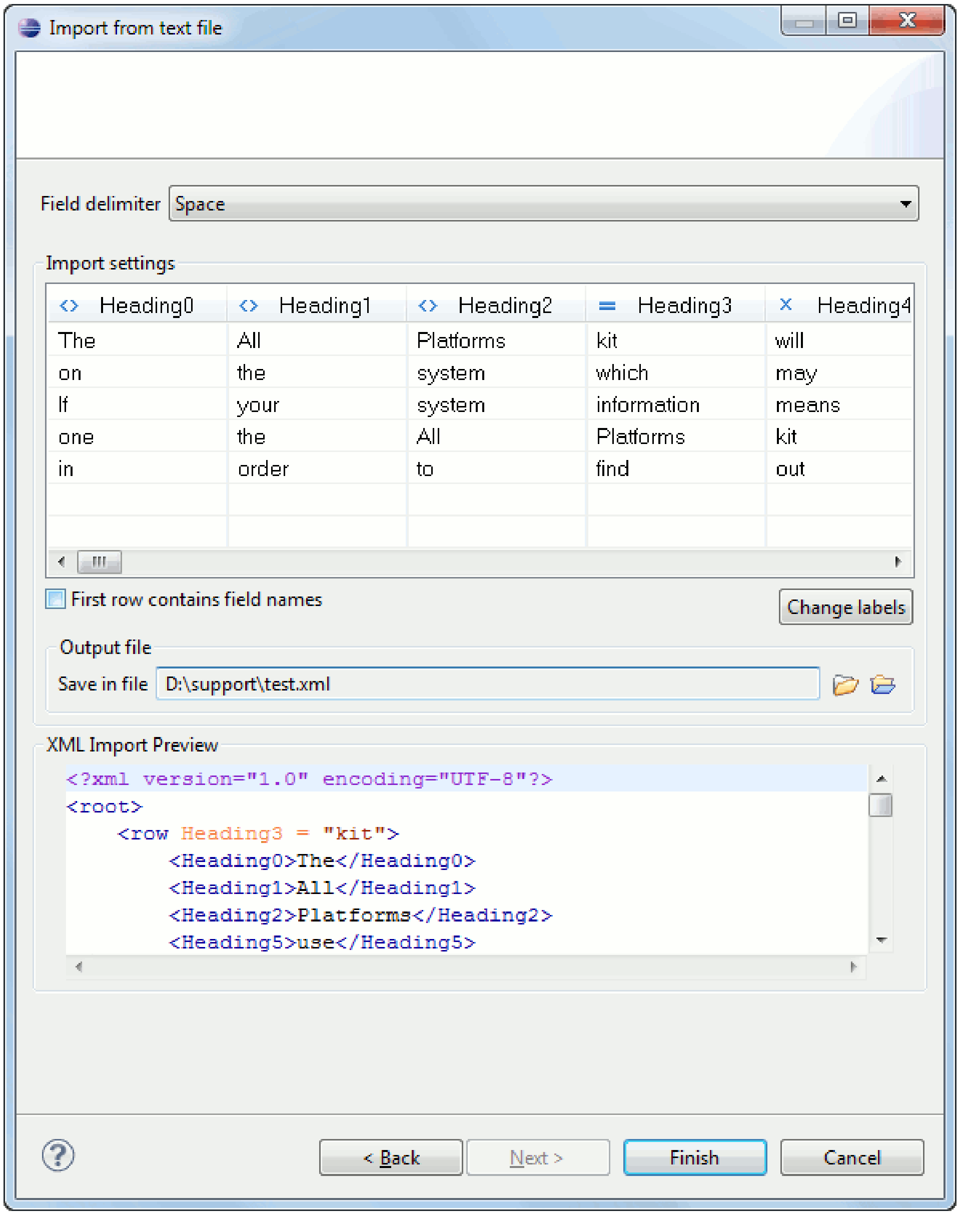
The resulted document will be an XHTML file containing a DOCTYPE declaration referring to the XHTML DTD definition on the Web. The parsed content of the imported file will be transformed to XHTML Transitional or XHTML Strict depending on what radio button the user chose when performing the import operation.

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:



Field delimiter: Space

Import settings

<> Heading0	<> Heading1	<> Heading2	= Heading3	× Heading4
The	All	Platforms	kit	will
on	the	system	which	may
If	your	system	information	means
one	the	All	Platforms	kit
in	order	to	find	out

□ First row contains field names

Output file

Save in file: D:\support\test.xml

XML Import Preview

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <row Heading3 = "kit">
        <Heading0>The</Heading0>
        <Heading1>All</Heading1>
        <Heading2>Platforms</Heading2>
        <Heading5>use</Heading5>
```

Figure 264: 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 suchinformation 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 24

Content Management System (CMS) Integration

Topics:

This chapter explains how Oxygen XML Editor 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 Editor plugin editor. Only the integration with the Documentum (CMS) is explained .

Integration with Documentum (CMS)

Oxygen XML Editor plugin provides support for browsing and managing Documentum repositories in the Data Source Explorer. You can easily create new resources on the repository, copy and move them using contextual actions or the drag and drop support, edit and transform the documents in the editor. The operations that can be performed on repository resources are described in the *Documentum (CMS) actions* section.

Oxygen XML Editor plugin supports Documentum (CMS) version 6.5 or later with *Documentum Foundation Services 6.5* or later installed.

Attention:

It is recommended to use the latest 1.6.x Java version. It is possible that the Documentum (CMS) support will not work properly if you use other Java versions.

Configure Connection to Documentum Server

This section explains how to configure a connection to a Documentum server.

How to Configure a Documentum (CMS) Data Source

To configure a Documentum (CMS) data source you need the Documentum Foundation Services Software Development Kit (*DFS SDK*) corresponding to your server version. The *DFS SDK* can be found in the Documentum (CMS) server installation kit or it can be downloaded from *EMC Community Network*.

Note: The *DFS SDK* can be found in the form of an archive named, for example, *emc-dfs-sdk-6.5.zip* for Documentum (CMS) 6.5.

1. Go to menu **Preferences** > **Data Sources**.
   The **Preferences** dialog is opened at the **Data Sources** panel.
2. In the **Data Sources** panel click the **New** button.
3. Enter a unique name for the data source.
4. Select **Documentum (CMS)** from the driver type combo box.
5. Press the **Choose DFS SDK Folder** button.
6. Select the folder where you have unpacked the *DFS SDK* archive file.

   If you have indicated the correct folder the following Java libraries (jar files) will be added to the list (some variation of the library names is possible in future versions of the *DFS SDK*):

   - `lib/java/emc-bpm-services-remote.jar`
   - `lib/java/emc-ci-services-remote.jar`
   - `lib/java/emc-collaboration-services-remote.jar`
   - `lib/java/emc-dfs-rt-remote.jar`
   - `lib/java/emc-dfs-services-remote.jar`
   - `lib/java/emc-dfs-tools.jar`
   - `lib/java/emc-search-services-remote.jar`
   - `lib/java/ucf/client/ucf-installer.jar`
   - `lib/java/commons/*.jar` (multiple jar files)
   - `lib/java/jaxws/*.jar` (multiple jar files)
   - `lib/java/utils/*.jar` (multiple jar files)

   Note: If for some reason the jar files are not found, you can add them manually by using the **Add Files** and **Add Recursively** buttons and navigating to the `lib/java` folder from the *DFS SDK*.

7. Click the **OK** button to finish the data source configuration.

How to Configure a Documentum (CMS) Connection

The steps for configuring a connection to a Documentum (CMS) server are the following:

1. Go to menu **Preferences** > **Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the previously configured Documentum (CMS) data sources in the **Data Source** combo box.
5. Fill-in the connection details:
   - **URL** - The URL to the Documentum (CMS) server: `http://<hostname>:<port>`
   - **User** - The user name to access the Documentum (CMS) repository.
   - **Password** - The password to access the Documentum (CMS) repository.
   - **Repository** - The name of the repository to log into.
6. Click the **OK** button to finish the configuration of the connection.

Known Issues

The following are known issues with the Documentum (CMS):

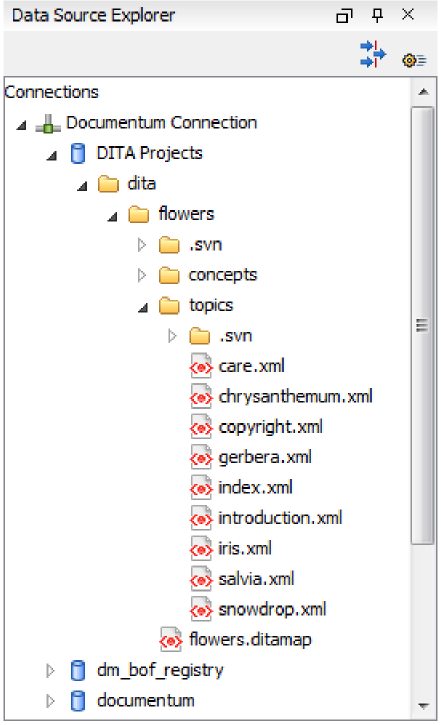
1. Please note that at the time of this implementation there is a problem in the UCF Client implementation for MAC OS X which prevents you from viewing or editing XML documents from the repository. The UCF Client is the component responsible for file transfer between the repository and the local machine. This component is deployed automatically from the server.
2. In order for the Documentum driver to work faster, you need to specify to the JVM to use a weaker random generator, instead of the very slow native implementation. This can be done by modifying in the Oxygen XML Editor plugin startup scripts (or in the *.vmoptions file) the system property:

```
-Djava.security.egd=file:/dev/./urandom
```

Documentum (CMS) Actions in the Data Source Explorer View

Oxygen XML Editor plugin allows you to browse the structure of a Documentum repository in the **Data Source Explorer** view and perform various operations on the repository resources.

You can drag and drop folders and resources to other folders to perform move or copy operations with ease. If the drag and drop is between resources (drag the child item to the parent item) you can create a relationship between the respective resources.



Data Source Explorer

Connections
- Documentum Connection
  - DITA Projects
    - dita
      - flowers
        - .svn
        - concepts
        - topics
          - .svn
          - care.xml
          - chrysanthemum.xml
          - copyright.xml
          - gerbera.xml
          - index.xml
          - introduction.xml
          - iris.xml
          - salvia.xml
          - snowdrop.xml
        - flowers.ditamap
  - dm_bof_registry
  - documentum

Figure 265: 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**).

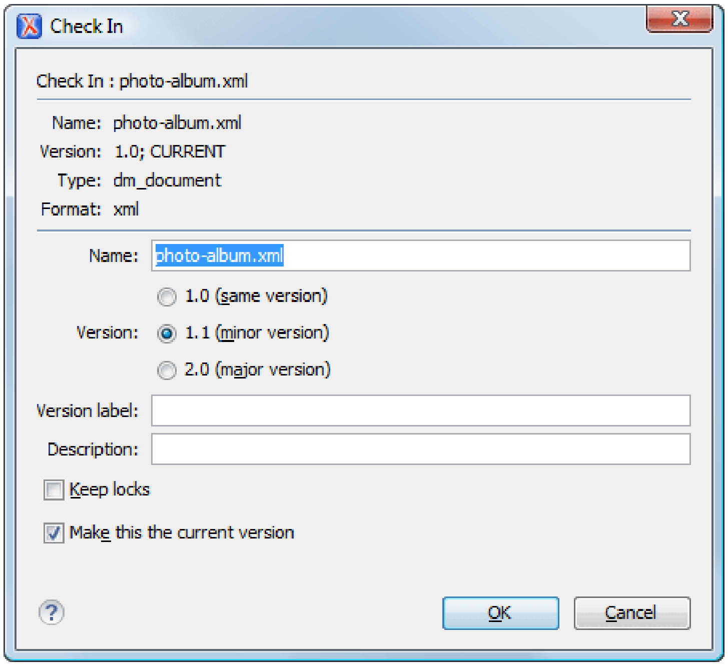
  - **Format** - The document content type format.

- **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.



Check In : photo-album.xml

Name: photo-album.xml
Version: 1.0; CURRENT
Type: dm_document
Format: xml

Name: photo-album.xml

Version: ○ 1.0 (same version) ● 1.1 (minor version) ○ 2.0 (major version)

Version label:

Description:

□ Keep locks

☑ Make this the current version

Figure 266: 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 Editor plugin comes with the DITA Open Toolkit which is able to transform a DITA map to various output formats. However DITA Open Toolkit requires local DITA files so first you need to check out a local version of your DITA content. Once you have a local version of a DITA map just load it in *the DITA Maps Manager view* and run one of the DITA transformations that are predefined in Oxygen XML Editor plugin or a customization of such a predefined DITA transformation.

Chapter 25

Composing Web Service Calls

Topics:

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 Editor 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 Editor 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 Editor plugin also checks if the WSDL file conforms with the WSDL specification (available only for WSDL 1.1 and SOAP 1.1).

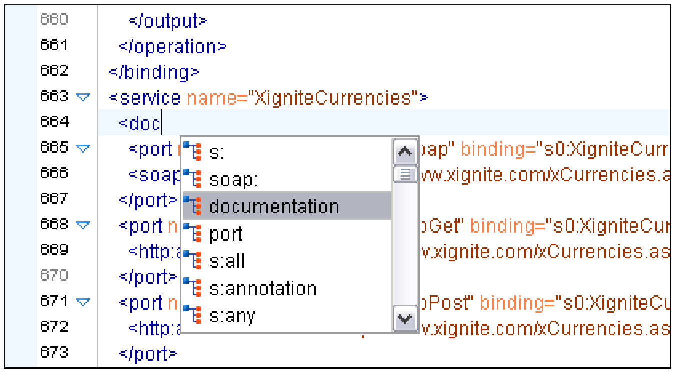
After you edit and validate your Web service descriptor against a mix of the XML Schemas for WSDL and SOAP, it is easy to check if the defined SOAP messages are accepted by the remote Web Services server using Oxygen XML Editor plugin 's **WSDL SOAP Analyser** integrated tool.

Composing a SOAP Request

To design, compose, and test Web service calls in Oxygen XML Editor 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 Editor plugin comes with these schemas and uses them by default to assist the user in editing Web Service descriptors.

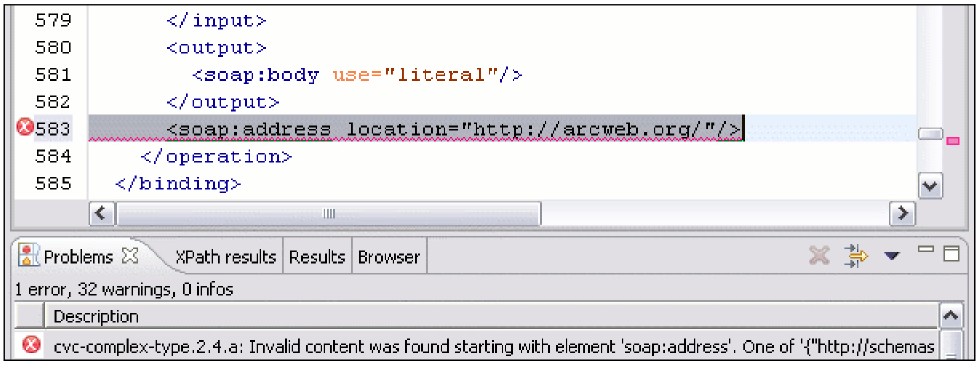


```
    </output>
  </operation>
</binding>
<service name="XigniteCurrencies">
  <doc
    <port                        ap" binding="s0:XigniteCurr
    <soap                        ww.xignite.com/xCurrencies.a
  </port>
  <port n                        pGet" binding="s0:XigniteCu
    <http:                       v.xignite.com/xCurrencies.as
  </port>
  <port n                        pPost" binding="s0:XigniteCu
    <http:                       v.xignite.com/xCurrencies.as
  </port>
```

Figure 267: 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.



