Oxygen XML Author 16.1

Notice

Copyright

Oxygen XML Author User Manual

Syncro Soft SRL.

Copyright © 2002-2014 Syncro Soft SRL. All Rights Reserved.

Contents

Chapter 19: Configuring Oxygen XML Author..725

Chapter 20: Performance Problems...813

Chapter 21: Common Problems..815

Chapter

1

Introduction

Topics:

- *Key Features and Benefits of Oxygen XML Author*

Welcome to the User Manual of Oxygen XML Author 16.1!

Oxygen XML Author is a cross-platform application designed for authors who want to edit XML documents visually without extensive knowledge about XML and XML related technologies. The WYSIWYG-like editor is driven by CSS stylesheets associated with the XML documents and offers the option to switch off XML tags completely when editing an XML document.

This user guide is focused mainly at describing features, functionality and application interface to help you get started in no time.

Key Features and Benefits of Oxygen XML Author

Multiplatform availability: Windows, OS X, Linux, Solaris	Multilanguage support: English, German, French, Italian and Japanese
Visual WYSIWYG XML editing mode based on W3C CSS stylesheets.	Visual DITA Map editor
Closely integrate with the DITA Open Toolkit for generating DITA output	Support for latest versions of document frameworks: DocBook and TEI.
Can be used as standalone desktop application, run through Java Web Start or as an Eclipse plugin	Non blocking operations, you can perform validation and transformation operations in background
Support for XML, XML Schema 1.0 and 1.1, Relax NG , Schematron, DTD, NVDL schemas, XSLT, XSL:FO, WSDL, XQuery, HTML, CSS	Support for XML, CSS, XSLT, XSL-FO.
Multiple built-in validation engines (Xerces, libxml, Saxon SA, MSXML 4.0, MSXML.NET) and support for custom validation engines (XSV, SQC).	Multiple built-in XSLT transformers (Saxon 6.5, Saxon 9 Enterprise (schema aware), Xalan, libxslt, MSXML 3.0 / 4.0, Microsoft .NET 1.0, Microsoft .NET 2.0), support for custom JAXP transformers.
Support for latest versions of document frameworks: DocBook and TEI.	Compare and merge files and directories
Ready to use FOP support to generate PDF or PS documents	XInclude support
Support for editing remote files over FTP, SFTP, HTTP / WebDAV and HTTPS / WebDAV	Easy error tracking - locate the error source by clicking on it
New XML document wizards to easily create documents specifying a schema or a DTD	Context sensitive content assistant driven by XML Schema, Relax NG, DTD, NVDL or by the edited document structure enhanced with schema annotation presenter
XML Catalog support	Unicode support
Pretty-printing of XML files	Easy configuration for external FO Processors
Apply XSLT and FOP transformations	XPath search and evaluation support
Preview transformation results as XHTML or XML or in your browser	Support for document templates to easily create and share documents
Canonicalize and sign documents	XML project manager
Batch validate selected files in project	Fully-fledged client for the Apache Subversion™ (SVN) versioning system with support for SVN 1.7 and SVN 1.8.
Configurable external tools	Configurable actions key bindings
Multi-line find and replace support allows regular expressions, is XML aware, is incremental, handles multiple files	Special viewer for very large files (up to 2 GB file size).
Associate extensions on Windows	Bookmark support
OS X ready	Print documents
Model View	Attributes View
Multi-document environment	SVG Viewer
XSLT 2.0 and XSLT 3.0 full support	XPath 2.0 and XPath 3.0 execution and debugging support

Dockable views and editors	Document folding
Spell checking supporting English, German and French including locals	Custom protocol plugin support
All the usual editor capabilities (cut, copy, paste, find, replace, windows management)	Drag&drop support
Support for editing, modifying and using files directly from ZIP-type archives	Outline view in sync with a non well-formed document

Chapter

2

Installation

Topics:

The platform requirements and installation instructions are presented in this chapter.

Installation Options for Oxygen XML Author

Choosing how Oxygen XML Author runs

You can install Oxygen XML Author to run in a number of ways:

- As a desktop application on *Windows*, *Linux*, or *Mac*.

- As a desktop application on a *Unix or Linux server* or on *Windows Terminal Server*.
- From within a browser though the *Java Web Start* technology.

Choosing an installer

You have a choice of installers;

- The native installer for your platform. On Windows and Linux, the native installer can run also in unattended mode.

- The All-platforms installer, which can be used on any supported platform.

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

Choosing a license option

You must *obtain and register a license* key to run Oxygen XML Author.

You can choose from two kinds of license:

- A named-person license, which can be used by a single person on multiple computers.
- A floating license, which can be used by different people at different times. Only one person can use a floating license at a time.

Upgrading, transferring, and uninstalling.

You can also *upgrade* Oxygen XML Author, *transfer a license*, or *uninstall* Oxygen XML Author.

Getting help with installation

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

Install Oxygen XML Author on Windows

Choosing an installer

You can install Oxygen XML Author on Windows using one of the following methods:

- Install using the *Windows installer*.
- Install using the *Windows installer in unattended mode*.
- Install using the *All Platforms installer*. Choose the all platforms installer if you have trouble installing using the Windows installer.

System Requirements

System requirements for a Windows install:

Operating systems

Windows XP, Windows Vista, Windows 7, Windows 8, Windows Server 2003, Windows Server 2008, Windows Server 2012

CPU

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

Memory

- Minimum - 1 GB of RAM
- Recommended - 4 GB of RAM

Storage

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

Java

Oxygen XML Author requires Java. If you use the native Windows installer, Oxygen XML Author will be installed with its own copy of Java. If you use the all platforms installer, your system must have a compatible Java virtual machine installed.

Oxygen XML Author supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.7) from Oracle available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*. Oxygen XML Author may work with JVM implementations from other vendors, but there is no guarantee that those implementations will work with future Oxygen XML Author updates and releases.

Oxygen XML Author uses the following rules to determine which installed version of Java to use:

1. If you install using the native Windows installer, which installs a version of Java as part of the Oxygen XML Author installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Windows environment variable `JAVA_HOME` is set, Oxygen XML Author uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your PATH environment variable is used.

If you run Oxygen XML Author using the batch file, `oxygenAuthor.bat`, you can edit the batch file to specify a particular version to use.

Install using the Windows installer

To install Oxygen XML Author using the Windows installer:

1. Make sure that your system meets the *system requirements*.
2. Download the Windows installer.
3. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
4. Run the installer and follow the instructions in the installation program.
5. Start Oxygen XML Author using one of the following methods:

   - Using one of the shortcuts created by the installer.
   - By running `oxygenAuthor.bat`, which is located in the install folder.

6. To license your copy of Oxygen XML Author go to **Help** > **Register...** and enter your *license information*.

Unattended Installation

You can run the installation in unattended mode by running the installer from the command line with the -q parameter. By default, running the installer in unattended mode installs Oxygen XML Author with the default options and does not overwrite existing files. You can change many options for the unattended installer using the *installer command line parameters*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`oxygenAuthor.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
   Oxygen XML Author is now installed in a new sub-folder called `oxygenAuthor`.
3. If you wish, you can move the directory where you installed Oxygen XML Author to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `oxygenAuthor16.1`.
4. Start Oxygen XML Author by running `oxygenAuthor.bat`, which is located in the install directory.
5. To license your copy of Oxygen XML Author go to **Help** > **Register...** and enter your *license information*.

Install Oxygen XML Author on Mac OS X

Choosing an installer

You can install Oxygen XML Author on Mac OS X using one of the following methods:

- Install using the Mac OS X installer, oxygenAuthor.zip.
- Install using the all platforms installer. Choose the all platforms installer if you have trouble installing using the Mac OS X installer.

System Requirements

System requirements for a Mac OS X install:

Operating system

Mac OS X version 10.5 64-bit or later

CPU

- Minimum - Intel-based Mac, 1 *GHz*
- Recommended - Dual Core class processor

Memory

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

Storage

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

Java

Oxygen XML Author requires Java to run. OS X includes Java by default or it will install it on the first attempt to run a Java application.

Oxygen XML Author supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.6.0 from Apple). Oxygen XML Author may work with JVM implementations from other vendors, but there is no guarantee that other implementations will work with future Oxygen XML Author updates and releases.

Oxygen XML Author uses the following rules to determine which installed version of Java to use:

1. If you start oXygen with the application launcher (.app) file then:

   a. if you use the zip distribution for OS X Oxygen XML Author uses the Apple Java SE 6 available on your Mac computer

    b. if you use the tar.gz distribution that contains a bundled JRE then Oxygen XML Author will use that bundled JRE

2. If you start Oxygen XML Author using a startup .sh script then:

    a. if a bundled JRE is available then it will be used

    b. otherwise, if the JAVA_HOME environment variable is set then the Java distribution indicated by it will be used

    c. otherwise the version of Java pointed to by your PATH environment variable will be used

If you run Oxygen XML Author using the oxygenAuthor.sh script, you can change the version of Java used by editing to script file. Go to the Java command at the end of the script file and specify the full path to the Java executable of the desired JVM version, for example:

```
/System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java "-Xdock:name= ...
```

OS X Installation

To install Oxygen XML Author on OS X:

1. Download the OS X Installation package (oxygenAuthor.zip).

The Safari web browser should recognize and expand the compressed file. If it is not automatically expanded, you can expand it manually by double clicking it.

2. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.

3. In **Finder**, move the expanded folder to your Applications folder.
Oxygen XML Author is now installed.

4. Start Oxygen XML Author, using one of the following methods:

- Double click Oxygen XML Author.app.

- Run sh oxygenAuthorMac.sh on the command line.

    ⚠️ **Notice:** You can start more than one instance on the same computer by running the following command for each new instance:

```
open -n OxygenAuthor.app
```

5. To license your copy of Oxygen XML Author, go to **Help** > **Register...** to enter your *license key*.

    📄 **Note:**

To avoid an unpredictable behavior of the cursor, use a fixed-size font, for example Monaco. You can find more details about this behaviour *here*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (oxygenAuthor.tar.gz) to a folder of your choice.

2. Extract the archive in that folder.
Oxygen XML Author is now installed in a new sub-folder called oxygenAuthor.

3. If you wish, you can move the directory where you installed Oxygen XML Author to your applications directory. You can also rename it to contain the product version information. For example you can rename it as oxygenAuthor16.1.

4. Start Oxygen XML Author by running oxygenAuthorMac.sh, which is located in the install folder.

5. To license your copy of Oxygen XML Author go to **Help** > **Register...** and enter your *license information*.

Install Oxygen XML Author on Linux

Choosing an installer

You can install Oxygen XML Author on Linux using any of the following methods:

- Install using the Linux installer.
- Install using the Linux installer in unattended mode.
- Install using the all platforms installer. Choose the all platforms installer if you have trouble installing using the Linux installer.

System Requirements

System requirements for a Linux install:

Operating system

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

CPU

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

Memory

- Minimum - 1 GB of RAM
- Recommended - 4 GB of RAM

Storage

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

Java

Oxygen XML Author requires Java. Oxygen XML Author supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.6.0) from Oracle available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*. Oxygen XML Author may work with JVM implementations from other vendors, but there is no guarantee that other implementations will work with future Oxygen XML Author updates and releases. Oxygen XML Author does not work with the GNU libgcj Java Virtual Machine.

Oxygen XML Author uses the following rules to determine which installed version of Java to use:

1. If you used the Linux installer, which installs a version of Java as part of the Oxygen XML Author installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Linux environment variable `JAVA_HOME` is set, Oxygen XML Author uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your PATH environment variable is used.

You can also change the version of the Java Virtual Machine that runs Oxygen XML Author by editing the script file, `oxygenAuthor.sh`. Go to the Java command at the end of the script file and specify the full path to the Java executable of the desired JVM version, for example:

```
/usr/bin/jre1.6.0_45/bin/java -Xmx256m ...
```

Linux Installation

Linux installation procedure.

To install Oxygen XML Author on Linux:

1. Download the Linux installer.
2. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
3. Run the installer that you downloaded and follow the instructions presented in the installation program.
4. Start Oxygen XML Author using one of the following methods:

   - Use the `author` shortcut created by the installer.
   - Run `sh oxygenAuthor.sh` from the command line. This file is located in the installation folder.

5. To license your copy of Oxygen XML Author go to **Help** > **Register...** and enter your *license key*.

Unattended Installation

You can run the installation in unattended mode by running the installer from the command line with the -q parameter. By default, running the installer in unattended mode installs Oxygen XML Author with the default options and does not overwrite existing files. You can change many options for the unattended installer using the *installer command line parameters*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`oxygenAuthor.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
   Oxygen XML Author is now installed in a new sub-folder called `oxygenAuthor`.
3. If you wish, you can move the directory where you installed Oxygen XML Author to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `oxygenAuthor16.1`.
4. Start Oxygen XML Author by running `oxygenAuthor.sh`, which is located in the install folder.
5. To license your copy of Oxygen XML Author go to **Help** > **Register...** and enter your *license information*.

Installing Oxygen XML Author on Windows NT Server

Choosing an installer

You can install Oxygen XML Author on Windows using one of the following methods:

- Install using the *Windows installer*.
- Install using the *Windows installer in unattended mode*.
- Install using the *All Platforms installer*. Choose the all platforms installer if you have trouble installing using the Windows installer.

System Requirements

System requirements for a Windows Server install:

Operating systems

Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2

CPU

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

Memory

- Minimum values per user - 512 MB of RAM
- Recommended values per user - 2 GB of RAM

Storage

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

Java

Oxygen XML Author requires Java. If you use the native Windows installer, Oxygen XML Author will be installed with its own copy of Java. If you use the all platforms installer, your system must have a compatible Java virtual machine installed.

Oxygen XML Author supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.7) from Oracle available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*. Oxygen XML Author may work with JVM implementations from other vendors, but there is no guarantee that those implementations will work with future Oxygen XML Author updates and releases.

Oxygen XML Author uses the following rules to determine which installed version of Java to use:

1. If you install using the native Windows installer, which installs a version of Java as part of the Oxygen XML Author installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Windows environment variable `JAVA_HOME` is set, Oxygen XML Author uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your PATH environment variable is used.

If you run Oxygen XML Author using the batch file, `oxygenAuthor.bat`, you can edit the batch file to specify a particular version to use.

Install using the Windows installer

To install Oxygen XML Author using the Windows installer:

1. Make sure that your system meets the *system requirements*.
2. Download the Windows installer.
3. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
4. Run the installer and follow the instructions in the installation program.
5. Start Oxygen XML Author using one of the following methods:

   - Using one of the shortcuts created by the installer.
   - By running `oxygenAuthor.bat`, which is located in the install folder.

6. To license your copy of Oxygen XML Author go to **Help** > **Register...** and enter your *license information*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`oxygenAuthor.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
   Oxygen XML Author is now installed in a new sub-folder called `oxygenAuthor`.
3. If you wish, you can move the directory where you installed Oxygen XML Author to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `oxygenAuthor16.1`.
4. Start Oxygen XML Author by running `oxygenAuthor.bat`, which is located in the install directory.
5. To license your copy of Oxygen XML Author go to **Help** > **Register...** and enter your *license information*.

Configuring Windows NT Terminal Server

Windows NT Terminal Server configuration procedure.

1. Install Oxygen XML Author on the server and make its shortcuts available to all users.

2. If you need to run multiple instances of Oxygen XML Author, make sure you add the *-Dcom.oxygenxml.MultipleInstances=true* parameter in the `.bat` startup script.

3. Make sure you allocate sufficient memory to Oxygen XML Author by adding the `-Xmx` parameter either in the `.bat` startup script, or in the `.vmoptions` configuration file (if you start it from an executable launcher).

Installing Oxygen XML Author on a Linux / UNIX Server

Choosing an installer

You can install Oxygen XML Author on Linux using any of the following methods:

- Install using the Linux installer.
- Install using the Linux installer in unattended mode.
- Install using the all platforms installer. Choose the all platforms installer if you have trouble installing using the Linux installer.

System Requirements

System requirements for a Linux install:

Operating system

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

CPU

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

Memory

- Minimum - 1 GB of RAM
- Recommended - 4 GB of RAM

Storage

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

Java

Oxygen XML Author requires Java. Oxygen XML Author supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.6.0) from Oracle available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*. Oxygen XML Author may work with JVM implementations from other vendors, but there is no guarantee that other implementations will work with future Oxygen XML Author updates and releases. Oxygen XML Author does not work with the GNU libgcj Java Virtual Machine.

Oxygen XML Author uses the following rules to determine which installed version of Java to use:

1. If you used the Linux installer, which installs a version of Java as part of the Oxygen XML Author installation, the version in the `jre` subdirectory of the installation directory is used.

2. Otherwise, if the Linux environment variable `JAVA_HOME` is set, Oxygen XML Author uses the Java version pointed to by this variable.

3. Otherwise the version of Java pointed to by your PATH environment variable is used.

You can also change the version of the Java Virtual Machine that runs Oxygen XML Author by editing the script file, `oxygenAuthor.sh`. Go to the Java command at the end of the script file and specify the full path to the Java executable of the desired JVM version, for example:

```
/usr/bin/jre1.6.0_45/bin/java -Xmx256m ...
```

Linux Installation

Linux installation procedure.

To install Oxygen XML Author on Linux:

1. Download the Linux installer.
2. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
3. Run the installer that you downloaded and follow the instructions presented in the installation program.
4. Start Oxygen XML Author using one of the following methods:

   • Use the `author` shortcut created by the installer.
   • Run `sh oxygenAuthor.sh` from the command line. This file is located in the installation folder.

5. To license your copy of Oxygen XML Author go to **Help** > **Register...** and enter your *license key*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`oxygenAuthor.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
   Oxygen XML Author is now installed in a new sub-folder called `oxygenAuthor`.
3. If you wish, you can move the directory where you installed Oxygen XML Author to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `oxygenAuthor16.1`.
4. Start Oxygen XML Author by running `oxygenAuthor.sh`, which is located in the install folder.
5. To license your copy of Oxygen XML Author go to **Help** > **Register...** and enter your *license information*.

Unix / Linux Server Configuration

To install Oxygen XML Author on a Unix / Linux server:

1. Install Oxygen XML Author on the server and make sure the oxygenAuthor.sh script is executable and the installation directory is in the PATH of the users that need to use the application.
2. If you need to run multiple instances of the Oxygen XML Author, make sure you add the *-Dcom.oxygenxml.MultipleInstances=true* parameter in the startup script.
3. Make sure you allocate sufficient memory to Oxygen XML Author by setting an appropriate value for the `-Xmx` parameter in the `.sh` startup script.
4. Make sure the X server processes located on the workstations allow connections from the server host. For this, use the xhost command.
5. Start telnet (or ssh) on the server host.
6. Start an xterm process, with the **display** parameter set on the current workstation. For example: `xterm -display workstationip:0.0`.
7. Start Oxygen XML Author by typing `oxygenAuthor.sh` .

Installing Oxygen XML Author using the Java Web Start (JWS) Installer

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

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

> **Note:** If you want to create your own JWS distribution package, please contact Syncro Soft for the Oxygen SDK agreement.