```
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
        <soap:address location="http://arcweb.org/"/>
      </operation>
    </binding>
```

1 error, 32 warnings, 0 infos

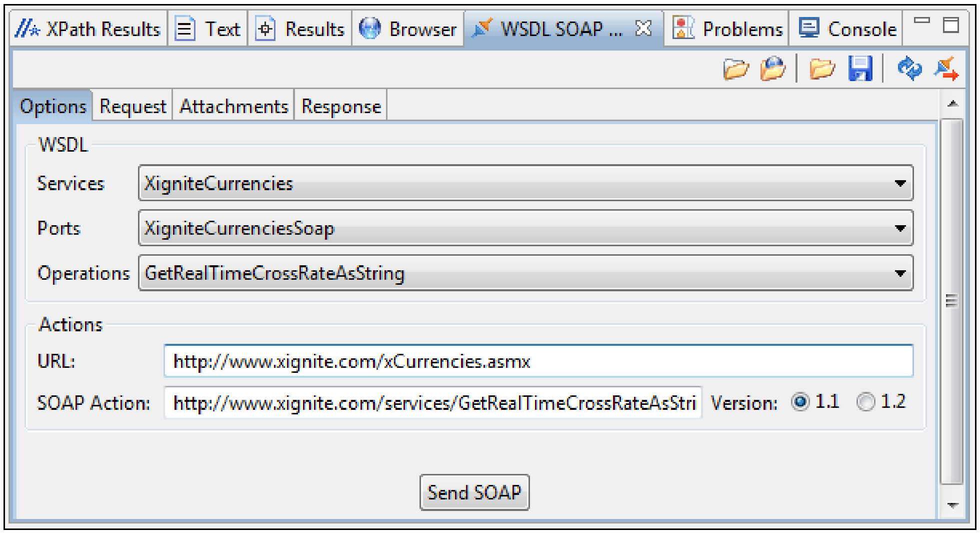
Description
cvc-complex-type.2.4.a: Invalid content was found starting with element 'soap:address'. One of '{"http://schemas

Figure 268: Validating a WSDL file

4. Check if the defined messages are accepted by the Web Services server.

   Oxygen XML Editor 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



WSDL

Services: XigniteCurrencies

Ports: XigniteCurrenciesSoap

Operations: GetRealTimeCrossRateAsString

Actions

URL: http://www.xignite.com/xCurrencies.asmx

SOAP Action: http://www.xignite.com/services/GetRealTimeCrossRateAsStri Version: ● 1.1 ○ 1.2

Figure 269: WSDL SOAP Analyser

This dialog contains a SOAP analyser and sender for Web Services Description Language file types. The analyser fields are:

- **Services** - The list of services defined by the WSDL file.
- **Ports** - The ports for the selected service.
- **Operations** - The list of available operations for the selected service.
- **Action URL** - Shows the script that serves the operation.
- **SOAP Action** - Identifies the action performed by the script.
- **Version** - Choose between 1.1 and 1.2. The SOAP version is selected automatically depending on the selected port.
- **Request Editor** - It allows you to compose the web service request. When an action is selected, Oxygen XML Editor plugin tries to generate as much content as possible for the SOAP request. The envelope of the SOAP request has the correct namespace for the selected SOAP version, that is *http://schemas.xmlsoap.org/soap/envelope/* for SOAP 1.1 or *http://www.w3.org/2003/05/soap-envelope* for SOAP 1.2. Usually you just have to change few

values in order for the request to be valid. The content completion assistant is available for this editor and is driven by the schema that defines the type of the current message. While selecting different operations, Oxygen XML Editor plugin remembers the modified request for each one. You can press the **Regenerate** button in order to overwrite your modifications for the current request with the initial generated content.

- **Attachments List** - You can define a list of file URLs to be attached to the request.
- **Response Area** - Initially it displays an auto generated server sample response so you can have an idea about how the response looks like. After pressing the **Send** button, it presents the message received from the server in response to the Web Service request. It may show also error messages. In case the response message contains attachments, Oxygen XML Editor plugin prompts you to save them, then tries to open them with the associated system application.
- **Errors List** - There may be situations in which the WSDL file is respecting the WSDL XML Schema, but it fails to be valid for example in the case of a message that is defined by means of an element that is not found in the types section of the WSDL. In such a case, the errors are listed here. This list is presented only when there are errors.
- **Send Button** - Executes the request. A status dialog is shown when Oxygen XML Editor plugin is connecting to the server.

The testing of a WSDL file is straight-forward: click the WSDL analysis button, then select the service, the port, and the operation. The editor generates the skeleton for the SOAP request. You can edit the request, eventually attach files to it and send it to the server. Watch the server response in the response area. You can find more details in the *Testing Remote WSDL Files* section.

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:

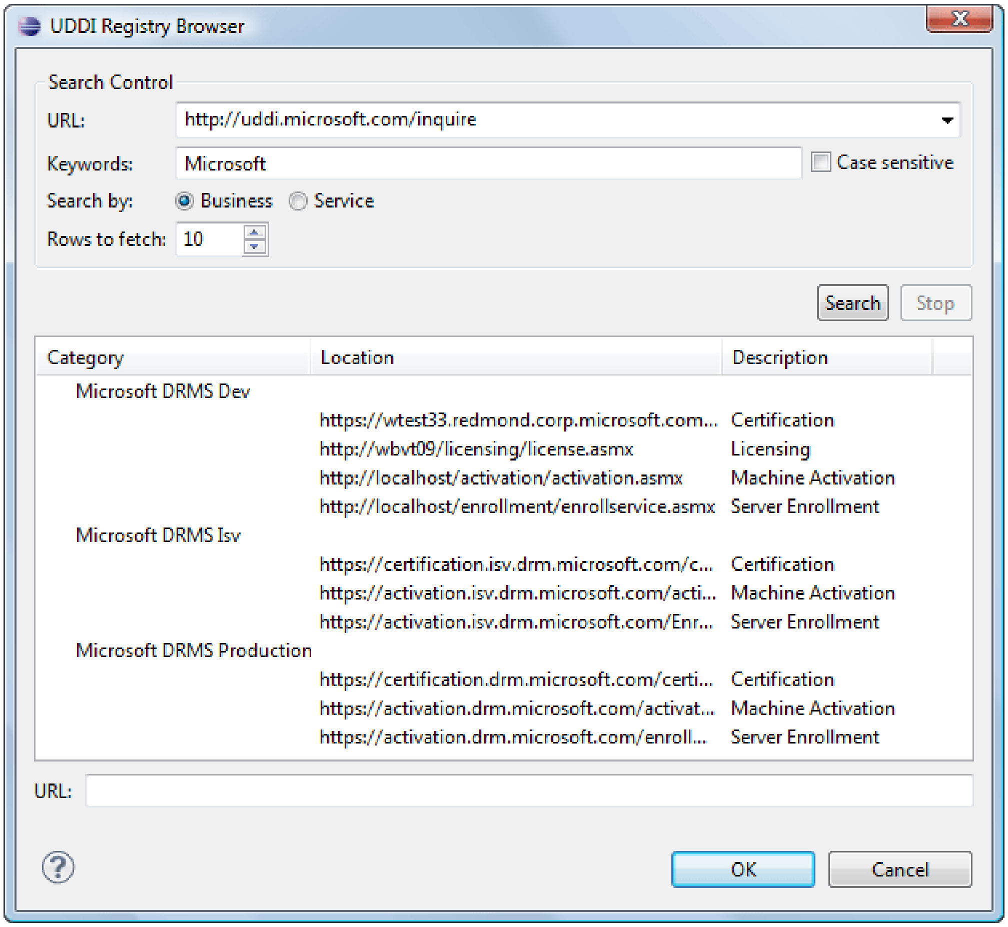
1. Go to menu **Window** > **Show View** > **Other** > **oXygen** > **WSDL SOAP Analyser ...**.
2. Press the **Choose WSDL** button and enter the URL of the remote WSDL file.

   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 Analyzer**) opens the **UDDI Registry Browser** dialog.



UDDI Registry Browser

Search Control

URL: http://uddi.microsoft.com/inquire

Keywords: Microsoft   □ Case sensitive

Search by: ● Business ○ Service

Rows to fetch: 10

Category	Location	Description
Microsoft DRMS Dev		
	https://wtest33.redmond.corp.microsoft.com...	Certification
	http://wbvt09/licensing/license.asmx	Licensing
	http://localhost/activation/activation.asmx	Machine Activation
	http://localhost/enrollment/enrollservice.asmx	Server Enrollment
Microsoft DRMS Isv		
	https://certification.isv.drm.microsoft.com/c...	Certification
	https://activation.isv.drm.microsoft.com/acti...	Machine Activation
	https://activation.isv.drm.microsoft.com/Enr...	Server Enrollment
Microsoft DRMS Production		
	https://certification.drm.microsoft.com/certi...	Certification
	https://activation.drm.microsoft.com/activat...	Machine Activation
	https://activation.drm.microsoft.com/enroll...	Server Enrollment

URL:

Figure 270: 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 26

Digital Signatures

Topics:

This chapter explains how to apply and verify digital signatures on XML documents.

Overview

Digital signatures are widely used as security tokens, not just in XML. A digital signature provides a mechanism for assuring integrity of data, the authentication of its signer, and the non-repudiation of the entire signature to an external party:

- A digital signature must provide a way to verify that the data has not been modified or replaced to ensure integrity.
- The signature must provide a way to establish the identity of the data's signer for authentication.
- The signature must provide the ability for the data's integrity and authentication to be provable to a third party for non-repudiation.

A public key system is used to create the digital signature and it's also used for verification. The signature binds the signer to the document because digitally signing a document requires the originator to create a hash of the message and then encrypt that hash value with his own private key. Only the originator has that private key and he is the only one that can encrypt the hash so that it can be unencrypted using his public key. The recipient, upon receiving both the message and the encrypted hash value, can decrypt the hash value, knowing the originator's public key. The recipient must also try to generate the hash value of the message and compare the newly generated hash value with the unencrypted hash value received from the originator. If the hash values are identical, it proves that the originator created the message, because only the actual originator could encrypt the hash value correctly.

XML Signatures can be applied to any digital content (data object), including XML (see W3C Recommendation, *XML-Signature Syntax and Processing*). An XML Signature may be applied to the content of one or more resources:

- enveloped or enveloping signatures are applied over data within the same XML document as the signature
- detached signatures are applied over data external to the signature element; the signature is "detached" from the content it signs. This definition typically applies to separate data objects, but it also includes the instance where the signature and data object reside within the same XML document but are sibling elements.

The XML Signature is a method of associating a key with referenced data. It does not normatively specify how keys are associated with persons or institutions, nor the meaning of the data being referenced and signed.

The original data is not actually signed. Instead, the signature is applied to the output of a chain of canonicalization and transformation algorithms, which are applied to the data in a designated sequence. This system provides the flexibility to accommodate whatever "normalization" or desired preprocessing of the data that might be required or desired before subjecting it to being signed.

To canonicalize something means to put it in a standard format that everyone generally uses. Because the signature is dependent on the content it is signing, a signature produced from a not canonicalized document could possibly be different from one produced from a canonicalized document. The canonical form of an XML document is physical representation of the document produced by the method described in this specification. The term canonical XML refers to XML that is in canonical form. The XML canonicalization method is the algorithm defined by this specification that generates the canonical form of a given XML document or document subset. The term XML canonicalization refers to the process of applying the XML canonicalization method to an XML document or document subset. XML canonicalization is designed to be useful to applications that require the ability to test whether the information content of a document or document subset has been changed. This is done by comparing the canonical form of the original document before application processing with the canonical form of the document result of the application processing.

A digital signature over the canonical form of an XML document or document subset would allows the signature digest calculations to be oblivious to changes in the original document's physical representation. During signature generation, the digest is computed over the canonical form of the document. The document is then transferred to the relying party, which validates the signature by reading the document and computing a digest of the canonical form of the received document. The equivalence of the digests computed by the signing and relying parties (and hence the equivalence of the canonical forms over which they were computed) ensures that the information content of the document has not been altered since it was signed.

The following canonicalization algorithms are used in Oxygen XML Editor plugin : Canonical XML (or Inclusive XML Canonicalization)(*XMLC14N*) and Exclusive XML Canonicalization(*EXCC14N*). The first is used for XML where the context doesn't change while the second was designed for canonicalization where the context might change.

Inclusive Canonicalization copies all the declarations, even if they are defined outside of the scope of the signature. In this way all the declarations you might use will be unambiguously specified. A problem appears when the signed XML is moved into another XML document which has other declarations because the Inclusive Canonicalization will copy then and the signature will be invalid.

Exclusive Canonicalization finds out what namespaces you are actually using (the ones that are a part of the XML syntax) and just copies those. It does not look into attribute values or element content, so the namespace declarations required to process these are not copied.

This type of canonicalization is useful when you have a signed XML document that you wish to insert into other XML documents and it will insure the signature verifies correctly every time, so it is required when you need self-signed structures that support placement within different XML contexts.

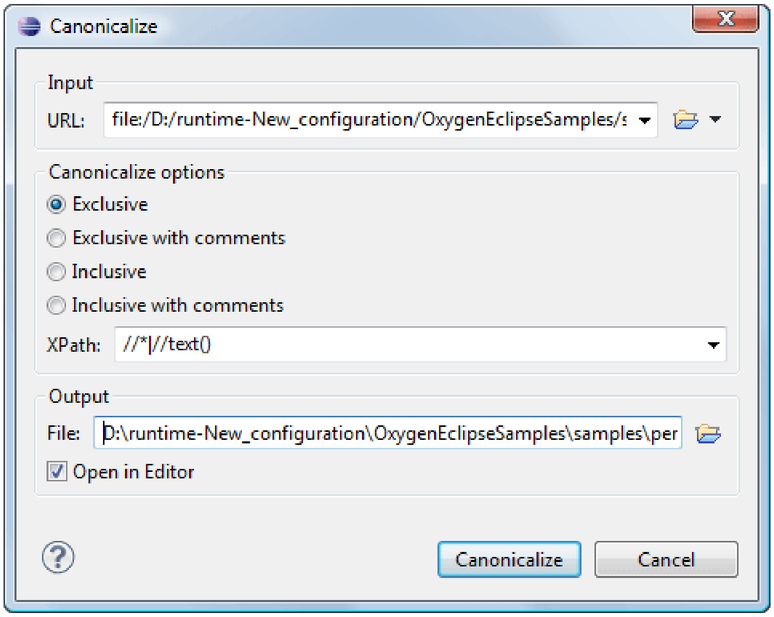
Inclusive Canonicalization is useful when it is less likely that the signed data will be inserted in other XML document and it's the safer method from the security perspective because it requires no knowledge of the data that are to be secured in order to safely sign them.

The canonicalization method can specify whether or not comments should be included in the canonical form output by the XML canonicalization method. If a canonical form contains comments corresponding to the comment nodes in the input node-set, the result is called canonical XML with comments. In an uncommented canonical form comments are removed, including delimiter for comments outside document element.

These three operations: Digital Signing, Canonicalization and Verification of the signature are available from the **Tools** menu or from the Editor's **contextual menu** > **Source**.

Canonicalizing Files

The user can select the canonicalization algorithm to be used for his document from the following dialog displayed by the action **Canonicalize** available from the editor panel's **contextual menu** > **Source** .



Canonicalize

Input

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

Canonicalize options

● Exclusive

○ Exclusive with comments

○ Inclusive

○ Inclusive with comments

XPath: //*|//text()

Output

File: D:\runtime-New_configuration\OxygenEclipseSamples\samples\per

☑ Open in Editor

Figure 271: Canonicalization settings dialog

The fields of the dialog are the following:

- **URL** - Specifies the location of the input URL.
- **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.
- **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
- **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.

- **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.
- **XPath** - The XPath expression provides the fragments of the XML document to be signed.
- **Output** - Specifies the output file path where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in the editor.

Certificates

A certificate is a digitally signed statement from the issuer (an individual, an organization, a website or a firm), saying that the public key (and some other information) of some other entity has a particular value. When data is digitally signed, the signature can be verified to check the data integrity and authenticity. Integrity means that the data has not been modified. Authenticity means the data comes indeed from the entity that claims to have created and signed it. Certificates are kept in special repositories called keystores.

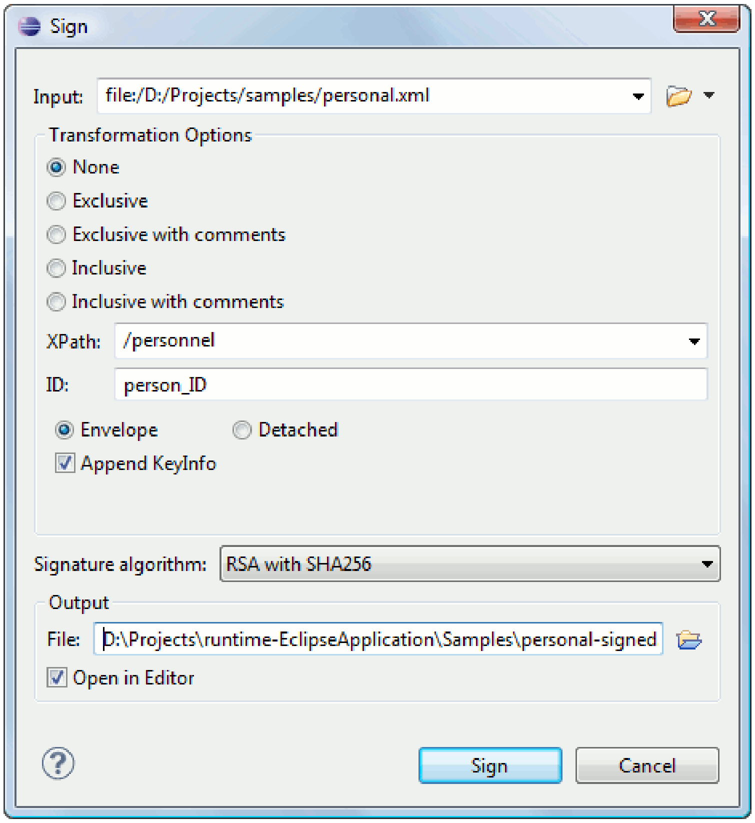
A *keystore* is an encrypted file that contains private keys and certificates. All keystore entries (key and trusted certificate entries) are accessed via unique aliases. An alias must be assigned for every new entry of either a key or certificate as a reference for that entity. No keystore can store an entity if it's alias already exists in that keystore and no keystore can store trusted certificates generated with keys in it's keystore.

In Oxygen XML Editor plugin there are provided two types of keystores: Java Key Store (JKS) and Public-Key Cryptography Standards version 12 (PKCS-12). A keystore file is protected by a password. In a PKCS 12 keystore you should not store a certificate without alias together with other certificates, with or without alias, as in such a case the certificate without alias cannot be extracted from the keystore.

To set the options for a certificate or to validate it, go to menu ***Options*** > ***Preferences*** > ***Certificates*** .

Signing Files

The user can select the type of signature to be used for his document from the following dialog displayed by the action **Sign** available from the editor panel's **contextual menu** > **Source**



Sign

Input: file:/D:/Projects/samples/personal.xml

Transformation Options

● None

○ Exclusive

○ Exclusive with comments

○ Inclusive

○ Inclusive with comments

XPath: /personnel

ID: person_ID

● Envelope ○ Detached

☑ Append KeyInfo

Signature algorithm: RSA with SHA256

Output

File: D:\Projects\runtime-EclipseApplication\Samples\personal-signed

☑ Open in Editor

Figure 272: 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 27

Text Editor Specific Actions

Topics:

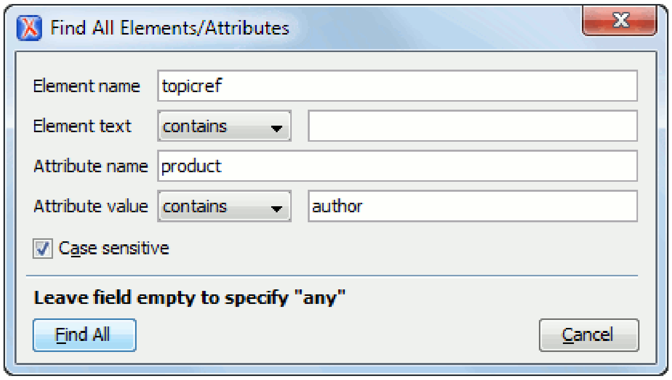
The Text mode of the editor panel provides the usual actions specific for a plain text editor: undo / redo, copy / paste, find / replace, etc. These actions are executed from the menu bar or toolbar and also by invoking their usual keyboard shortcuts.

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.



Find All Elements/Attributes

Element name: topicref

Element text: contains

Attribute name: product

Attribute value: contains author

☑ Case sensitive

Leave field empty to specify "any"

Figure 273: 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.

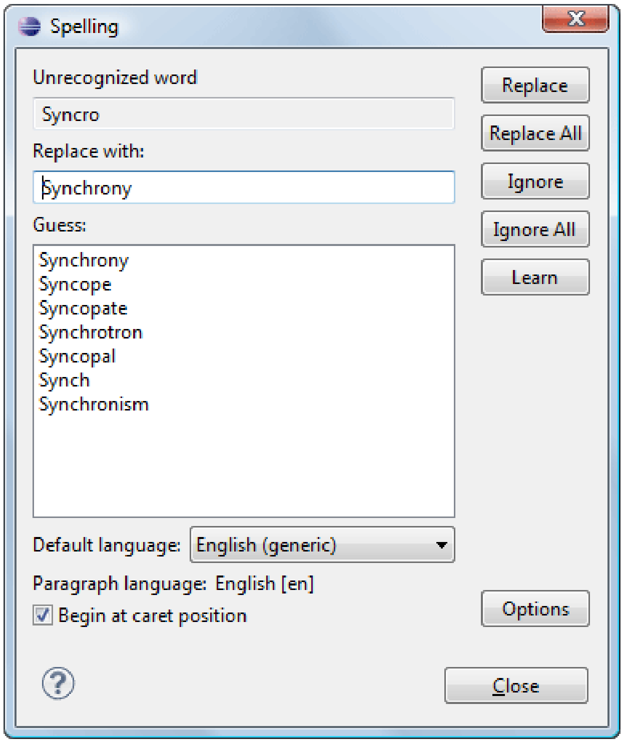
  If you leave the field empty but select **equals** in the combo box, only elements that have at least an attribute with an empty value will be found.

  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**.



Spelling

Unrecognized word

Syncro

Replace with:

Synchrony

Guess:

Synchrony
Syncope
Syncopate
Synchrotron
Syncopal
Synch
Synchronism

Default language: English (generic)

Paragraph language: English [en]

☑ Begin at caret position

Figure 274: The Check Spelling Dialog

The dialog contains the following fields:

- **Unrecognized word** - Contains the word that cannot be found in the selected dictionary. The word is also highlighted in the XML document.
- **Replace with** - The character string which is suggested to replace the unrecognized word.
- **Guess** - Displays a list of words suggested to replace the unknown word. Double click a word to automatically insert it in the document and resume the spell checking process.
- **Default language** - Allows you to select the default dictionary used by the spelling engine.
- **Paragraph language** - In an XML document you can mix content written in different languages. To tell the spell checker engine what language was used to write a specific section, you need to set the language code in the `lang` or `xml:lang` attribute to that section. Oxygen XML Editor plugin automatically detects such sections and instructs the spell checker engine to apply the appropriate dictionary.
- **Replace** - Replaces the currently highlighted word in the XML document, with the selected word in the **Replace with** field.
- **Replace All** - Replaces all occurrences of the currently highlighted word in the XML document, with the selected word in the **Replace with** field.
- **Ignore** - Ignores the first occurrence of the unrecognized word and allows you to continue checking the document. Oxygen XML Editor plugin skips the content of the XML elements *marked as ignorable*.
- **Ignore All** - Ignores all instances of the unknown word in the current document.
- **Learn** - Includes the unrecognized word in the list of valid words.
- **Options** - Sets the *configuration options of the spell checker.*
- **Begin at caret position** - Instructs the spell checker to begin checking the document starting from the current cursor position.

- **Close** - Closes the dialog.

Spell Checking Dictionaries

There are two spell checking engines available in Oxygen XML Editor plugin : **Hunspell** checker (default setting) and **Java** checker. You can set the spell check engine in the ***Spell checking engine*** preferences page. The dictionaries used by the two engines differ in format, so you need to follow specific procedures in order to add another dictionary to your installation of Oxygen XML Editor plugin .

Dictionaries for the Hunspell Checker

The Hunspell spell checker is open source and has LGPL license. The format of the Hunspell spell dictionary is supported by Mozilla, OpenOffice and the Chrome browser. Oxygen XML Editor plugin comes with the following built-in dictionaries for the Hunspell checker:

- English (US)
- English (UK)
- French
- German (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 Editor 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 Editor plugin preferences folder only if it is a new dictionary (it is not available as built-in dictionary in Oxygen XML Editor plugin).

   The Oxygen XML Editor 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 Editor plugin comes with the following built-in dictionaries for the Java checker:

- English (US)
- English (UK)
- English (Canada)
- French (France)
- French (Belgium)
- French (Canada)

- French (Switzerland)
- German (old orthography)
- German (new orthography)
- Spanish

A pre-built dictionary can be added by copying the corresponding `.dar` file to the folder `[Oxygen-install-folder]/dicts` and restarting Oxygen XML Editor plugin . There is one dictionary for each language-country variant combination. 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 Editor plugin to automatically check the spelling as you write, you need to enable the **Automatic spell check** option from the *Spell Check* preferences page. Unknown words are highlighted and feature a contextual menu which offers the following:

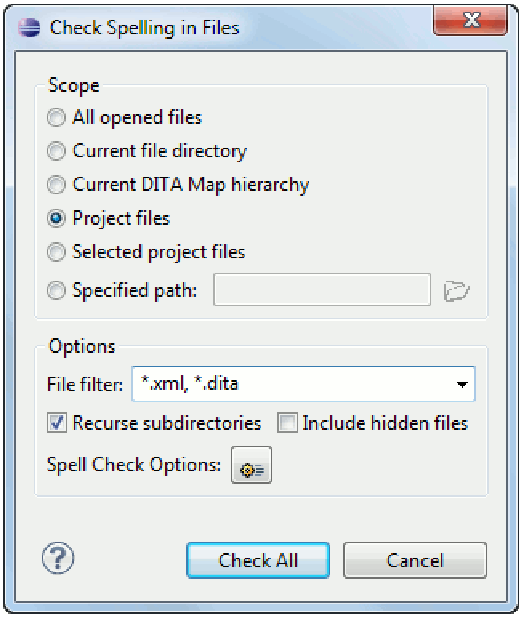
- **Delete Repeated Word** action - allows you to delete repeated words;
- a list of words suggested by the spell checking engine as possible replacements of the unknown word;
- **Learn Word** action - allows you to add the current unknown word to the persistent dictionary.

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.



Check Spelling in Files

Scope

○ All opened files

○ Current file directory

○ Current DITA Map hierarchy

● Project files

○ Selected project files

○ Specified path:

Options

File filter: *.xml, *.dita

☑ Recurse subdirectories ☐ Include hidden files

Spell Check Options:

Figure 275: Check Spelling in Files Dialog

The following scopes are available:

- **All opened files** - Spell check in all opened files.
- **Directory of the current file** - All the files from the folder of the current edited file.
- **Scope of the current DITA Map** - All the files referred in the current DITA map opened in the **DITA Maps Manager** view.
- **Project files** - All files from the current project.
- **Selected project files** - The selected files from the current project.
- **Specified path** - A custom path.

You can also choose a file filter, decide whether to recurse subdirectories or process hidden files.

The spell checker processor uses the options available in the ***Spell Check*** *preferences panel* .

Chapter 28

Configuring the Application

Topics:

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 Editor 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 Editor plugin comes with several distinct layers of option values, arranged here according to their priority, from low to high:

- *Default Options* - should be regarded as factory defaults or built-in values.

  This predefined set of values are tuned 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 Editor 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.