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

The following schematics depicts the Oxygen XML Author Java Web Start deployment procedure:
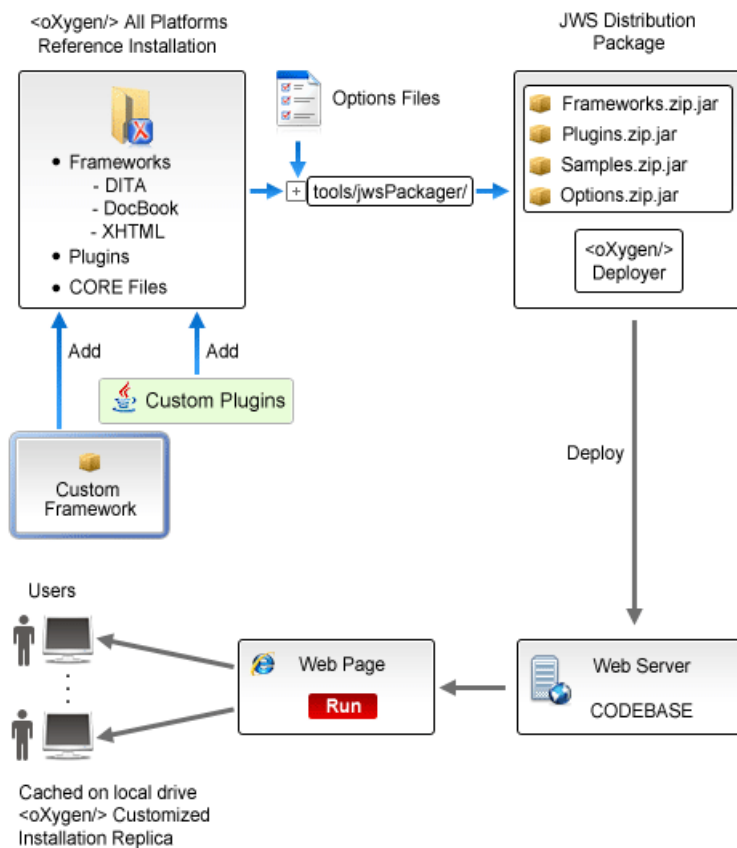
Figure 1: Java Web Start Deployment Procedure

To deploy an Oxygen XML Author installation on a custom server.

1. Go to `http://www.oxygenxml.com/InstData/Author/All/oxygenAuthor.tar.gz` and download the All Platforms Installation package to a local drive.
2. Expand the archive.
   The `oxygenAuthor` folder is created.
3. Optionally, you can customize the content of the **frameworks** folder.
4. Edit the `oxygenAuthor\tools\jwsPackager\packager.properties` configuration file. Adjust the following properties appropriately for your server:
   - **codebase** - represents the location of the future JWS distribution.

- **keystore** - keystore location path.
- **storepass** - password for keystore integrity.
- **storetype** - type of the certificate file, like `PKCS12` or `JKS`.
- **alias** - keystore alias.
- **optionsDir** - points to the options directory that may be distributed with the JWS installer. If the directory contains an XML document named `options.xml` or `default.xml` containing exported options, these options will be used. Otherwise, the structure of the options folder has to match the structure of a stand alone application *options folder*.

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

  The values of **keystore**, **storepass**, and **alias** properties are all provided by the code signing certificate. For more information, please check the documentation of the *jarsigner* tool.

5. Edit the JNLP `oxygenAuthor\tools\jwsPackager\dist\javawebstart\author\author.jnlp` template file to modify default settings. You can specify the list of files opened at startup by modifying the `<argument>` list. To pass system properties directly to Oxygen XML Author when it is started, add the `oxy` prefix to them (for example: `<property name="oxyPropertyName" value="testValue"/>`). The system property is passed to Oxygen XML Author with the prefix stripped.
6. Open a command-line console and run `ant` in the `oxygenAuthor\tools\jwsPackager` folder. The **ant** process creates the `oxygenAuthor\tools\jwsPackager\dist\InstData\authorJWS.zip` archive that contains the actual remote JWS installer.
7. Copy the expanded content of the archive to the folder specified in the **codebase** property, previously set in the **packager.properties** file.
8. ⚠️ **Important:** When running the Java Web Start distribution on OS X, due to changes in this *security release*, clicking the link to the JNLP file does not start the application. The selected JNLP is downloaded locally. Right click it and choose to open the resource.

Using your favorite web browser, go to the address specified in the **codebase** property or to its parent folder and start the remote installer.

Site-wide Deployment

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

Creating custom default options

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

Creating default project files

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

Shared project files

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

Using the unattended installer

You can speed up the installation process by using the *unattended installer for Windows* or *Linux installs*.

Using floating licenses

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

Obtaining and Registering a License Key for Oxygen XML Author

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

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

Choosing a license type

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

1. A named-user license may be used by a single named user on one or more computers. Named-user licenses are not transferable to a new named user. If you order multiple named-user licenses, you will receive a single license key good for a specified number of named users. It is your responsibility to keep track of the named users that each license is assigned to.
2. A floating license may be used by any user on any machine. However, the total number of copies of Oxygen XML Author in use at one time must not be more than the number of floating licenses available. A user who runs two different distributions of Oxygen XML Author (for example Standalone and Eclipse Plugin) at the same time on the same computer, consumes a single floating license.

For definitions and legal details of the license types, consult the End User License Agreement available at *http://www.oxygenxml.com/eula_author.html*.

Obtaining a license

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

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

Register a named-user license

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

1. Save a backup copy of the message containing the new license key.
2. Start Oxygen XML Author.
   If this is a new install of Oxygen XML Author, the registration dialog is displayed. If the registration dialog is not displayed, go to **Help** > **Register...**.
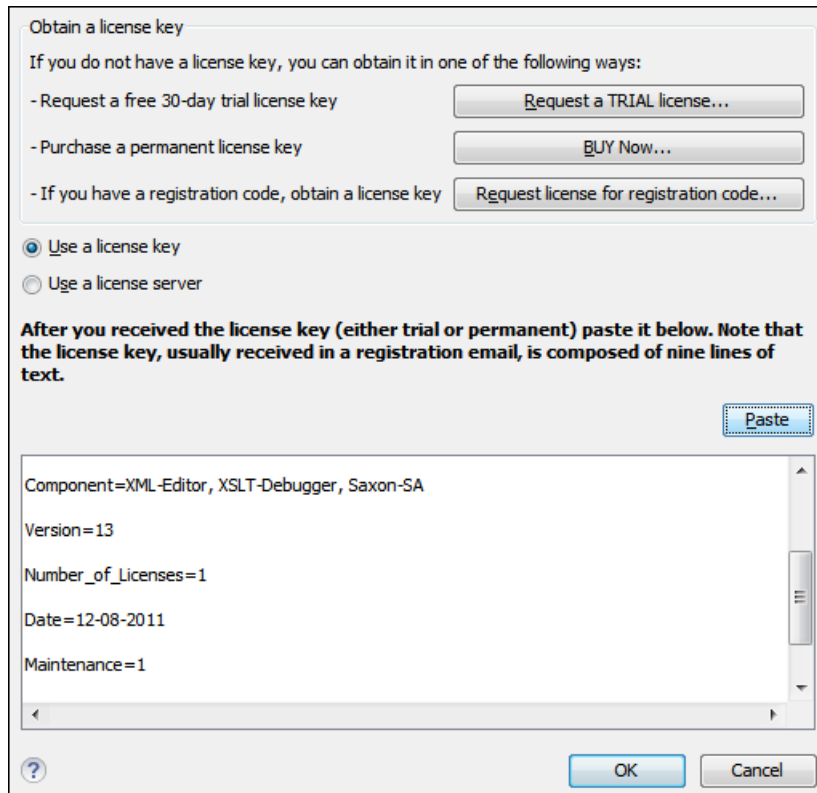
Figure 2: License Registration Dialog

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

Register multiple licenses

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

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

To register using a text file:

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

To register using an XML file:

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

Registering a floating license

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

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

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

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

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

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

If a floating license is available, it is registered in Oxygen XML Author. To display the license details, open the **About** dialog from the **Help** menu. If a floating license is not available, you will get a message listing the users currently using floating licenses.

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

1. Contact your server administrator to get network address and login details for the license server.
2. Start Oxygen XML Author.
3. Go to **Help** > **Register** .
   The license registration dialog is displayed.
4. Choose **Use a license server** as licensing method.
5. Select **HTTP/HTTPS Server** as server type.
6. In the *URL* field enter the address of the license server.
   The URL address has the following format:
   `http://hostName:port/oXygenLicenseServlet/license-servlet`
7. Complete the *User* and *Password* fields.
8. Click the **OK** button.

If a floating license is available, it is registered in Oxygen XML Author. To display the license details, open the **About** dialog from the **Help** menu. If a floating license is not available, you will get a message listing the users currently using floating licenses.

Release a Floating License

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

- The connection with the license server is lost.

- You exit the application running on your machine, and no other copies of Oxygen XML Author running on your machine are using your floating license.

- You register a named user license with your copy of Oxygen XML Author, and no other copies of Oxygen XML Author running on your machine are using your floating license.

Setting Up a Floating License Server for Oxygen XML Author

Determine if your need to set up a license server

If you are using floating licenses for Oxygen XML Author, you may need to set up a license server.

- If the computers that will be using the floating licenses are all on the same network segment, Oxygen XML Author can manage the licenses by itself. Different running instances of Oxygen XML Author communicate among themselves, using UDP broadcast on the 59153 port, to the 239.255.255.255 group.
- If the computers that will be using the floating licenses are on different network segments, you must use an Oxygen XML Author floating license server. A floating license server can be installed either as a *Java servlet* or as a *standalone process*.

Split or combine license keys to work with your license servers

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

Setting up a Floating License Server Running as a Java Servlet

Setting up the floating license servlet.
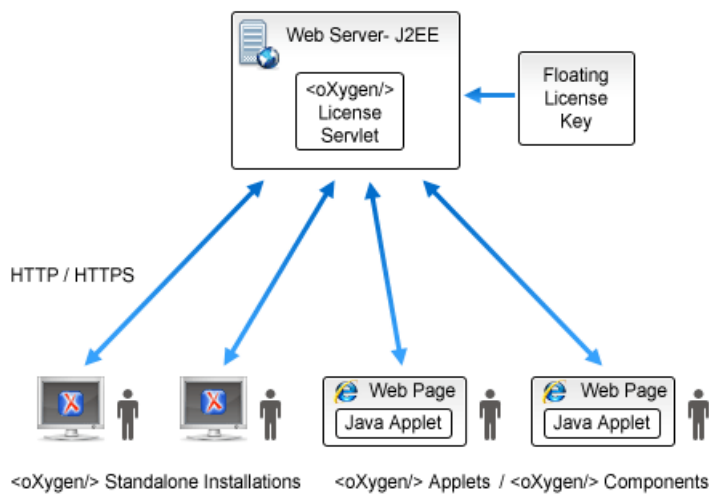
Figure 3: Floating License Server Running as a Servlet

1. Make sure that Apache Tomcat 5.5 or higher is running on the machine you have selected to be the license server. To get it, go to *http://tomcat.apache.org*.
2. Download the **Web ARchive** (**.war**) license servlet from one of the download URLs included in the registration email message.
3. Go to the Tomcat Web Application Manager page. In the **WAR file to deploy** section choose the WAR file and click the **Deploy** button. The *oXygen License Servlet* is now up and running, but there are no license keys registered yet.
4. To register a license key, log on the license server 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 into this file and save it.
5. We recommend that you password protect your pages using a Tomcat Realm. Please refer to the *Tomcat Documentation* for more information.

6. Once you have defined a Realm resource, 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.

7. By default, the license server logs 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.

8. Restart *oXygen License Servlet* from the Tomcat Web Application Manager page.

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 Author runs
  - MAC address of the computer where Oxygen XML Author 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.
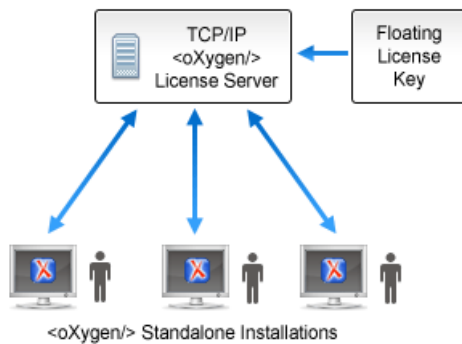
Figure 4: Floating License Server Running as a Standalone Process

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. You can:

   - 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, 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 OS X and Unix / Linux. It has 2 parameters:

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

   The following is an example command line for starting the license server on Unix/Linux and 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 Author version 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. Ensure that a up-to-date version of Java is installed on the server machine and that the JAVA_HOME variable points to the location Java is installed.

2. Add oXygenLicenseServer.exe manually in the Windows Firewall list of exceptions. Go to **Control Panel** > **System and Security** > **Windows Firewall** > **Allow a program or feature through Windows Firewall** > **Allow another program** and browse for oXygenLicenseServer.exe from the Oxygen XML Author License Server installation folder.

3. Download the Windows installer of the license server from the URL provided in the registration email message containing your floating license key.

4. Run the downloaded installer.

5. 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 manually install, start, stop, or uninstall the server as a Windows service, 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:** 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.*

Transferring or Releasing a License Key

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

1. Go to **Help** > **Register...**.
   The license registration dialog is displayed.
2. The license key field should be empty (this is normal). If it is not empty, delete any text in the field.
3. Make sure the option **Use a license key** is selected.
4. Click **OK**.
   A dialog is displayed asking if your want to reset your license key.

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

   The **Reset** button erases all the licensing information. To complete the reset operation, close and restart Oxygen XML Author.

Upgrading Oxygen XML Author

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

Checking for New Versions of Oxygen XML Author

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

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

What is preserved during an upgrade

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

If you install over a previously installed version:

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

If you install in a new directory.

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

Upgrading the Standalone Application

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

2. Download and install the new version according to the instructions for your platform and the type of installer you selected.

3. If you installed from an archive (as opposed to an executable installer) you may have to update any shortcuts you have created or modify the system PATH to point to the new installation folder.

4. Start Oxygen XML Author.

5. If you require an new license for your upgrade, install it now according to the procedure for your platform and the type of installer you selected.

Installing and Updating Add-ons in Oxygen XML Author

Oxygen XML Author provides an add-on mechanism that can automatically discover and install *frameworks* and *plugins* from a remote location.

• 📑 **Note:** Frameworks that you install through the add-ons system are read-only.

Installing Add-ons

To install a new add-on, follow these steps:

• go to **Help** > **Install new add-ons...**
• in the opened dialog, fill-in the **Show add-ons from** with the *update site* that hosts add-ons. The add-ons list contains the name, status, update version, Oxygen XML Author version, and the type of the add-on (either framework, or plugin). A short description of each add-on is presented under the add-ons list.

  📑 **Note:** To see all the add-ons from the remote update site, disable **Show only compatible add-ons** and **Show only the latest version of the add-ons**. Incompatible add-ons are shown only to acknowledge their presence on the remote update site. You cannot install an incompatible add-on.

• by default, only the latest versions of the add-ons that are compatible with the current version of Oxygen XML Author are displayed
• choose the add-ons you want to install, press the **Next** button, then follow the on-screen instructions.

  📑 **Note:** Accepting the license agreement of the add-on is a mandatory step in the installation process

  📑 **Note:** All add-ons are installed in the *preferences directory* of Oxygen XML Author, under the `extensions` directory.

Managing installed add-ons

To manage the installed add-ons, follow these steps:

• go to **Help** > **Manage add-ons...**
• the opened dialog presents a list with the available updates (compatible with the current version of Oxygen XML Author) and with the already installed updates. Under the updates list, Oxygen XML Author presents a short description of each update.
• check an add-on, then press **Update** to update it (or **Uninstall** to remove it). If there is available a newer version of the add-on, Oxygen XML Author will download the package and install it. Follow the on-screen instructions to complete the installation process

  📑 **Note:** Accepting the license agreement of the add-on is a mandatory step in the installation process

Checking for add-on updates

To check if there are available updates for the installed add-ons, go to **Help** > **Check for add-ons updates...**. This action displays only updates compatible with the current Oxygen XML Author version.

To watch a video demonstration about the add-ons support in Oxygen XML Author, go to *http://www.oxygenxml.com/demo/AddonsSupport.html*.

Uninstalling Oxygen XML Author

Uninstalling the Oxygen XML Author Standalone

⚠ **Caution:** The following procedure will remove Oxygen XML Author from your system. **All data stored in the installation directory will be removed, including any customizations or any other data you have stored within that directory. Please make a back up of any data you want to keep before uninstalling Oxygen XML Author.**

1. Backup any data you want to keep from the Oxygen XML Author installation folder.
2. Remove the application.

   - On Windows use the appropriate uninstaller shortcut provided with your OS.
   - On OS X and Unix manually delete the installation folder and all its contents.

3. If you want to remove the user preferences:

   - **On Windows**, remove the directory: `%APPDATA%\com.oxygenxml.author` (usually %APPDATA% has the value `[user-home-dir]\Application Data`).
   - **On Linux**, remove the directory: `.com.oxygenxml.author` from the user's home directory.
   - **On Mac OS X**, remove the directory: `Library/Preferences/com.oxygenxml.author` of the user home folder .

Unattended Uninstall

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

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

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

Oxygen XML Author Installer Command Line Reference

Command line options of the Oxygen XML Author installer.

The `response.varfile`

The Oxygen XML Author installers for Windows and Linux creates a file called `response.varfile`, which records the choices that the user made when running the installer interactively. You can use a `response.varfile` to set the options for an unintended install. Here is an example of a `response.varfile`:

```
#install4j response file for Oxygen XML Editor 16.0
#Fri Jul 18 21:52:15 EDT 2014
sys.adminRights$Boolean=true
sys.programGroupDisabled$Boolean=false
createDesktopLinkAction$Boolean=true
autoVersionChecking=true
sys.fileAssociation.launchers$StringArray="19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19"
sys.programGroupAllUsers$Boolean=true
reportProblem=true
downloadResources=true
sys.languageId=en
sys.installationDir=C:\\Program Files (x86)\\Oxygen XML Editor 16
createQuicklaunchIconAction$Boolean=true
sys.programGroupName=Oxygen XML Editor 16.0
executeLauncherAction$Boolean=true
sys.fileAssociation.extensions$StringArray="xml","dita","ditamap","ditaval","xsl","xslt","xspec","xsd","rng","rnc","sch","dtd","mod","ent","nvdl","fo","wsdl","xquery","xq","xqy","xqm","xql","xpl","css","json","xpr"
```

The following table describes some of the settings that can be used in the `response.varfile`:

Table 1: `response.varfile` Options Parameters

Parameter name	Description	Values
autoVersionChecking	Automatic version checking.	`true` / `false`. Default setting is `true`.
`reportProblem`	Allows you to report a problem encountered while using Oxygen XML Author.	`true` / `false`. Default setting is `true`.
downloadResources	Allows Oxygen XML Author to download resources (links to video demonstrations, webinars and upcoming events) from *http://www.oxygenxml.com* to populate the application welcome screen.	`true` / `false`. Default setting is `true`.

The Oxygen XML Author installation uses the install4j installer. A description of *the `response.varfile` format* can be found on the install4j site.

Command line parameters

The Oxygen XML Author installer supports the following command line parameters:

Option	Meaning
`-q`	Run the installer in unattended mode. The installer will not prompt the user for input during the install. Default settings will be used for all options unless a `response.varfile` is specified using the `-varfile` option or individual settings are specified using **- on Windows:** `oxygenAuthor.exe -q` **- on Linux:** `oxygenAuthor.sh -q`
`-overwrite`	In unattended mode, the installer does not overwrite files with the same name if a previous version of the Oxygen XML Author is installed in the same folder. The -overwrite parameter added after the -q parameter forces the overwriting of these files. **- on Windows:** `oxygenAuthor.exe -q -overwrite` **- on Linux:** `oxygenAuthor.sh -q -overwrite`
`-console`	To display a console for the unattended installation, add a -console parameter to the command line. **- on Windows:** `start /wait oxygenAuthor.exe -q -console` 📄 **Note:** The use of `start /wait` on Windows is required to make the installer run in the foreground. It you run it without `start /wait`, it will run in the background. **- on Linux:** `oxygenAuthor.sh -q -console`

Option	Meaning

-varfile	Points to the location of a `response.varfile` to be used during an unattended installation. For example:
	- on Windows:
	
	```oxygenAuthor.exe -q -varfile response.varfile```
	
	- on Linux:
	
	```oxygenAuthor.sh -q -varfile response.varfile```

-V	Is used to define a variable to be used by an unattended installation. For example:
	- on Windows:
	
	```oxygenAuthor.exe -q -VusageDataCollector=false```
	
	- on Linux:
	
	```oxygenAuthor.sh -q -VusageDataCollector=false```

The Oxygen XML Author installation uses the install4j installer. A full list of the *command line parameters supported by the install4j installer* can be found on the install4j site.

Chapter

3

Getting Started

Topics:

The basic steps for creating an XML document are presented here.