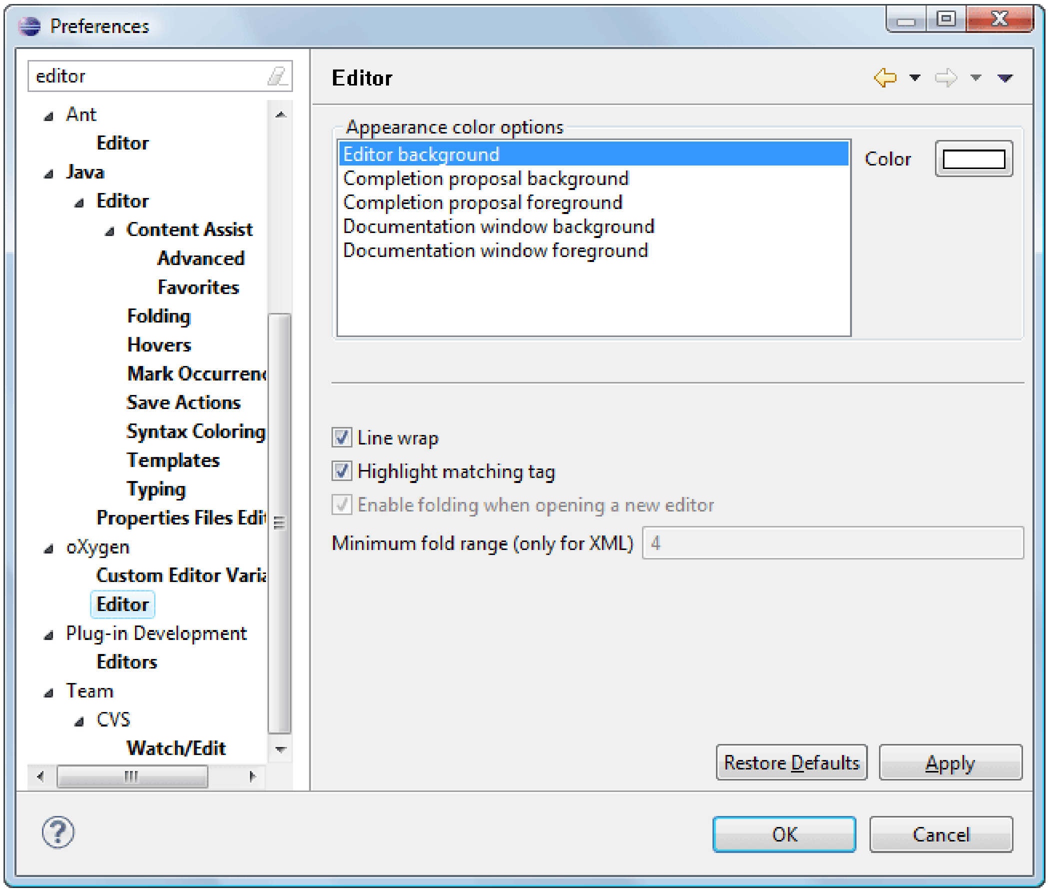
Preferences

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 Editor plugin plugin by right-clicking in the editor panel and selecting **Preferences**:



Appearance color options

- Editor background
- Completion proposal background
- Completion proposal foreground
- Documentation window background
- Documentation window foreground

Color

☑ Line wrap

☑ Highlight matching tag

☑ Enable folding when opening a new editor

Minimum fold range (only for XML) 4

Figure 276: Eclipse Preferences dialog - restricted version

Oxygen XML Editor 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 Editor plugin plugin: registration name, category and number of purchased licenses, encrypted signature of the license key. Clicking on the **Register** button opens the Oxygen XML Editor 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:

- **Use custom frameworks directory** - For editing different types of XML documents (for content completion, validation, authoring) Oxygen XML can use information from the document types which are stored in the `frameworks` subfolder of the application's install folder. If a custom `frameworks` folder is specified, then Oxygen XML loads the document types from this location.
- **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.
- **Author** - 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 *custom frameworks folder* 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

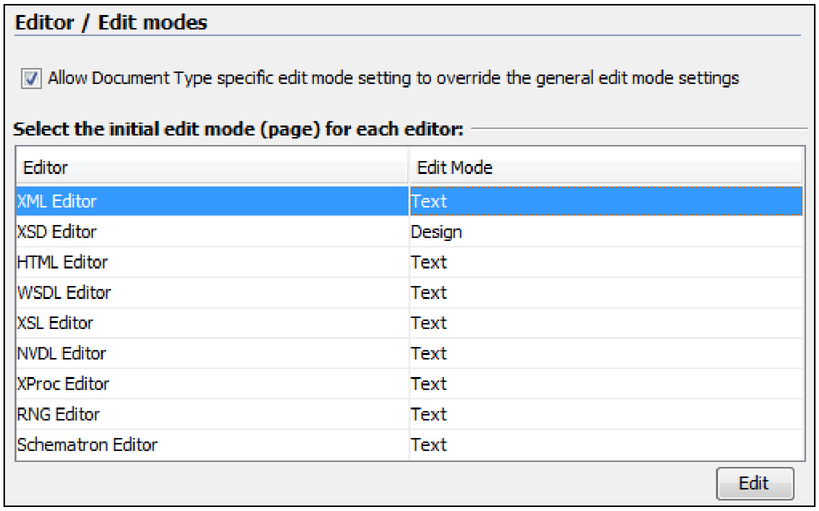
The **Edit modes** preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **Editor** > **Edit modes** and allows you to select the initial edit mode for an editor. The mode in which a file was edited in the previous session is saved and is used when the application is restarted 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 *the Document Type dialog* overrides the general edit mode setting from the table below.

The initial edit mode of each editor type has one of the following values:

- Text
- Author
- Grid
- Design (available only for the W3C XML Schema editor)

The Oxygen XML Editor plugin Edit modes Preferences Panel



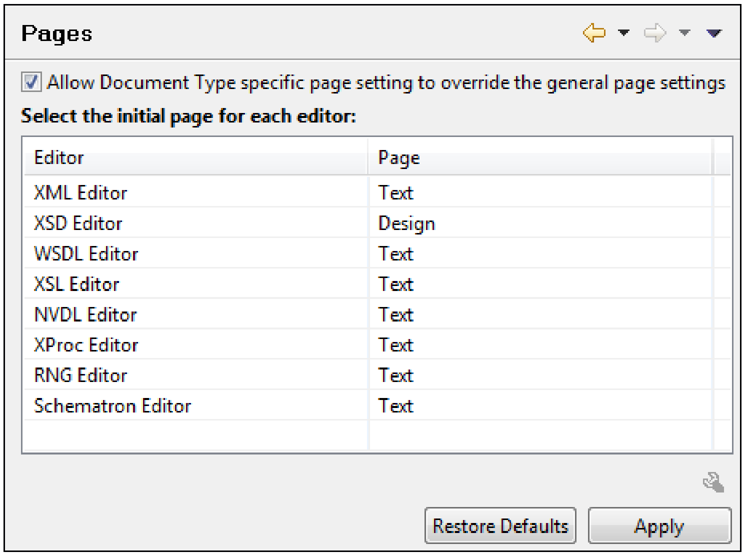
Editor / Edit modes

☑ Allow Document Type specific edit mode setting to override the general edit mode settings

Select the initial edit mode (page) for each editor:

Editor	Edit Mode
XML Editor	Text
XSD Editor	Design
HTML Editor	Text
WSDL Editor	Text
XSL Editor	Text
NVDL Editor	Text
XProc Editor	Text
RNG Editor	Text
Schematron Editor	Text

The Oxygen XML Editor plugin Edit modes Preferences Panel



Pages

☑ Allow Document Type specific page setting to override the general page settings

Select the initial page for each editor:

Editor	Page
XML Editor	Text
XSD Editor	Design
WSDL Editor	Text
XSL Editor	Text
NVDL Editor	Text
XProc Editor	Text
RNG Editor	Text
Schematron Editor	Text

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.

Author

The **Author** preferences page is opened from menu **Window** > **Preferences** > **oXygen** > **Editor** > **Edit modes** > **Author**.

The following options are available:

- **Author default background color** - Default background color of the Author editing mode. The `background-color` CSS property set in the CSS file associated with the current edited document overwrites this option.
- **Author default foreground color** - Default foreground color of the Author editing mode. The `color` CSS property set in the CSS file associated with the current edited document overwrites this option.
- **Highlight elements near caret** - Defines the background color of the element or elements at caret position.

- **Show caret position tooltip** - If enabled, the caret position information tooltip is displayed. More information about the position information tooltip can be found in the section *Position information tooltip*. The documentation tooltip can be disabled from the *Annotations preferences panel*.
- **Show placeholders for empty elements** - When enabled, placeholders are displayed for empty elements to make them clearly visible. A placeholder is rendered as a light grey box displaying the element name.
- **Show Author layout messages** - If enabled, all errors reported during layout creation are presented in the **Errors** view.
- **Hide comments** - Hides comments from the documents edited in Author mode.
- **Hide processing instructions** - Hides processing instructions from the documents edited in Author mode.
- **Hide doctype** - Hides *doctype* sections from the documents edited in Author mode.
- **Show very large images** - Enables the very large (bigger than 6 *megapixels*) images support in Author mode. Disabled by default.

  Important: If you enable this option and your document contains many such images, Oxygen XML may consume all available memory, throwing an *OutOfMemory error*. This means that you need to restart the application after you increase the available memory limit.

- **Display referred content (e.g.: entities, XInclude, DITA conref, etc.)** - When enabled, the references (like entities, XInclude, DITA conref) also display the content of the resources they refer. If you toggle this option while editing, reload the file for the modification to take effect. Enabled by default.
- **Format and indent** - Here you can set the format and indent method that is applied when a document is saved in Author mode, or when switching the editing mode (from Text to Author or vice versa):
  - **Only the modified content** - The **Save** operation only formats the nodes that were modified in Author mode.

    The rest of the document preserves its original formatting. Selected by default.
  - **The entire document** - The **Save** operation applies formatting to the entire document regardless of the nodes that were modified in Author mode.

    If the **Apply also 'Format and Indent' action as in 'Text' edit mode** option is enabled, the content of the document is formatted by applying the **Format and Indent** rules from the *Editor/Format/XML* option page. In this case, the result of the **Format and indent** operation will be the same as when it is applied in **Text** editing mode.

- **Quick up/down navigation** - Speeds up navigation when using up / down keys. The cursor jumps from line to line, without stopping at intermediate positions between element tags.
- **Tags display mode** - Default display mode for element tags presented in **Author** mode. You can choose between:
  - **Full Tags with Attributes** - This mode reveals the entire XML markup, easing the transition from a Text based editing to an Author mode editing. Tags contain element names, attribute names, and attribute values.
  - **Full Tags** - Displays name tags that enclose block and inline elements.
  - **Block Tags** - Displays a mix of element name tags that enclose block elements and compact tags that enclose the inline elements.
  - **Inline Tags** - Displays only name tags that enclose the inline elements.
  - **Partial Tags** - Displays compact tags that enclose the inline elements.
  - **No Tags** - No tags are displayed. This representation is as close as possible to the document published content.
- **Tags background color** - Allows you to configure the Author tags background color.
- **Tags foreground color** - Allows you to configure the Author tags foreground color.

Schema Aware

The **Schema Aware** preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **Editor** > **Edit modes** > **Author** > **Schema aware**.

- **Schema aware normalization, format, and indent**

When opening a document in Author, white spaces can be normalized or removed in order to obtain a more compact display. The reverse process takes place when saving the document in the Author. By default this algorithm is controlled by the CSS `display` property.

If this option is checked, then this process will be schema aware so the algorithm will take into account if the element is declared as element-only or mixed. It will also take into account options **Preserve space elements**, **Default space elements**, **Mixed content elements** from option page ***Window > Preferences > oXygen > Editor > Format > XML***

- **Indent blocks-only content**

  If checked, even if an element is declared in the schema as being mixed but it has a blocks-only content (as specified by the CSS property `display` of its children), it will be treated as being element-only.

- **Schema Aware Editing**

  Editing in Author is schema driven.

  - **On** - Enables all schema aware editing options.
  - **Off** - Disables all schema aware editing options.
  - **Custom** -
    - **Delete element tags with backspace and delete**

      Controls the behavior for deleting element tags using delete or backspace keys. Available options:

      - **Smart delete**

        If the result of the delete action is invalid, different strategies will be applied in order to keep the document valid. If backspace / delete is pressed at the beginning / end of an element the action that should take place is unwrap (the element will be deleted and its content will be put in its place). If its content is not accepted by the schema in that position, you can keep a valid document by applying different strategies like:

        - Search for a preceding (backspace case)/following (delete case) element in which you can append that content.
        - If the tag markers of the element to unwrap are not visible a caret move action in the delete action direction will be performed.

      - **Reject action when its result is invalid**

        If checked and the result of the delete action is invalid, the action will not be performed.

    - **Paste and Drag and Drop**

      Controls the behavior for paste and drag and drop actions. Available options:

      - **Smart paste and drag and drop**

        If the content inserted by a paste or drop action is not valid at the caret position, according to the schema, different strategies are applied to find an appropriate insert position, like:

        - If the sibling element can accept the content, then a new element with the same name as the sibling is created in which the content will be inserted.
        - You will iterate to the left or to the right of the insertion position, without leaving the current context, and try to insert the fragment in one of the encountered elements (that accepts the content to be inserted).

      - **Reject action when its result is invalid**

        If checked and the result of the paste or drop action is invalid, the action will not be performed.

    - **Typing**

      Controls the behavior that takes place when typing. Available options:

      - **Smart typing**

If the typed character cannot be inserted at element from the caret position then a sibling element that can accept it will be searched for. If the sibling element can accept the content, then a new element with the same name as the sibling is created in which the content will be inserted.

    - **Reject action when its result is invalid**

      If checked and the result of the typing action is invalid, the action will not be performed.

  - **Content Completion**

    Controls the behavior that takes place when inserting elements using content completion. Available options:

    - **Allow only insertion of valid elements and attributes**

      If checked, only elements or attributes form the content completion proposals list can be inserted in the document through content completion.

  - **Warn on invalid content when performing action**

    A warning message will be displayed when performing an action that will result in invalid content. Available options:

    - **Delete Element Tags**

      If checked, when the *Delete Element Tags* action will result in an invalid content, a warning message will be displayed in which the user is asked if the operation should continue.

    - **Join Elements**

      If checked, when the *Join Elements* action will result in an invalid content, a warning message will be displayed in which the user is asked if the operation should continue.

- **Convert external content on paste**

  If checked, Oxygen XML Editor plugin preserves the formatting style when you paste content copied from external applications (like web browsers or Office-like applications). This option is enabled by default and applies only to the major document type frameworks (DocBook, DITA, TEI, XHTML) and to those customized to support the *content conversion on paste*.

If the Schema Aware Editing is **On** or **Custom** all actions that can generate invalid content will be forwarded first toward *AuthorSchemaAwareEditingHandler*.

Review

The Author **Review** preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **Editor** > **Edit modes** > **Author** > **Review**.

The available preferences are:

- **Author** - The name of the user who performs the changes when **Track Changes** is active for a given editor. This information is associated with each performed change.
- **Initial Track Changes State** - Controls the initial **Track Changes** state.
  - **Stored in document** - Recommended when multiple editors work on the same set of documents because the state of **Track Changes** (enabled/disabled) is kept in the edited document. When you open such a document and the **Store in document** option is active, if the `<?oxy_options track_changes="on"?>` processing instruction was saved in the document, the **Track Changes** is enabled. When this option is enabled and you open a document in the Author mode, the **Track Changes** state is restored from the previous use of the document. This means that if another user edited the document with **Track Changes** turned on before sending it to you, you will also have **Track Changes** switched on when you open it in the Author mode.
  - **Always On** - The **Track Changes** feature is active when you open a document.
  - **Always Off** - The **Track Changes** feature is inactive when you open a document.

Note: Initial Track Changes State options do not affect documents already open in the Author mode.

- **Inserted content color** - Sets the color used for marking the inserted content. This includes the font foreground and background, and the underline that decorates all inserted text.