Your First XML Document

To create your first XML document, choose **File** > **New...**. The *New Document Wizard* is displayed:
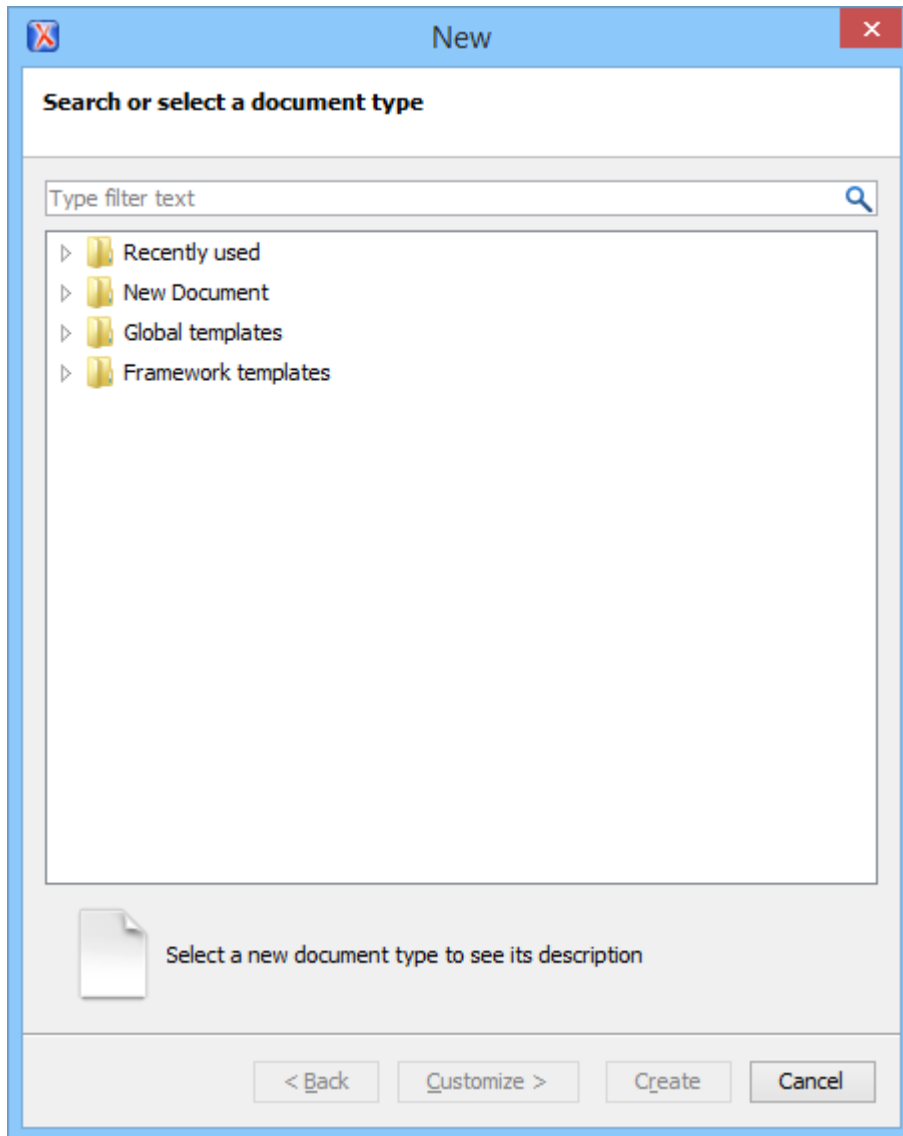
You can either create a new XML document from scratch by choosing one of the available types in the **New Document** folder, or you can choose to create one from a template by choosing a template from the **Global templates** or **Framework templates** folders. If you are looking for a common document types such as DITA or DocBook, you will find templates for these document types in the **Framework templates** folder. If your company has created its own templates, your will also find them there. After you use this dialog to create a few documents, those document types will appear in the **Recently used** folder so you can easily create other new documents of those types.

For some document types, you may find a number of different templates. For example, there are a number of different templates for different DocBook documents, and for different DITA topic types and maps. Choose the template that best meets your needs.

Writing your first document

Depending on the type of document you choose, the Oxygen XML Author interface will change to support editing that document type. This may include new menus, new toolbar buttons, and new items on the context menus.

Also, depending on the type of document you choose, Oxygen XML Author may open your document in *Text* or *Author* mode. Text mode shows the raw XML source file. Author mode shows a graphical view of the document.

Whether there is an Author mode view available for your document type depends on what type you choose and whether there is an Author mode CSS style sheet available to create the Author view. Oxygen XML Author supplies default Author mode views for most of the document types it supports. If your company has created its own document types, it may also have created Author mode style sheets for that type. However, if you choose to create a plain XML file, or to create one based on a schema for which Oxygen XML Author has no built in support, you will have to edit in Text mode or *create your own Author mode style sheet* for it.

You can switch back and forth between Author mode and Text mode at any time by clicking the buttons at the bottom left of the editor window. You won't lose any formatting when switching from Author to Text mode. Text and Author mode are just different views on the same XML document. (There is also a *Grid mode* available, which is useful for certain kinds of documents, particularly those that are structured like databases. You can also use it to *sort things like list items and table rows*.)

If you are using Author mode, you will find that it works a lot like whatever word processor you are used to. If you are using Text mode, you will find it works a lot like many typical text editors. The biggest difference you will find, if you are new to XML, is that XML documents have a particular structure that you have to follow.

Structuring your first document

Each XML document type has a particular structure that you have to follow as you write and edit the document. Some document types give you a lot of choices. Some give you very few. In either case, you need to make sure that your document follows the structure for the document type you are creating. This means:
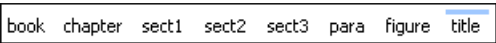
- At any given point in the document, there are only certain XML elements allowed. Oxygen XML Author helps you determine which elements are allowed. In Author mode, when you hit Enter, Oxygen XML Author assumes you want to enter a new element and shows you a list of elements that can be created in this spot. Keep typing until the element you want is highlighted and press **Enter** to insert the element. If you want to get an idea of the overall structure of a document and what is allowed where, you can check out the **Model** view (**Window** > **Show View** > **Model**).
- When you create certain elements, you may find that your text gets a jagged red underline and you get a warning that your content is invalid. This is usually because the element you have just created requires certain other elements inside it. Your document will be invalid until you create those elements. Oxygen XML Author will do its best to help you with this. If there is only one possible element that can go inside the element you just created, Oxygen XML Author will create it for you. If there is more than one possibility, though, you will have to create the appropriate elements yourself.

Editing your first document

Once you have completed the first draft of your document, you may need to edit it. As with any editor, Oxygen XML Author provides the normal cut, copy, and paste options as well as drag and drop editing. However, when you are editing an XML document, you have to make sure that your edits respect the structure of the XML document type. In fact, you are often editing the structure as well as the content of your document.

Oxygen XML Author provides many tools to help you edit your structure and to keep your structure correct when editing text. These include:

The document breadcrumbs

Across the top of the editor window there is a set of breadcrumbs that shows you exactly were the insertion point is in the structure of the document. book chapter sect1 sect2 sect3 para figure title . You can click on any element in the breadcrumbs to select that entire element in the document.

Showing tags

To see exactly where you are in the structure of the document, you can show the tags graphically in the Author view. There are several levels of tag visibility that you can choose using the ⏵◁ ▾ toolbar button. (The button may look a little different as it changes to reflect the level of tags currently displayed.)

Outline view

The outline view shows you the structure of your document in outline format. You can use the outline to select elements, or to move elements around in the document.
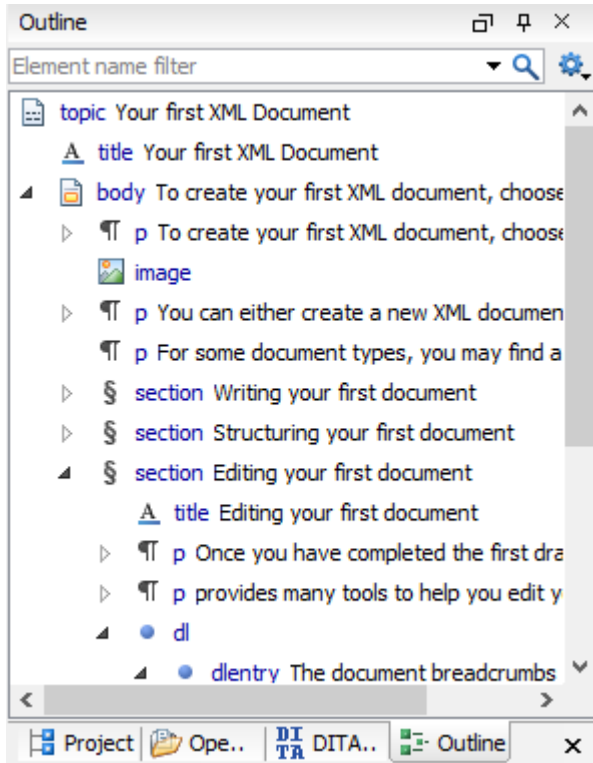
You can configure the outline view to determine what will be shown in the outline, such as element names, attributes, and comments. Different choices may work better for different document types. You can also filter the outline to show only elements with a certain name.
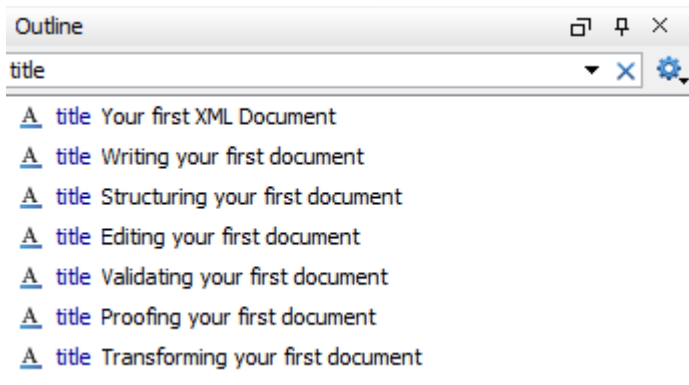
Cut and paste, drag and drop

You can cut and paste or drag and drop text just as you would in any other editor. However, when you do this in Author view it is important to remember that you are actually moving blocks of XML, and when you cut and paste or drag and drop a block of XML, the result has to be valid both where the content is inserted, and where it is removed from.

A big part of getting this right is to make sure that you pick up the right block of text in the first place. Using the breadcrumbs or the Outline view, or turning on show tags and using the displayed tags to select content can all help ensure that you are selecting the right chunk of XML.

If you do try to paste or drop a chunk of XML somewhere that would not be valid, Oxygen XML Author will warn you and will try to suggest how to make it valid, either by removing surrounding elements from the chunk you are moving, by creating a new element at the destination, or by inserting it in a nearby location.

You can also switch to Text view to see exactly which bits of XML you are selecting and moving.

Refactoring actions

You can perform many common structure edits, such as renaming an element or wrapping text in an element, using the actions on the XML Refactoring menu (**Document** > **XML Refactoring**). The refactoring actions are also available from the context menu and on the Refactoring toolbar. (Right click the toolbar area to choose which tool bars to show.)

Validating your first document

Validation is the process of checking that an XML document obeys the rules of its schema. If Oxygen XML Author knows how to find the schema, it will validate the document for you as you type. (Oxygen XML Author can find the schema automatically for most of the document types created from templates. In some cases, though, you may have to *tell it how to find the schema*.)

When Oxygen XML Author validates as you type, there is a small bar at the right edge of the editor that tells you if the document is valid, and where in the document any errors are found. If the indicator at the top of that bar is green, your document is valid. If the document is invalid, the indicator turns red and a red bar shows you where in the document the error was found. Click on that bar to jump to the error. Remember that sometimes your document is invalid simply because the structure you are creating is not complete yet.

In addition to problems with the validity of the XML document itself, Oxygen XML Author will also report warnings for a number of conditions such as if your document contains a cross reference that cannot be resolved, or if Oxygen XML Author cannot find the schema specified by the document. The location of these warning will be marked in yellow on the validation bar. If the document contains warnings but no errors, the validity indicator will turn yellow.

You can also validate your document at any time by clicking the Validate button or choosing **Document** > **Validate...**. When you validate this way, the validation result will open in a new pane. It will show each validation error on a separate line. Clicking on the error will take you to the place in your document where the error was detected. (Be aware that the problem is sometimes in a different place from where the validator was able to detect the error.)

To get more information on a validation error, right-click on a validation error message, and choose **Show Message**.

Proofing your first document

Oxygen XML Author provides spell checking support. You can check the spelling of your document by pressing the spell check button or choosing **Edit** > **Check Spelling...**. Interactive spell checking is also available. It is turned off by default. (You may have noticed that validation errors are marked the same way spelling errors are marked in ordinary word processors.) To turn it on, *open the Preferences dialog* and go to **Editor** > **Spell Check** > **Automatic spell check**.

Transforming your first document

An XML document must be transformed in order to be published. Transformations are specific to the particular type of document you have created. A DITA transformation cannot be used on a DocBook file or vice versa. A single document type may have many different transformations that produce different kinds of outputs. For some document types, such a DITA, many different content files may be combined together by a transformation. You need to locate and launch a transformation that is right for your document type and the kind of output you want to generate.

Oxygen XML Author uses *transformation scenarios* to control the transformation process. Depending on the document type you have created, there may be several transformation scenarios already configured for your use. This may include the default transformation scenarios supplied by Oxygen XML Author or ones created by your organization.

To see the list of transformations available for your document, click the transform button or select **Document** > **Transformation** > **Apply Transformation Scenario(s)**. A list of available transformation scenarios will be displayed.

Choose one or more scenarios to apply, and click **Apply associated**. Exactly how your transformed content will appear will depend on how the transformation scenario is configured.

Your First DITA Topic

To create your first DITA topic, choose **File** > **New...**. The *New Document Wizard* is displayed:
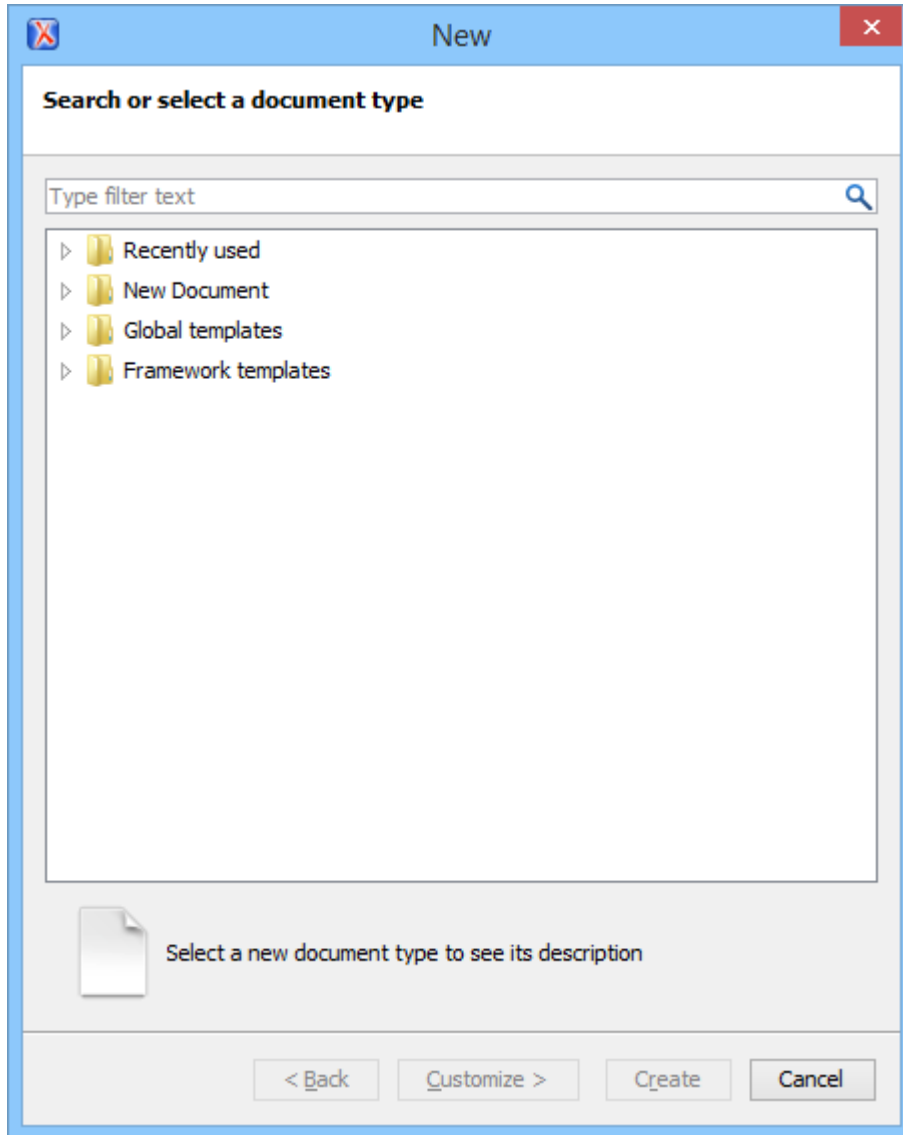
Go to **Framework templates** > **DITA** > **topics** and select the topic type that you want to create.

> **Note:** If you organization has created any kind of DITA customizations, the appropriate template files may be in another location, and different topic types may be provided for your use. Check with the person who manages your DITA system to see if you should be using templates from a different directory.

Your DITA topic is an XML document. As such, all *the editing features that Oxygen XML Author provides for editing XML documents* apply to DITA topics. However, Oxygen XML Author also provides extensive additional support for editing DITA topics and their associated maps and other files, and for creating DITA outputs.

Understanding DITA topics

It is important to understand the role that a DITA topic plays in a DITA system. A DITA topic is not associated with a single document. It is a separate entity that can potentially be included in many different documents, help systems, or websites. When you write a DITA topic, therefore, you are not writing a book or a help systems or a website, you are writing an individual topic. This affects how you approach the writing task and how Oxygen XML Author works to support you as you write.

Of course, most of your topics are actually connected more or less strongly to other topics, and those connections can affect how you write and how you handle things like links and reuse as you write. Oxygen XML Author can help you manage those relationships. Depending on how tightly or loosely your topics are connected, you may use Oxygen XML Author's tools (and features of DITA itself) in different ways.

The role of maps

The basic way that DITA expresses the relationship between topics is through a DITA map. Other relationships between topics, such as cross references, generally need to be between topics in the same map, as DITA uses maps to determine which topics will be part of any output set that you create. While customized DITA solutions can use other mechanisms, generally DITA does not provide a way to output individual topics. Output is created from a map and includes the topics referenced by the map.

A publication is not always represented by a single map. For instance, if you are writing a book, you might use a map to create each chapter, and then collect the chapters together in another map to create the book. If you are writing help topics, you might use a map to combine several DITA topics to create a single help topic, and then create another map to organize your help topics into a help systems. In this way, you can potentially reuse a map that represents a chapter or a help topic in more than one book or help system.