  - **Auto** - Automatically assigns colors for the insert changes for each author name. Enabled by default.
  - **Custom** - Uses a custom color for all insert changes, regardless of the author name.
  - **Use same color for text foreground** - Uses the same color to paint the inserted text foreground.
  - **Use same color for background** - Uses the same color for the insert text background. Use the slider to adjust the transparency level.
- **Deleted content color** - Sets the color used for marking the deleted content. This includes the font foreground and background, and the strike-through that decorates all deleted text.
  - **Auto** - Automatically assigns colors for the delete changes for each author name. Enabled by default.
  - **Custom** - Uses a custom color for all delete changes, regardless of the author name.
  - **Use same color for text foreground** - Uses the same color to paint the deleted text foreground.
  - **Use same color for background** - Uses the same color for the deleted text background. Use the slider to adjust the transparency level.
- **Commented content color** - Sets the background color of the annotated text.
  - **Auto** - Automatically assigns colors for the annotated content for each author name. Enabled by default.
  - **Custom** - Use a custom color for all annotated content, regardless of the author name. Use the slider to control the transparency level.

Profiling / Conditional Text

The Author **Profiling/Conditional Text** preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text**.

Two tables contain the profiling attributes and condition sets defined for the supported document types. You can customize your *profiling attributes* and *condition sets*, but Oxygen XML comes with predefined ones.

If you have two or more identically named entries that match the same document type, Oxygen XML uses the one that is positioned highest in the table. Use the **Up / Down** buttons to alter the priority of the entries.

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 atr1="v1"></a>`. When not checked, the same operation represents an empty element in a more compact form: `<a atr1="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 in any of the following cases:

- when the **Format and Indent** action is applied on an XML document;
- when you switch from **Text** mode to **Author** mode;
- when you switch from **Author** mode to **Text** mode;

The characters with the codes 9 (TAB), 10 (LF), 13 (CR) and 32 (SPACE) are always in the group of whitespace characters that must be normalized so they are always enabled in this panel.

CSS

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.

JavaScript

The **JavaScript** format preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **Editor** > **Format** > **JavaScript**.

The following options control the behavior of **Format and Indent** action:

- **Start curly brace on new line** - Opening curly braces start on a new line.
- **Preserve empty lines** - Empty lines in the JavaScript code are preserved. Enabled by default.
- **Allow formatting embedded JavaScript** - Applied only to XHTML documents, this option allows the application to format embedded JavaScript code, taking precedence over the *Schema aware format and indent* option. 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 Editor plugin closes the last open tag when you type `</`.
- **Automatically rename/delete matching tag** - Oxygen XML Editor 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*. This option is applied also in the **Author** mode of the XML editor.
    - **Add optional content** - Oxygen XML inserts the optional elements specified in the DTD, XML Schema, or RELAX NG schema. This option is applied also in the **Author** mode of the XML editor.
    - **Add first Choice particle** - Oxygen XML inserts the first **choice** particle specified in the DTD, XML Schema, or RELAX NG schema. This option is applied also in the **Author** mode of the XML editor.
- **Case sensitive search** - The search in the content completion assistant window when you type a character is case-sensitive ('a' and 'A' are different characters). This option is applied also in the **Author** mode of the XML editor.
- **Cursor position between tags** - When enabled, Oxygen XML 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. This option is applied also in the **Author** mode of the XML editor.
- **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. This option is applied also in the **Author** mode of the XML editor.
- **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.. This option is applied also in the **Author** mode of the XML editor.
- **Maximum number of recent items shown** - Limits the number of recently used items presented at the top of the content completion assistant window. This option is applied also in the **Author** mode of the XML editor.
- **Learn attributes values** - Oxygen XML learns the attribute values used in a document. This option is applied also in the **Author** mode of the XML editor.
- **Learn on open document** - Oxygen XML automatically learns the document structure when the document is opened. This option is applied also in the **Author** mode of the XML editor.
- **Learn words (Dynamic Abbreviations, available on CTRL+SPACE)** - 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. This option is applied also in the **Author** mode of the XML editor.
- **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 *the Author editing mode* one). This option is applied also in the **Author** mode of the XML editor.

- **Use DTD comments as annotation** - When enabled, Oxygen XML 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.

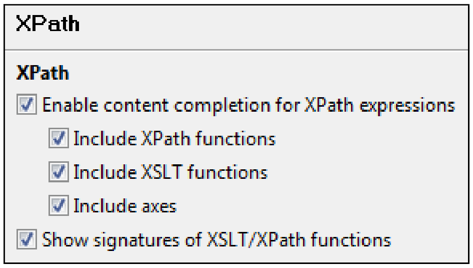
  If the detection fails, choose one of the following alternatives:

  - **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**.



XPath

XPath

☑ Enable content completion for XPath expressions

  ☑ Include XPath functions

  ☑ Include XSLT functions

  ☑ Include axes

☑ Show signatures of XSLT/XPath functions

Figure 277: 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 *the content completion assistant list* 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 Editor 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.

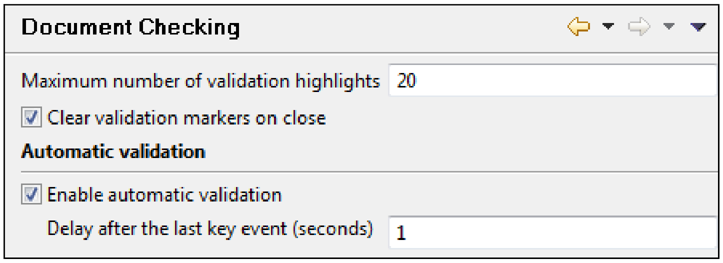
- **Delete learned words** - Press this button to open the list of learned words. Here you can select the items you want to remove.
- **Use "lang" and "xml:lang" attributes** - If enabled, the contents of any element with one of the `lang` or `xml:lang` attributes is checked using a dictionary suitable 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.



Document Checking

Maximum number of validation highlights: 20

☑ Clear validation markers on close

Automatic validation

☑ Enable automatic validation

Delay after the last key event (seconds): 1

Figure 278: Document Checking Preferences Panel

The error checking preferences are the following:

- **Maximum number of validation highlights** - If validation generates more errors than the number from this option only the first errors up to this number are highlighted in editor panel and on stripe that is displayed at right side of editor panel. This option is applied both for *automatic validation* and *manual validation*.
- **Clear validation markers on close** - If this option is selected all the error markers added in the **Problems** view for that document are removed when a document edited with the Oxygen XML Editor plugin plugin is closed.
- **Enable automatic validation** - Validation of edited document is executed in background as the document is modified by editing in Oxygen XML Editor plugin .
- **Delay after the last key event (s)** - The period of keyboard inactivity which starts a new validation (in seconds).

Mark Occurrences

The **Mark Occurrences** preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **Editor** > **Mark Occurrences**:

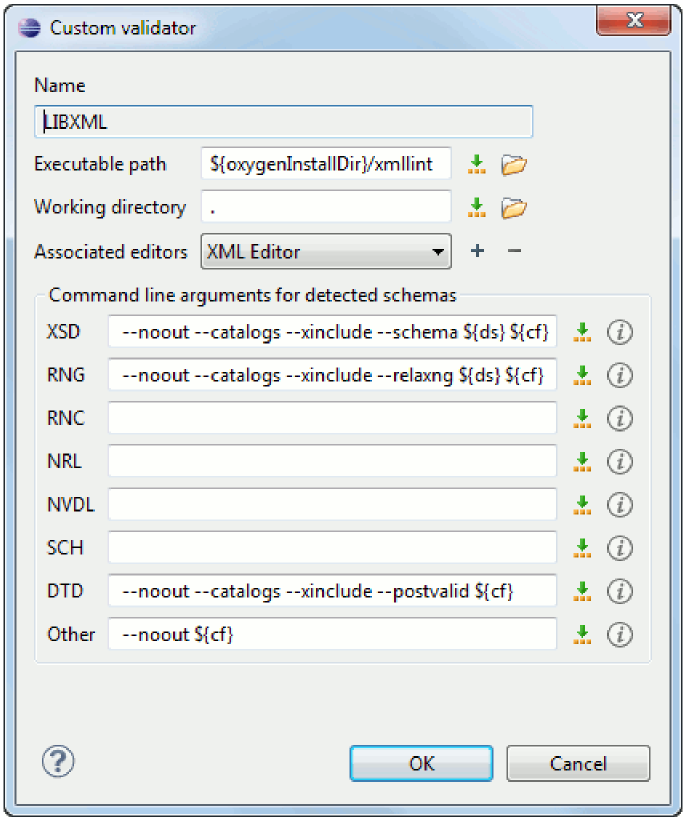
The following preferences can be set:

- **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 exiting one you can use the **Custom Validator** dialog displayed by pressing the **New** button or the **Edit** button.



Name: LIBXML

Executable path: ${oxygenInstallDir}/xmllint

Working directory: .

Associated editors: XML Editor

Command line arguments for detected schemas

XSD	--noout --catalogs --xinclude --schema ${ds} ${cf}
RNG	--noout --catalogs --xinclude --relaxng ${ds} ${cf}
RNC	
NRL	
NVDL	
SCH	
DTD	--noout --catalogs --xinclude --postvalid ${cf}
Other	--noout ${cf}

Figure 279: 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 Editor 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 Editor plugin recognizes (no validation is performed) browser-specific CSS properties. The content completion assistant lists these properties at the end of its list, prefixed with the following particles:
  - `-moz-` for Mozilla;
  - `-ms-` for Internet Explorer;
  - `-o-` for Opera;
  - `-webkit-` for Safari/Webkit.

XML

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 Editor 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 Editor plugin will use to resolve references at document validation and transformation will be a default built-in catalog. This catalog maps such references to the built-in local copies of the schemas of the Oxygen XML Editor plugin frameworks: DocBook, DITA, TEI, XHTML, SVG, etc.

You can also add or configure catalogs at framework level in the *Document Type Association* preferences page.

When you add, delete or edit an XML catalog to / from the list 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 Editor plugin is turned on for validation, rendering in Author mode and transformation of XML documents.
- **Base URI fix-up** - According to the specification for XInclude, processors must add an `xml:base` attribute to elements included from locations with a different base URI. Without these attributes, the resulting infoset information would be incorrect.

  Unfortunately, these attributes make XInclude processing not transparent to Schema validation. One solution to this is to modify your schema to allow `xml:base` attributes to appear on elements that might be included from different base URIs.

  If the addition of `xml:base` and / or `xml:lang` is undesired by your application, you can disable base URI fix-up.

- **Language fix-up** - The processor will preserve language information on a top-level included element by adding an `xml:lang` attribute if its include parent has a different [language] property. If the addition of `xml:lang` is undesired by your application, you can disable the language fix-up.
- **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 Editor 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:

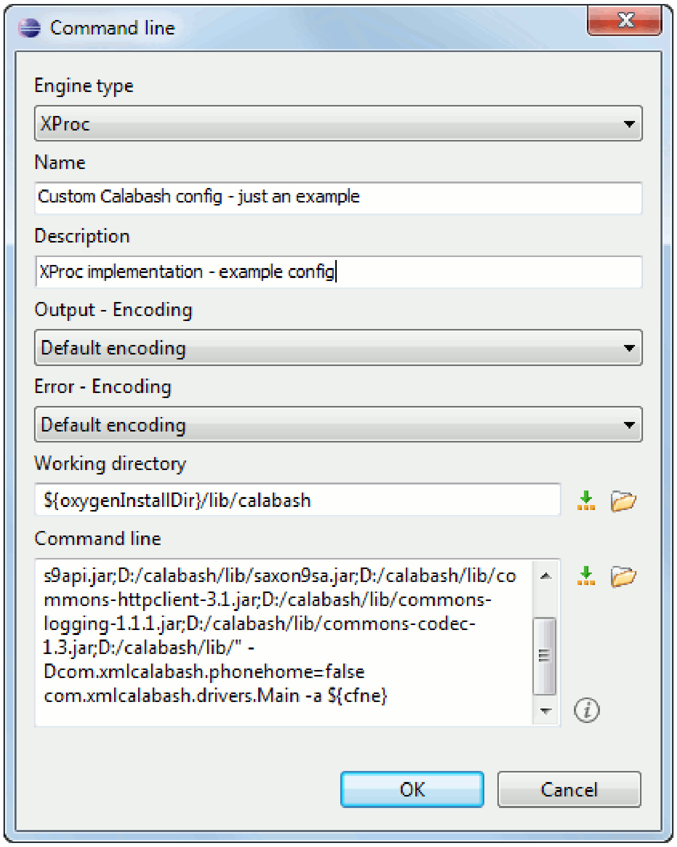
- **Generate optional elements** - If checked, the elements declared optional in the schema will be generated in the XML instance.
- **Generate optional attributes** - If checked, the attributes declared optional in the schema will be generated in the XML instance.
- **Values of elements and attributes** - 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 Editor plugin comes with a built-in XProc engine called *Calabash*. An external XProc engine can be configured in this panel.

When **Show XProc messages** is selected all messages emitted by the XProc processor during a transformation will be presented in the results view.

For an external engine the value of the **Name** field will be displayed in the XProc transformation scenario and in the command line that will start it.



Engine type: XProc

Name: Custom Calabash config - just an example

Description: XProc implementation - example config

Output - Encoding: Default encoding

Error - Encoding: Default encoding

Working directory: ${oxygenInstallDir}/lib/calabash

Command line:

```
s9api.jar;D:/calabash/lib/saxon9sa.jar;D:/calabash/lib/co
mmons-httpclient-3.1.jar;D:/calabash/lib/commons-
logging-1.1.1.jar;D:/calabash/lib/commons-codec-
1.3.jar;D:/calabash/lib/" -
Dcom.xmlcalabash.phonehome=false
com.xmlcalabash.drivers.Main -a ${cfne}
```

Figure 280: 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**.

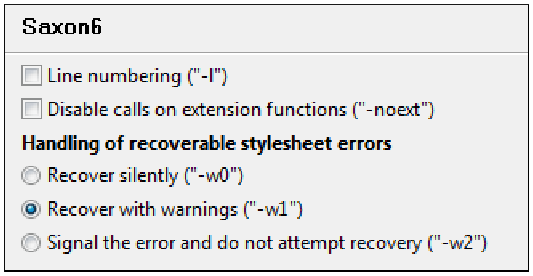
If you want to use an XSLT transformer implemented in Java different than the ones that ship with Oxygen XML Editor plugin namely Apache Xalan and Saxon all you have to do is to specify the name of the transformer's factory class which Oxygen XML Editor plugin will set as the value of the Java property `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 Editor plugin libraries folder (the `lib` subfolder of the Oxygen XML Editor 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**.



Saxon6

□ Line numbering ("-l")

□ Disable calls on extension functions ("-noext")

Handling of recoverable stylesheet errors

○ Recover silently ("-w0")

● Recover with warnings ("-w1")

○ Signal the error and do not attempt recovery ("-w2")

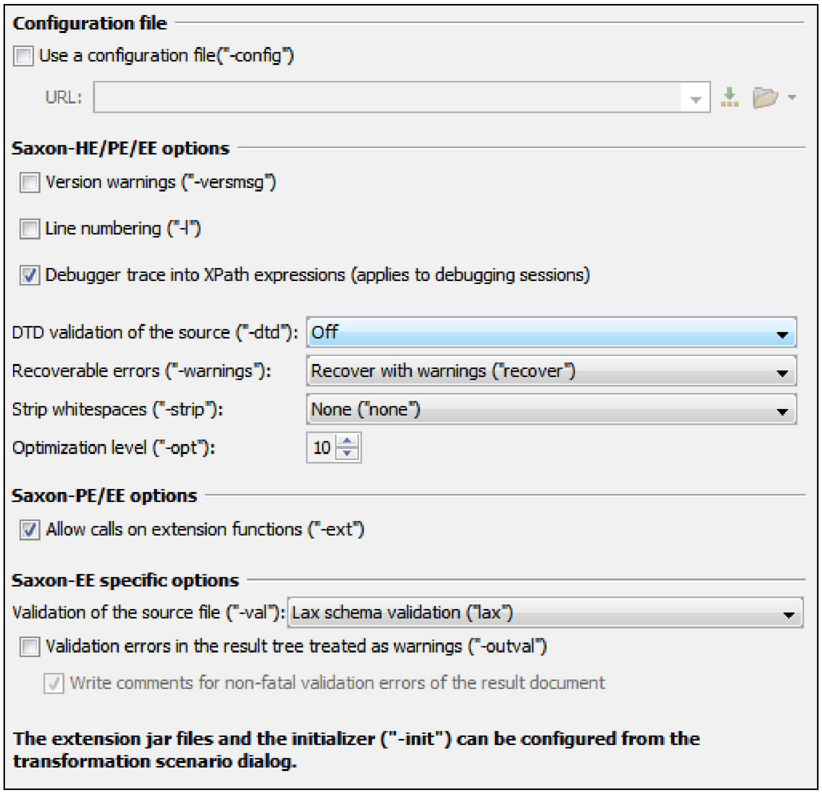
Figure 281: The Saxon 6 XSLT Preferences Panel

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**.



Configuration file

☐ Use a configuration file("-config")

URL:

Saxon-HE/PE/EE options

☐ Version warnings ("-versmsg")

☐ Line numbering ("-l")

☑ Debugger trace into XPath expressions (applies to debugging sessions)

DTD validation of the source ("-dtd"): Off

Recoverable errors ("-warnings"): Recover with warnings ("recover")

Strip whitespaces ("-strip"): None ("none")

Optimization level ("-opt"): 10

Saxon-PE/EE options

☑ Allow calls on extension functions ("-ext")

Saxon-EE specific options

Validation of the source file ("-val"): Lax schema validation ("lax")

☐ Validation errors in the result tree treated as warnings ("-outval")

☑ Write comments for non-fatal validation errors of the result document

The extension jar files and the initializer ("-init") can be configured from the transformation scenario dialog.

Figure 282: 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`.

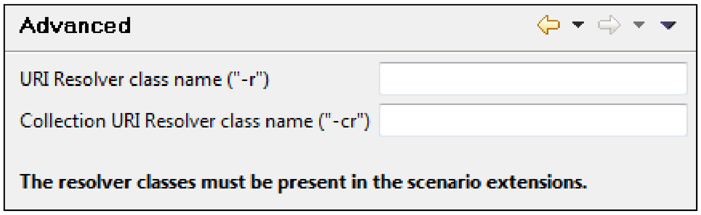
- **Optimization level ("-opt")** - Set optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization); currently all values other than 0 result in full optimization but this is likely to change in the future. The default is full optimization; this feature allows 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**.



Advanced

URI Resolver class name ("-r")

Collection URI Resolver class name ("-cr")

The resolver classes must be present in the scenario extensions.

Figure 283: 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 Editor plugin will display in the **Warnings** view the version of the **libxml** and **libxslt** libraries invoked by XSLTProc.
- **Show time information** - If checked, the **Warnings** view will display the time necessary for running the transformation.
- **Show debug information** - If checked, the **Warnings** view will display debug information about what templates are matched, parameter values, etc.
- **Show all documents loaded during processing** - If checked, Oxygen XML Editor plugin will display in the **Warnings** view the URL of all the files loaded during transformation.
- **Show profile information** - If checked, Oxygen XML Editor plugin will display in the **Warnings** view a table with all the matched templates, and for each template will display: the match XPath expression, the template name, the number of template modes, the number of calls, the execution time.
- **Show the list of registered extensions** - If checked, Oxygen XML Editor plugin will display in the **Warnings** view a list with all the registered extension functions, extension elements and extension modules.
- **Refuses to write to any file or resource** - If checked, the XSLTProc processor will not write any part of the transformation result to an external file on disk. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.
- **Refuses to create directories** - If checked, the XSLTProc processor will not create any directory during the transformation process. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.

MSXML

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

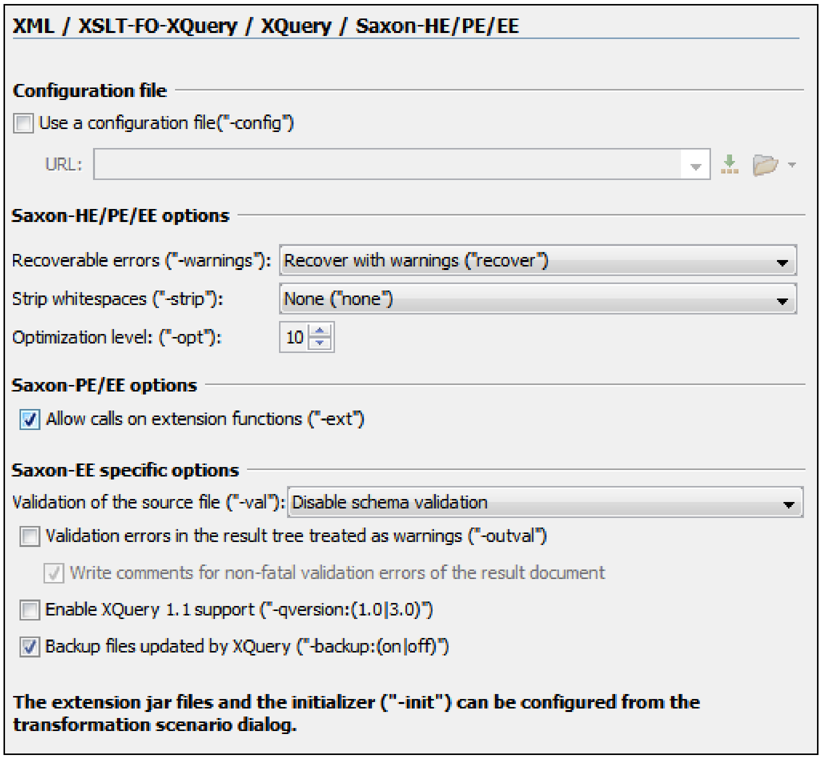
The **XQuery** preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **XML** > **XSLT/FO/XQuery** > **XQuery**.

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 Editor plugin uses the processor specified in the scenario. If no validation scenario is associated, but the file has an associated transformation scenario, the processor specified in the scenario will be used. If the processor does not support validation or if no scenario is associated, then the value from this combo box will be used as validation processor
- **Size limit of Sequence view (MB)** - When the result of an XQuery transformation is *set in the transformation scenario as sequence* the size of one chunk of the result that is fetched from the database in lazy mode in one step is set in this option. If this limit is exceed you can extract more data from the database by a click on the **More result available** node from the **Sequence** view.
- **Format transformer output** - When checked the transformer's output is formatted and indented (pretty printed). The option is ignored if in the transformation scenario you choose **Sequence** (lazy extract data from a database).
- **Create structure indicating the type nodes** - If checked, Oxygen XML Editor 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**.



XML / XSLT-FO-XQuery / XQuery / Saxon-HE/PE/EE

Configuration file

□ Use a configuration file("-config")

URL:

Saxon-HE/PE/EE options

Recoverable errors ("-warnings"): Recover with warnings ("recover")

Strip whitespaces ("-strip"): None ("none")

Optimization level: ("-opt"): 10

Saxon-PE/EE options

☑ Allow calls on extension functions ("-ext")

Saxon-EE specific options

Validation of the source file ("-val"): Disable schema validation

□ Validation errors in the result tree treated as warnings ("-outval")

☑ Write comments for non-fatal validation errors of the result document

□ Enable XQuery 1.1 support ("-qversion:(1.0|3.0)")

☑ Backup files updated by XQuery ("-backup:(on|off)")

The extension jar files and the initializer ("-init") can be configured from the transformation scenario dialog.

Figure 284: 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.

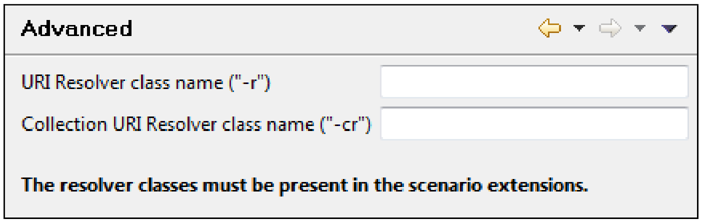
The Saxon 9.3.0.5 Enterprise Edition specific options are the following:

- **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**.



Advanced

URI Resolver class name ("-r")

Collection URI Resolver class name ("-cr")

The resolver classes must be present in the scenario extensions.

Figure 285: 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 Editor plugin preferences are available:

- **XSLT/XQuery Debug Current Instruction Pointer** - Controls the background color of the current execution node, both in the document (XML) and XSL/XQuery views.

Profiler

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 554).

The following profiles settings are available:

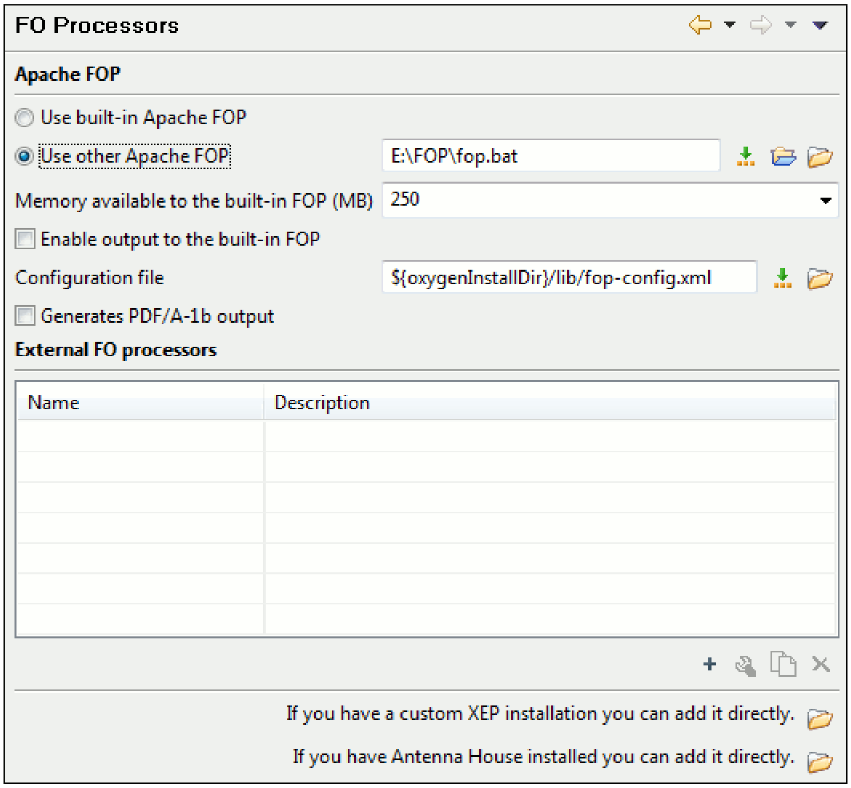
- **Show time** - Shows the total time that was spent in the node.
- **Show inherent time** - Shows the inherent time that was spent in the node. The inherent time is defined as the total time of a node minus the time of its child nodes.
- **Show invocation count** - Shows how many times the node was 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 Editor 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 Editor plugin uses the environmental variables set by the XSL Formatter installation to detect and use it for XSL-FO transformations. If the environmental variables are not set for the XSL Formatter installation, you can browse and choose the executable just as you would for XEP.

The **FO Processors** preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **XML** > **XSLT/FO/XQuery** > **FO Processors**.



FO Processors

Apache FOP

○ Use built-in Apache FOP

● Use other Apache FOP: E:\FOP\fop.bat

Memory available to the built-in FOP (MB): 250

□ Enable output to the built-in FOP

Configuration file: ${oxygenInstallDir}/lib/fop-config.xml

□ Generates PDF/A-1b output

External FO processors

Name	Description

If you have a custom XEP installation you can add it directly.

If you have Antenna House installed you can add it directly.

Figure 286: The FO Processors Preferences Panel

Apache FOP

The options for FO processors are the following:

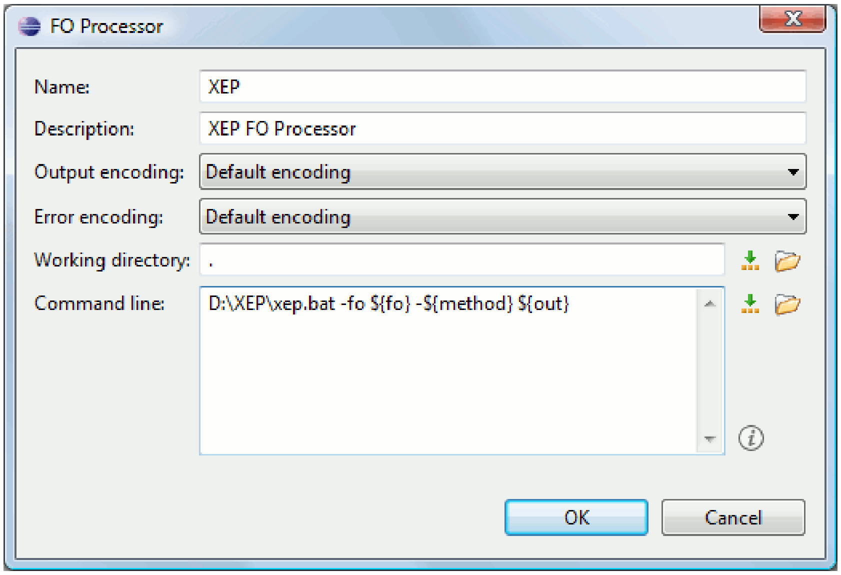
- **Use built-in Apache FOP** - Instructs Oxygen XML Editor plugin to use its built-in Apache FO processor.
- **Use other Apache FOP** - Instructs Oxygen XML Editor plugin to use another Apache FO processor installed on your computer.
- **Enable the output of the built-in FOP** - All Apache FOP output is displayed in a results pane at the bottom of the Oxygen XML Editor plugin window including warning messages about FO instructions not supported by Apache FOP.
- **Memory available to the built-in FOP** - If your Apache FOP transformations fail with an Out of Memory error (**OutOfMemoryError**) select from this combo box a larger value for the amount of memory reserved for FOP transformations.