Adding your topic to a map

To add your topic to a map, you must first create a map. A map is an XML document, just like a topic, so to create one, you choose **File** > **New...** and **Framework templates** > **DITA Map** > **map** and select the type of map you want. (If in doubt, choose Map for now.) Oxygen XML Author will ask you if you want to open your map in the editor or in the **DITA Maps Manager**. Most of the time, opening it in the **DITA Maps Manager** is the best choice. You can also open the map in the editor from the **DITA Maps Manager** if you need to.
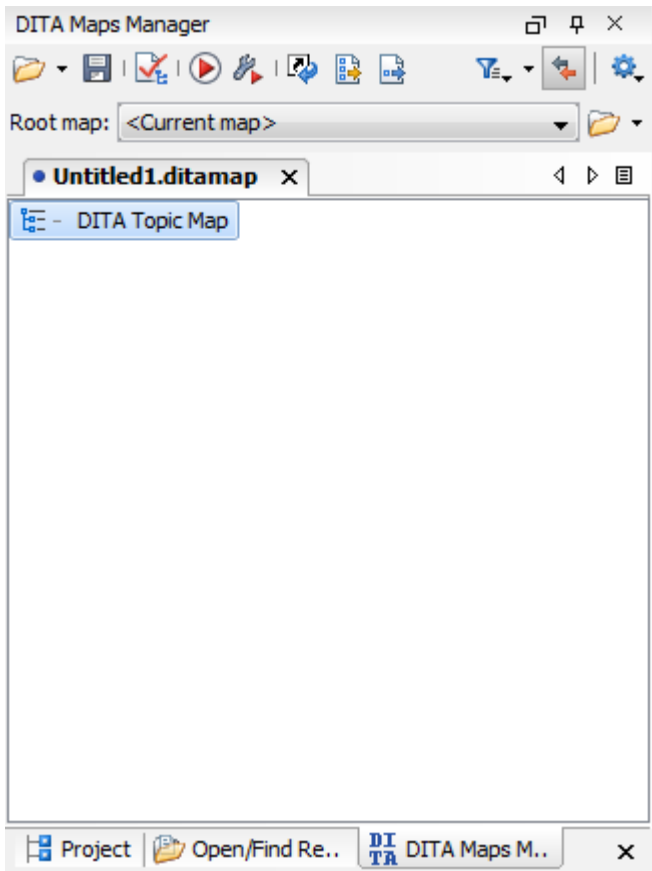
The DITA Maps Manager presents a view of the DITA map similar to a table of contents. To add a topic to a map, you add a topic reference to the map which points to your topic. The easiest way to do this it to have the topic open in the editor and right-click on **DITA Topic Map** in the **DITA Maps Manager** view and choose **Append child** > **Reference to the currently edited file...**. This brings up the **Insert Reference** dialogue with all of the required fields already filled in for you. You can fill in additional information here or add it to the map later. When you select **Insert and Close**, a reference to your topic is added to the map.
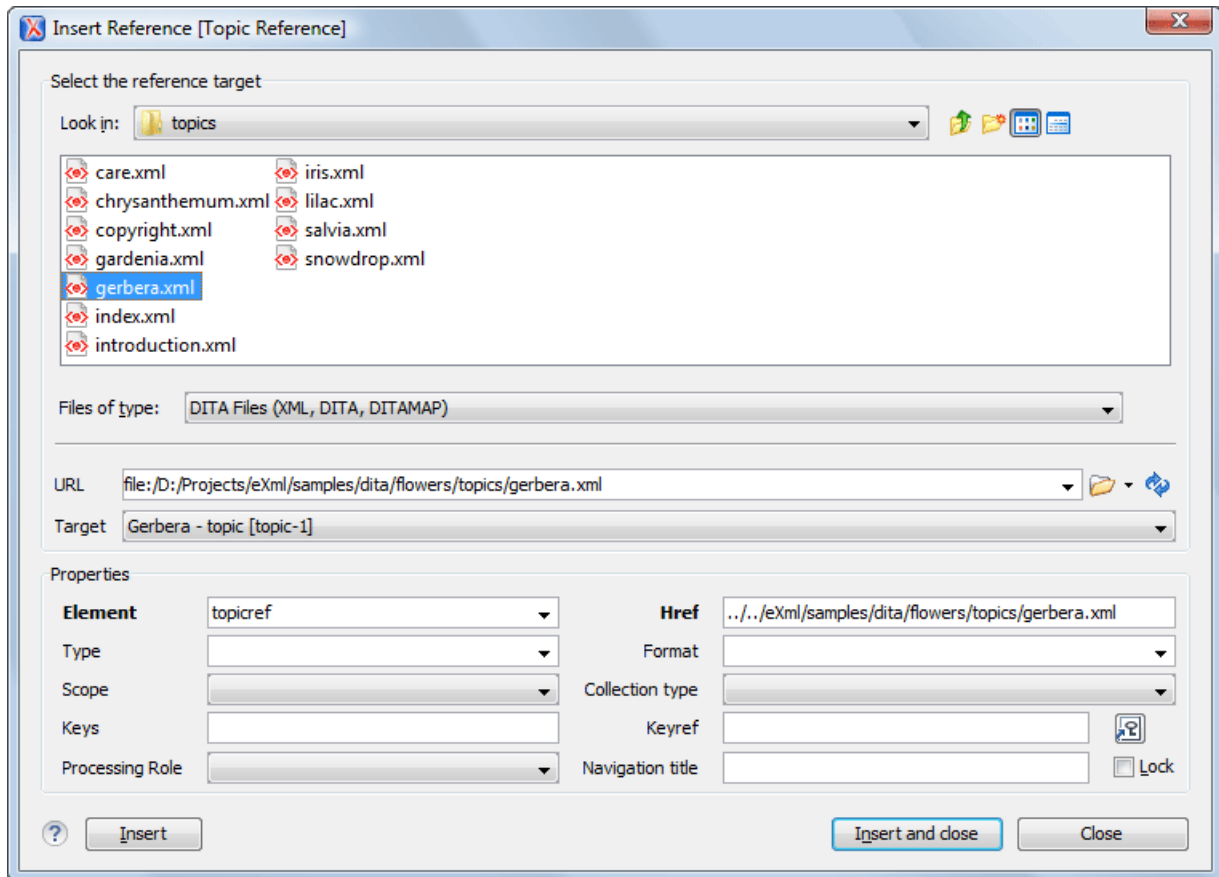
If you want to see what the resulting map looks like in XML, save your map and then double click on **DITA Topic Map** in the **DITA Maps Manager** view. The XML version of the map will open in the editor.

> **Note:** Maps have titles. The default title in Oxygen XML Author is **DITA Topic Map**, which is what is shown in the **DITA Maps Manager** view. You can (and should) change the title by right clicking on the default title, **DITA Topic Map**, and selecting **Edit properties...**. In this description, however, we will continue to use the title **DITA Topic Map**.

Child topics

As you add more topics to your map, you may want to make one topic the child of another topic. Making a topic a child of another is usually done at the map level, by making one topic reference the child of another. To create a child topic reference, right click on the parent topic in the DITA Maps Manager view and choose **Append child.** You can then choose one of the following options:

- **New topic...**, to create a new topic

- **Reference...**, to create a reference to an existing topic

- **Reference to the currently edited file...**, to make the file that is currently open in the editor the child topic

You can also change the order and nesting of topics in the **DITA Maps Manager** view by selecting the topic to move, holding down the **Alt** key, and using the arrow keys.

How your parent and child topics are displayed relative to each other in any particular output will depend both on the configuration of those topics in the map and on the rules of the output transform that is applied to them. Also, don't assume that your topics have to have the same relationship to each other in all outputs. Only the map defines the relationship between the topics, not the topics themselves. It is perfectly possible to create different maps, with different relationships and different configuration options, to produce different outputs.

Child maps

If you have a large information set, such as a long book or an extensive help system, a single map can become long and hard to manage. You can break content up into smaller maps. A smaller map might represent a chapter of a book, a section of a reference, or even a single help topic or web page made up of a number of smaller DITA topics.

To build a publication out of these smaller maps, you must add them to a map that represents the overall publication. To add a child map to a the current map, right click on the map title (**DITA Topic Map**, by default) and choose **Append child** > **Map reference...**.

Validating a map

Just as you should validate your topics, you should validate your maps. Oxygen XML Author provides a validation function for DITA maps that does more than just validate that they are correct XML. It also validates all of the relationships described by the maps, validates all the topic files, and validates any links between topics described in the topic files. Validating the map that describes your entire publication, therefore, validates all the files that make up the publication and all of the relationships between them. To validate a map, click the Validate and **Check for Completeness** button () in the **DITA Maps Manager** view.

Publishing your topics

As noted above, in DITA you do not publish a topic by itself. To create published output, you run a DITA transformation on a map. This collects all the topics referenced in the map, organizes them, and produced output in a particular format. By default, Oxygen XML Author uses the transformations provided by the DITA Open Toolkit for publishing to a number of different output format. Your organization may have created different transformations, or modified the DITA Open Toolkit transformations. In either case, Oxygen XML Author manages transformations using transformation scenarios.

To publish a map, you must select the transformation scenario you want to run, and, if necessary, set any parameters it requires. To select a transformation, click the **Configure Transformation Scenario(s)** button in the **DITA Maps Manager** view (). This brings up the **Configure Transformation Scenario(s)** dialog.

Configure Transformation Scenario(s)

Type filter text

Scenario	Type
▲ DITA Map (13)	
🔑 DITA Map PDF	DITA OT
🔑 DITA Map WebHelp	DITA OT
🔑 DITA Map WebHelp - Mobile	DITA OT
🔑 DITA Map WebHelp with Feedback	DITA OT
🔑 DITA Map XHTML	DITA OT
🔑 DITA Map EPUB	DITA OT
🔑 DITA Map CHM	DITA OT
🔑 DITA Map ODF	DITA OT
🔑 DITA Map RTF	DITA OT
🔑 DITA Map Eclipse Help	DITA OT
🔑 DITA Map Kindle	DITA OT
🔑 Run DITA OT Integrator	ANT
🔑 DITA Map Metrics Report	XSLT
▲ Project (35)	
🔲 Author User Manual - EPUB	DITA OT
🔲 Author User Manual - Eclipse Help	DITA OT
🔲 Author User Manual - PDF (using Prince)	DITA OT
🔲 Author User Manual - PDF (using XEP)	DITA OT
🔲 Author User Manual - WebHelp	DITA OT

☑ Association follows selection

New   Edit   Duplicate   Remove

▲ Associated scenarios

🔑 DITA Map PDF ✖

? Save and close   Apply associated (1)   Cancel

Choose the transformation (or transformations) you want to apply and click **Apply associated**. Depending on how the transformation scenario is configured, your output may be opened in the appropriate application automatically. To change how a transformation is configured (or just to figure out where this transformation places its output), select the transformation and press **Edit**.

Chapter

4

Perspectives

Topics:

- *Perspectives*
- *Dockable Views and Editors*

This chapter describes the editing perspectives of Oxygen XML Author.

Perspectives

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

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

Editor perspective

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

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.

Editor Perspective

To edit the content of your documents, use the **Editor** perspective (**Window** > **Open perspective** > **Editor**).