- **Configuration file for the built-in FOP** - You should specify here the path to an Apache FOP configuration file, necessary for example to render to PDF a document containing Unicode content using a special *true type* font.
- **Generates PDF/A-1b output** - When selected PDF/A-1b output is generated.

  Note: All fonts have to be embedded, even the implicit ones. More information about configuring metrics files for the embedded fonts can be found in *Add a font to the built-in FOP*.

  Note: You cannot use the `<filterList>` key in the configuration file because FOP would generate the following error: *The Filter key is prohibited when PDF/A-1 is active*.

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:



FO Processor

Name: XEP

Description: XEP FO Processor

Output encoding: Default encoding

Error encoding: Default encoding

Working directory: .

Command line: D:\XEP\xep.bat -fo ${fo} -${method} ${out}

Figure 287: The External FO Processor Configuration Dialog

- **Name** - The name displayed in the list of available FOP processors on the FOP tab of the transformation scenario dialog.
- **Description** - A textual description of the FO processor displayed in the FO processors table and in tooltips of UI components where the processor is selected.
- **Output Encoding** - The encoding of the FO processor output stream displayed in a results panel at the bottom of the Oxygen XML Editor plugin window.
- **Error Encoding** - The encoding of the FO processor error stream displayed in a results panel at the bottom of the Oxygen XML Editor plugin window.

- **Working directory** - The directory where the intermediate and final results of the processing is stored. Here you can use one of the following editor variables:
  - **${homeDir}** - The path to user home directory.
  - **${cfd}** - The path of current file directory. If the current file is not a local file, the target is the user's desktop directory.
  - **${pd}** - The project directory.
  - **${oxygenInstallDir}** - The Oxygen XML Editor plugin installation directory.
- **Command line** - The command line that starts the FO processor, specific to each processor. Here you can use one of the following editor variables:
  - **${method}** - The FOP transformation method: **pdf**, **ps** or **txt**.
  - **${fo}** - The input FO file.
  - **${out}** - The output file.
  - **${pd}** - The project directory.
  - **${frameworksDir}** - The path of the `frameworks` subdirectory of the Oxygen XML Editor plugin install directory.
  - **${oxygenInstallDir}** - The Oxygen XML Editor plugin installation directory.
  - **${ps}** - The platform-specific path separator. It is used between the library files specified in the class path of the command line.

XPath

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:

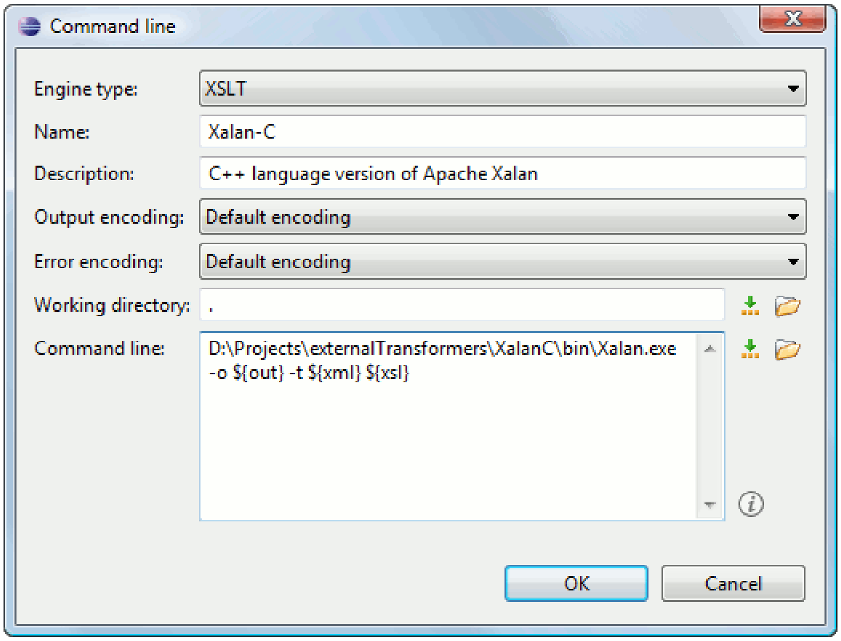
  ```
  //varlistentry[starts-with(@os,'s')]
  ```
- **No namespace** - If checked, Oxygen XML Editor plugin will consider unprefixed element names in XPath 2.0 expressions evaluated in *the XPath console* as belonging to no namespace.
- **Use the default namespace from the root element** - If checked, Oxygen XML Editor 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 Editor 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 Editor 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*. However it cannot be used in *the Debugger perspective*.

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:



Engine type: XSLT

Name: Xalan-C

Description: C++ language version of Apache Xalan

Output encoding: Default encoding

Error encoding: Default encoding

Working directory: .

Command line:

```
D:\Projects\externalTransformers\XalanC\bin\Xalan.exe
-o ${out} -t ${xml} ${xsl}
```

Figure 288: Parameters of a Custom Engine

- **Engine type** - Combo box allowing you to choose the transformer type. There are two options: XSLT engines and XQuery engines.
- **Name** - The name of the transformer displayed in the dialog for editing transformation scenarios
- **Description** - A textual description of the transformer.
- **Output Encoding** - The encoding of the transformer output stream.
- **Error Encoding** - The encoding of the transformer error stream.
- **Working directory** - The start directory of the transformer executable program. The following editor variables are available for making the path to the working directory independent of the location of the input files:
  - **${homeDir}** - The user home directory in the operating system.
  - **${cfd}** - The path to the directory of the current file.
  - **${pd}** - The path to the directory of the current project.
  - **${oxygenInstallDir}** - The Oxygen XML Editor plugin install directory.
- **Command line** - The command line that must be executed by Oxygen XML Editor plugin to perform a transformation with the engine. The following editor variables are available for making the parameters in the command line (the transformer executable, the input files) independent of the location of the input files:
  - **${xml}** - The XML input document as a file path.
  - **${xmlu}** - The XML input document as a URL.
  - **${xsl}** - The XSL / XQuery input document as a file path.
  - **${xslu}** - The XSL / XQuery input document as a URL.
  - **${out}** - The output document as a file path.
  - **${outu}** - The output document as a URL.
  - **${ps}** - The platform separator which is used between library file names specified in the class path.

Import

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 Editor plugin to escape the imported content to an XML-safe form.
- **Add annotations for generated XML Schema** - If checked, the generated XML Schema contains an annotation for each of the imported table columns. The documentation inside the annotation tag contains the remarks of the database columns (if available) and also information about the conversion between the column type and the generated XML Schema type.

The section **Date / Time Format** specifies the format used for importing date and time values from Excel spreadsheets or database tables and in the generated XML schemas. The following format types are available:

- **Unformatted** - If checked, the date and time formats specific to the database are used for import. When importing data from Excel a string representation of date or time values are used. The type used in the generated XML Schema is `xs:string`.
- **XML Schema date format** - If checked, the XML Schema-specific format ISO8601 is used for imported date / time data (`yyyy-MM-dd'T'HH:mm:ss` for `datetime`, `yyyy-MM-dd` for `date` and `HH:mm:ss` for `time`). The types used in the generated XML Schema are `xs:datetime`, `xs:date` and `xs:time`.
- **Custom format** - If checked, the user can define a custom format for timestamp, date, and time values or choose one of the predefined formats. A preview of the values is presented when a format is used. The type used in the generated XML Schema is `xs:string`.

Date / Time Patterns

Table 10: Pattern letters

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am / pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am / pm (0-11)	Number	0

Letter	Date or Time Component	Presentation	Examples
h	Hour in am / pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

Pattern letters are usually repeated, as their number determines the exact presentation:

- *Text* - If the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available.
- *Number* - The number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount.
- *Year* - If the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a number.
- *Month* - If the number of pattern letters is 3 or more, the month is interpreted as text; otherwise, it is interpreted as a number.
- *General time zone* - Time zones are interpreted as text if they have names. For time zones representing a GMT offset value, the following syntax is used:

  - *GMTOffsetTimeZone* - GMT Sign Hours : Minutes
  - *Sign* - one of + or -
  - *Hours* - one or two digits
  - *Minutes* - two digits
  - *Digit* - one of 0 1 2 3 4 5 6 7 8 9

  Hours must be between 0 and 23, and Minutes must be between 00 and 59. The format is locale independent and digits must be taken from the Basic Latin block of the Unicode standard.

- *RFC 822 time zone*: The RFC 822 4-digit time zone format is used:

  - *RFC822TimeZone* - Sign *TwoDigitHours* Minutes
  - *TwoDigitHours* - a number of two digits

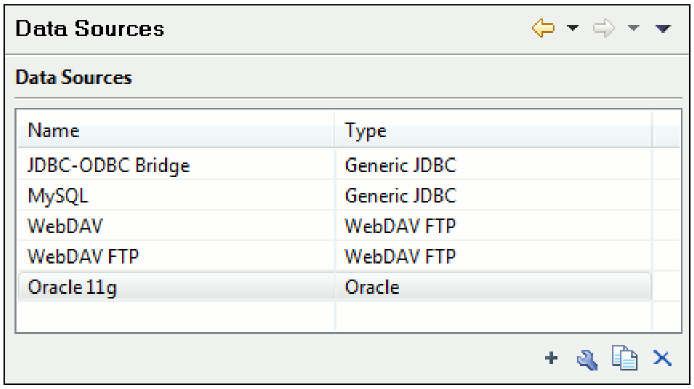
  TwoDigitHours must be between 00 and 23.

Data Sources

The **Data Sources** preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **Data Sources**.

Configuration of Data Sources

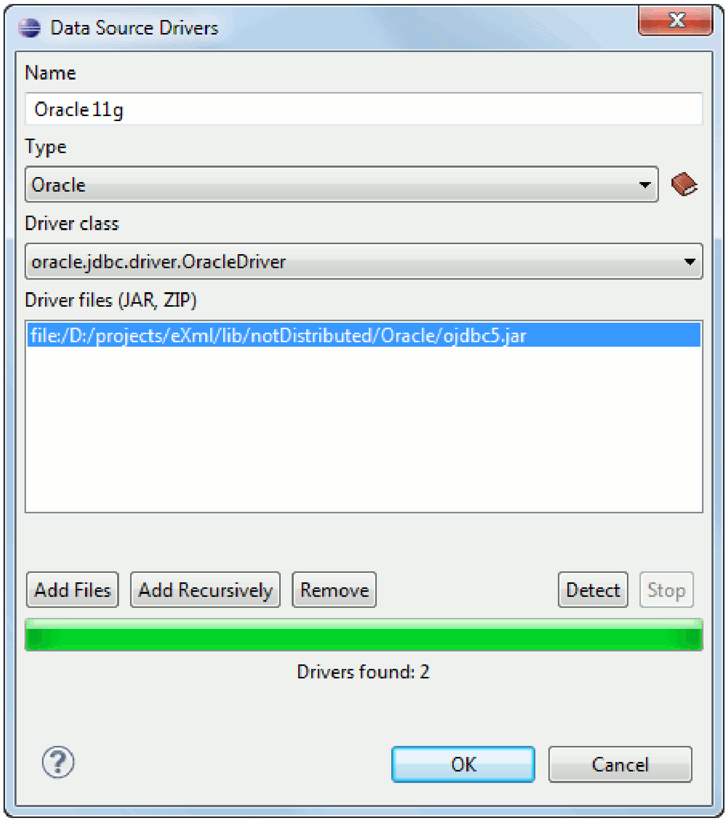
Here you can configure data sources and connections to relational databases as well as native XML databases. You can check the list of drivers (*http://www.oxygenxml.com/database_drivers.html*) available for the major database servers.



Name	Type
JDBC-ODBC Bridge	Generic JDBC
MySQL	Generic JDBC
WebDAV	WebDAV FTP
WebDAV FTP	WebDAV FTP
Oracle 11g	Oracle

Figure 289: The Data Sources Preferences Panel

- **New** - Opens the **Data Sources Drivers** dialog that allows you to configure a new database driver.



Name: Oracle 11g

Type: Oracle

Driver class: oracle.jdbc.driver.OracleDriver

Driver files (JAR, ZIP): file:/D:/projects/eXml/lib/notDistributed/Oracle/ojdbc5.jar

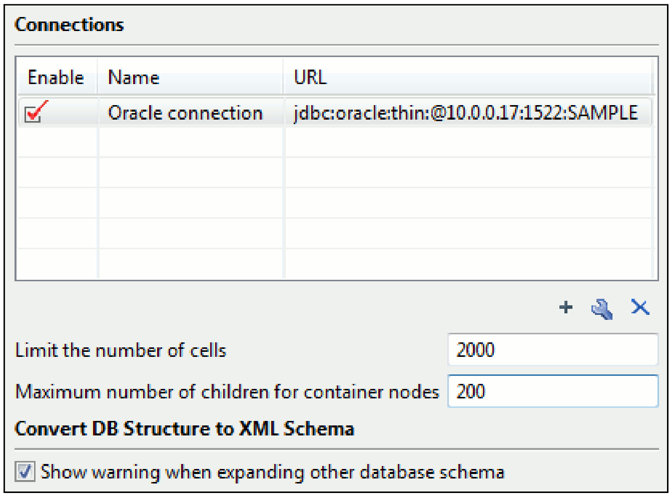
Drivers found: 2

Figure 290: 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.



Connections

Enable	Name	URL
✓	Oracle connection	jdbc:oracle:thin:@10.0.0.17:1522:SAMPLE

Limit the number of cells	2000
Maximum number of children for container nodes	200

Convert DB Structure to XML Schema

☑ Show warning when expanding other database schema

Figure 291: The Connections Preferences Panel

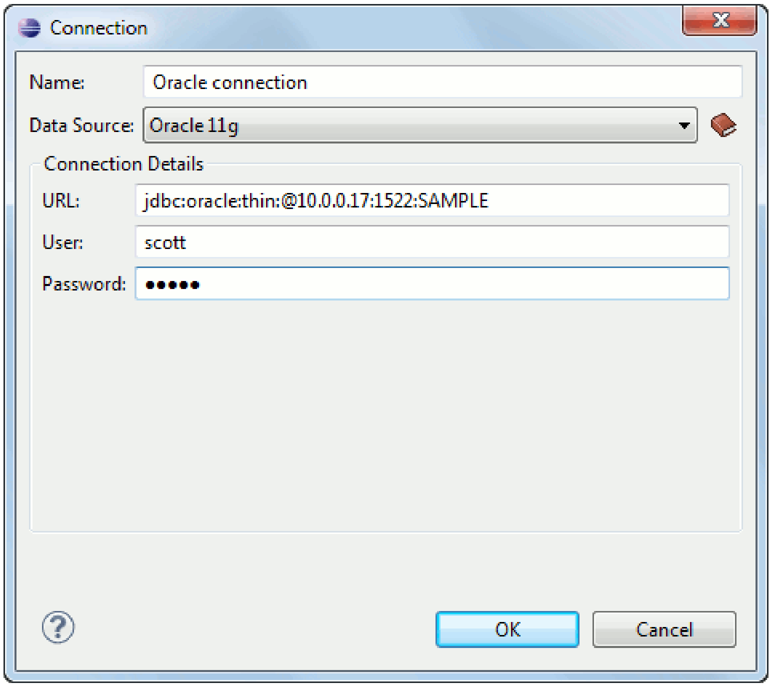
For performance issues, you can set the maximum number of cells that will be displayed in the **Table Explorer** view for a database table. Leave the field **Limit the number of cells** empty if you want the entire content of the table to be displayed. By default this field is set to 2,000. If a table 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:



Name: Oracle connection

Data Source: Oracle 11g

URL: jdbc:oracle:thin:@10.0.0.17:1522:SAMPLE

User: scott

Password: ●●●●●

Figure 292: 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 Editor plugin .

- **Berkeley DB XML database** - Copy the jar files from the Berkeley database install directory to the Oxygen XML Editor plugin install directory as described in *the procedure* for configuring a Berkeley DB data source.
- **IBM DB2 Pure XML database** - Go to the *IBM website* and in the *DB2 Clients and Development Tools* category select the *DB2 Driver for JDBC and SQLJ* download link. Fill the download form and download the zip file. Unzip the zip file and use the `db2jcc.jar` and `db2jcc_license_cu.jar` files in Oxygen XML Editor plugin for *configuring a DB2 data source*.

- **eXist database** - Copy the jar files from the eXist database install directory to the Oxygen XML Editor plugin install directory as described in *the procedure* for configuring an eXist data source.
- **MarkLogic database** - Download the 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 Editor plugin with a Java 1.6 virtual machine in order to use this driver.
- **Oracle 11g database** - Download the Oracle 11g JDBC driver called `ojdbc5.jar` from the *Oracle website* and use it for *configuring an Oracle data source*.
- **PostgreSQL 8.3 database** - Download the PostgreSQL 8.3 JDBC driver called `postgresql-8.3-603.jdbc3.jar` from the *PostgreSQL website* and use it for *configuring a PostgreSQL data source*.
- **RainingData TigerLogic XDMS database** - Copy the jar files from the TigerLogic JDK `lib` directory from the server side to the Oxygen XML Editor 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 Editor 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 Editor 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.

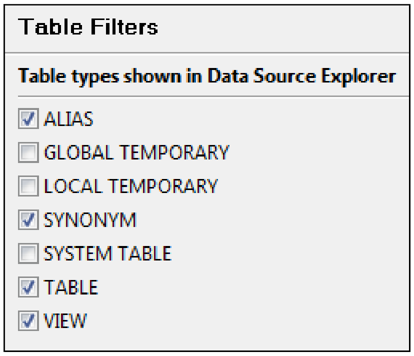


Table Filters

Table types shown in Data Source Explorer

☑ ALIAS
□ GLOBAL TEMPORARY
□ LOCAL TEMPORARY
☑ SYNONYM
□ SYSTEM TABLE
☑ TABLE
☑ VIEW

Figure 293: Table Filters Preferences Panel

Archive

The **Archive** preferences panel is opened from menu **Window** > **Preferences** > **oXygen** > **Archive**.

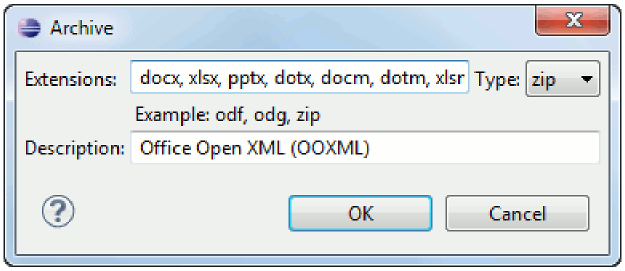
The following options are available in the **Archive** preferences panel:

- **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 Editor plugin . You can edit an existing mapping or create a new one by associating your own list of extensions to an archive format.



Extensions: docx, xlsx, pptx, dotx, docm, dotm, xlsr Type: zip

Example: odf, odg, zip

Description: Office Open XML (OOXML)

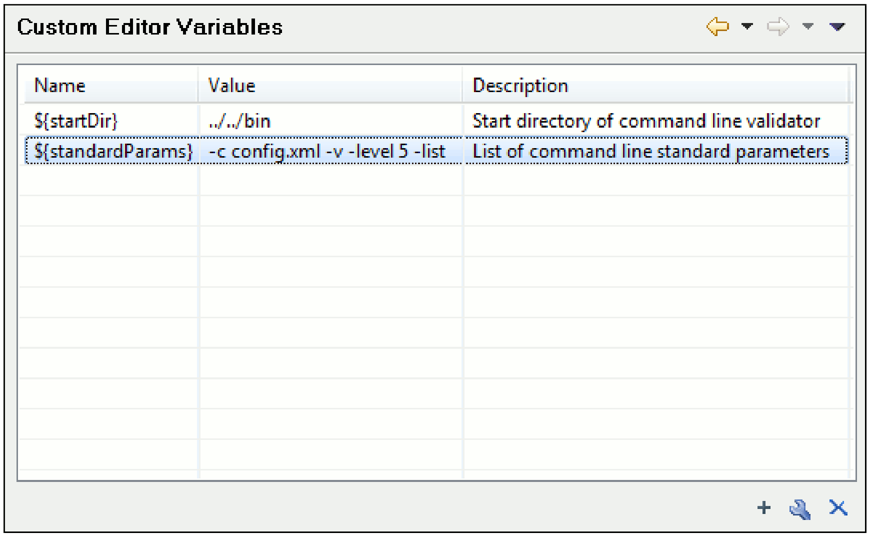
Figure 294: Edit Archive Extension Mappings

Important: You have to restart Oxygen XML Editor plugin after removing an extension from the table in order for that extension to not be recognised anymore as an archive extension.

- **Store Unicode file names in Zip archives** - Use this option when you archive files that contain international (that is, non-English) characters in file names or file comments. If an archive is modified in any way with this option turned on, UTF-8 characters are used in the names of all files in the archive.

Custom Editor Variables

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
${startDir}	../../bin	Start directory of command line validator
${standardParams}	-c config.xml -v -level 5 -list	List of command line standard parameters

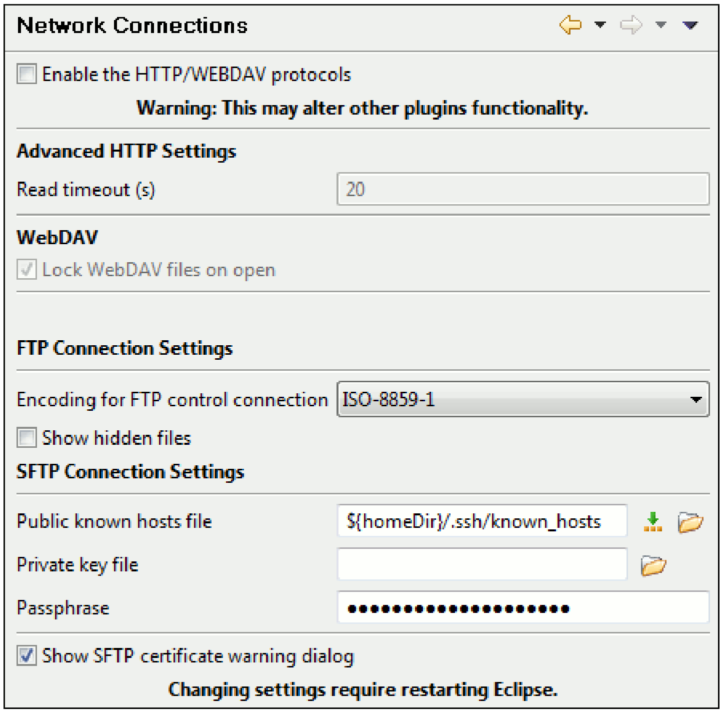
Figure 295: Custom Editor Variables

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



Network Connections

□ Enable the HTTP/WEBDAV protocols

Warning: This may alter other plugins functionality.

Advanced HTTP Settings

Read timeout (s): 20

WebDAV

☑ Lock WebDAV files on open

FTP Connection Settings

Encoding for FTP control connection: ISO-8859-1

□ Show hidden files

SFTP Connection Settings

Public known hosts file: ${homeDir}/.ssh/known_hosts

Private key file:

Passphrase: ••••••••••••••••••••

☑ Show SFTP certificate warning dialog

Changing settings require restarting Eclipse.

Complete the dialog as follows:

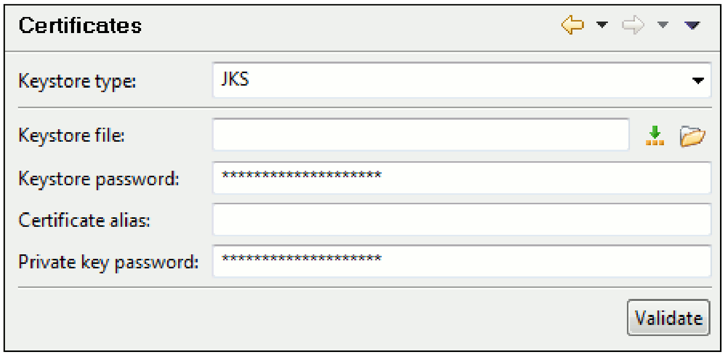
- **Enable the HTTP/WEBDAV protocols** - If checked, the HTTP(S) connections go through the proxy with the host, port, user name and password that are set in Eclipse's built-in general *Network Connections* preferences panel.

  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 Editor 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 Editor 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 Editor plugin in the **Certificates** preferences panel which is opened from menu **Window** > **Preferences** > **oXygen** > **Certificates**. The parameters of a keystore are the following:



Certificates

Keystore type: JKS

Keystore file:

Keystore password: ********************

Certificate alias:

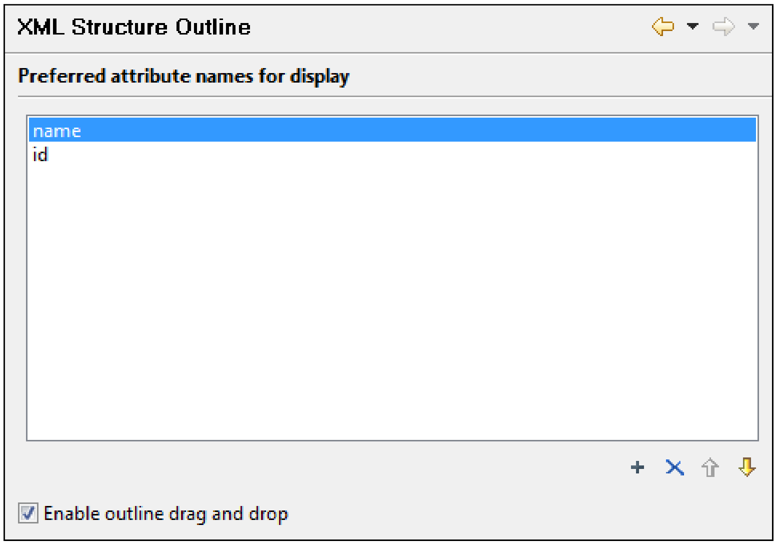
Private key password: ********************

Figure 296: 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:



XML Structure Outline

Preferred attribute names for display

name
id

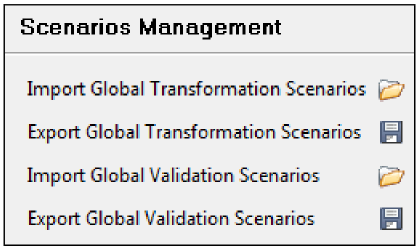
☑ Enable outline drag and drop

Figure 297: 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.



Scenarios Management

Import Global Transformation Scenarios

Export Global Transformation Scenarios

Import Global Validation Scenarios

Export Global Validation Scenarios

Figure 298: 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.

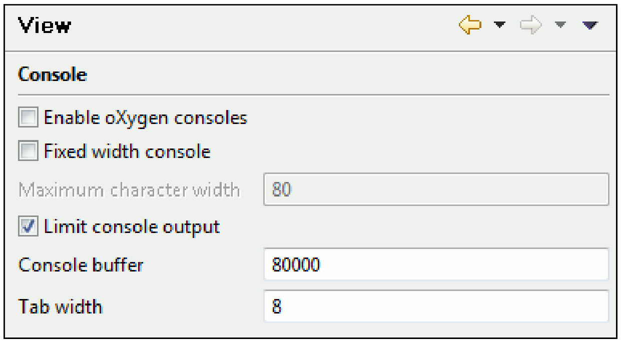
- **Export Global Transformation Scenarios** - Allows you to export all global transformation scenarios available in the **Configure Transformation Scenario** dialog.
- **Import Global Validation Scenarios** - Allows you to import at global level all scenarios from a properties file created with the 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:



View

Console

☐ Enable oXygen consoles

☐ Fixed width console

Maximum character width 80

☑ Limit console output

Console buffer 80000

Tab width 8

Figure 299: 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 Editor plugin commands of external engines or other external tools, in transformation scenarios, validation scenarios and Author operations:

- **${oxygenHome}** - Oxygen XML Editor plugin installation folder as URL.
- **${oxygenInstallDir}** - Oxygen XML Editor plugin installation folder as file path.
- **${frameworks}** - The path (as URL) of the `frameworks` subfolder of the Oxygen XML Editor plugin install folder.
- **${frameworksDir}** - The path (as file path) of the `frameworks` subfolder of the Oxygen XML Editor plugin installation folder.
- **${home}** - The path (as URL) of the user home folder.
- **${homeDir}** - The path (as file path) of the user home folder.
- **${pdu}** - Current project folder as URL.
- **${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 *code template* , in Author operations, or in a selection plugin.
- **${selection}** - The XML content of the current selection in the editor panel. This variable can be used in a *code template* and Author operations,
- **${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 Editor plugin checks that the URL is valid before passing it to the transformation.
    - `password` - input characters are hidden.
    - `generic` - the input is treated as generic text that requires no special handling.
  - `'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 *configured in Preferences.*

Chapter 29

Common Problems

Topics:

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 Editor plugin in Eclipse. Why does an `.ext` file opened with the Oxygen XML Editor plugin plugin not have syntax highlight?

Associating an extension with Oxygen XML Editor plugin in Eclipse 3.7+ requires three steps:

1. Associate the `.ext` extension with the Oxygen XML Editor plugin plugin.
   a) Go to menu **Windows** > **Preferences** > **General** > **Editors** > **File Associations**.
   b) Add `*.ext` to the list of file types.
   c) Select `*.ext` in the list by clicking on it.
   d) Add Oxygen XML Editor plugin to the list of **Associated editors** and make it the default editor.
2. Associate the `.ext` extension with the **Oxygen XML** content type.
   a) Go to menu **Windows** > **Preferences** > **General** > **Content Types**.
   b) Add `*.ext` to the **File associations** list for the **Text** > **XML** > **oXygen XML** content type.
3. Press the **OK** button in the Eclipse preferences dialog.

Now when an `*.ext` file is opened the icon of the editor and the syntax highlight should be the same as for XML files opened with the Oxygen XML Editor plugin 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 30

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