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

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:

- **Main menu** - Provides menu driven access to all the features and functions available in Oxygen XML Author.
- **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.
- **Outline view** - It provides an XML document overview and offers functions such as modifications follow-up, document structure change, document tag selection, elements filtering.
- **Model view** - Presents the current edited element structure model and additional documentation as defined in the schema.
- **Results view** - Displays result messages obtained by performing user operations like search (the results of the **Find All** action applied to the current file or the results of a find action applied on a set of files), *validation*, *transformation* and *spell check*. The following actions are available in a vertical toolbar on the right side of the view:

    Grouping options

    Allows you to choose grouping criteria, which reflects into a flat or tree-like presentation of the results. To display the results in a flat layout, you can either uncheck all selected criteria or press the **Ungroup all** button. All these actions are also available in the table header contextual menu.

    Remove selected

    Removes the currently selected messages from the list.

    Remove all

    Clears the message list.

    A contextual menu available on the table grid allows you to:

    - Navigate to previous and next message. As alternative, you can use the default shortcut keys: **Ctrl+Shift+] (Command+Shift+] on OS X)** for navigating to the next and **Ctrl+Shift+[(Command+Shift+[on OS X)** for navigating to the previous message.
    - Print the results or save them either in text or XML format.
    - Expand or collapse all displayed items (with the shortcuts **Alt+Shift + PageUp** for the **Expand All** action and **Alt+Shift + PageDown** for the **Collapse All** action).

- Restore default grouping criteria and table column widths.

- **Project view** - Enables the definition of projects and logical management of the documents they contain.
- **Attributes view** - Presents all possible attributes of the current element
- **Elements view** - Presents a list of all defined elements that you can insert at the current caret position according to the document's schema.
- **Entities view** - Displays a list with all entities declared in the current document as well as built-in ones.
- **Transformation Scenarios view** - Displays a list with all currently configured transformation scenarios.

The Results View

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

- *Validate action*
- *Transform action*
- *Check Spelling in Files action*
- **Find All** action from *the Find/Replace dialog*
- *Find/Replace in Files dialog*
- *XPath expression results*
- *SQL results*

Description - 12 items	Resource	Location
◢ D:\Projects\UserGuide\DITA\topics\batch-transformation.dita (2 items)		
href="results-view.dita#results-view" format="dita">Results View</xref>. All entries in the	batch-transformation.dita	29:61
Results View point to the location of the code that triggered them. </p>	batch-transformation.dita	30:7
◢ D:\Projects\UserGuide\DITA\topics\editor-perspective.dita (1 item)		
<uicontrol>Results view</uicontrol> - Displays result messages obtained by performing user	editor-perspective.dita	52:22
◢ D:\Projects\UserGuide\DITA\topics\oxygen-xpath-view.dita (1 item)		
<title>The XPath Results View</title>	oxygen-xpath-view.dita	12:30

Find in Files - Resul... ✕

Figure 5: Results View

The view provides a toolbar with the following actions:

Group by actions

A set of actions that group the messages according to a selected criteria so that they can be presented in a hierarchical layout. The criteria used for grouping can be the severity of the errors (error, warning, info message and so on), the resource name, the description of the message and so on. The **Ungroup all** action removes the grouping rule so that the messages are presented as a continuous list.

Highlight all results in editor

Oxygen XML Author highlights all matches obtained after executing an XPath expression, or performing one of the following operations: **Find All**, **Find in Files**, **Search References**, and **Search Declarations**. Click **Highlight all results in editor** again to turn off highlighting.

> 📋 **Note:** To customize highlighting behavior, *open the Preferences dialog* and go to **Editor** > **Highlights category**. You can do the following customizations:
>
> - set a specific color of the highlights depending on the type of action you make.
> - set a maximum number of highlights that the application displays at any given time.

Remove actions

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

The actions available on the contextual menu are:

Show message

Displays a dialog box with the full error message, which is useful for a long message that does not have enough room to be displayed completely in the view.

Previous message

Moves the selection in the view to the message above the current one.

Next message

Moves the selection in the view to the message below the current one.

Remove selected

Removes selected messages from the view.

Remove all

Removes all messages from the view.

Copy

Copies the information associated with the selected messages:

- the file path of the document that triggered the output message,
- the path of the main file (in case of *validation scenario* it is the path of the file from which the validation starts and which can be different than the validated file),
- error severity (error, warning, info message an so on.),
- name of validating processor,
- name of *validation scenario*,
- the line and column in the file that triggered the message.

Select All

Extends the selection to all the messages from the view.

Print Results ...

Sends the complete list of messages to a printer. For each message the included details are the same as the ones for *the **Copy** action*.

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*.

Group by

A set of actions that group the messages according to a selected criteria so that they can be presented in a hierarchical layout. The criteria used for grouping can be the severity of the errors (error, warning, info message and so on), the resource name, the description of the message and so on. The **Ungroup all** action removes the grouping rule so that the messages are presented as a continuous list.

Ungroup all

Removes the grouping rule set by a **Group by** action so that the errors are presented as a continuous list.

Show Group Columns

Displays/hides columns used as grouping criteria.

Restore Defaults

Restores the column size for each column and the grouping rule that were saved in the user preferences the last time when this view was used (possible in the previous Oxygen XML Author session). If it is the first time when this view is used, the action sets an initial default column size for each column and a grouping rule which is appropriate for the type of messages, for example:

- group the messages by the path of the validated file in case of error messages from a validation action or spelling errors reported by the action **Check Spelling in Files**,
- no grouping rule for the results of *applying an XPath expression*.

Expand All

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

Collapse All

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

Database Perspective

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

This perspective offers database specific support for:

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

The XML capabilities of the databases marked in this list with "Enterprise edition only" are available only in the Enterprise edition of Oxygen XML Author. The non-XML capabilities of any database listed here are available also in the Academic and Professional editions of Oxygen XML Author 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 Author.

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 Author 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 Author 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 Author please *go to the Oxygen XML Author website*.

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

The perspective provides the following functional areas:

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

- 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.

Dockable Views and Editors

All the Oxygen XML Author views available in the Editor Perspective are dockable.

You can drag any view to any margin of another view or editor inside the Oxygen XML Author window. Once you create a layout that suites your needs, you are able to save it from **Window** > **Save Layout...**. Oxygen XML Author creates a layout file containing the preferences of the saved layout. To load a layout, go to **Window** > **Load Layout...**. To reset it, select **Window** > **Reset Layout**.

> **Note:** The **Load Layout...** menu lets you select between the default layout, a predefined layout, or a custom layout. The changes you make using the **Load Layout...** menu are also reflected in the **Perspectives Layout** preferences page.

The changes you make to any layout are preserved between working sessions. Also, changing to a different layout and returning to the previous one does not alter the changes you made to the first layout. The predefined layout files are saved in the *preferences directory* of Oxygen XML Author.

To gain more editing space in the Oxygen XML Author window, click ⊡ **Toggle auto-hide** in any view. This button sets the view in the *auto-hide* state, making it visible only as a vertical tab, at the margins of the Oxygen XML Author window. To display a view in the *auto-hide* state, hover its side-tab with your cursor, or click it to keep the view visible until you click elsewhere. A view can also be set to a floating state, making it independent from the rest of the Oxygen XML Author window.

You can drag the editors and arrange them in any order you like, both horizontally and vertically.

The next figure presents two editors arranged as horizontal tiles. To arrange them vertically, drag one of them on top of the other. In the example below, the personal.xml file was dragged over the personal-schema.xml file. When doing this, a dark grey rectangle marks the rearranged layout.
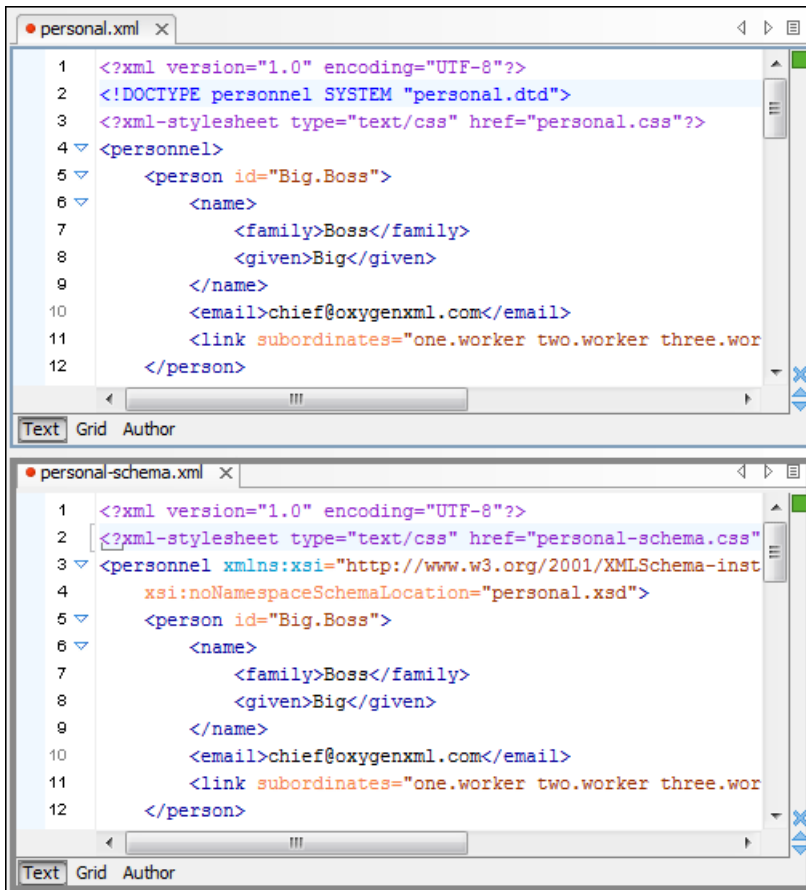
Figure 6: Drag and Drop Editors

You can also tile or stack all open editors, both in the *Editor Perspective* or in the *Database Perspective*, using these actions from the **Window** menu:

Tile Editors Horizontally

Splits the editing area into horizontal tiles, one for each open file.

Tile Editors Vertically

Splits the editing area into vertical tiles, one for each open file.

Stack Editors

The reverse of the **Tile Editors Horizontally/Vertically** actions. Stacks all open editors.

Note:

When tiled, you can still drag and drop the editors, but note that they are docked in the same way as a window/view (instead of just tabs). You are actually rearranging the editor windows, so drag the editor tab and drop it to one of the sides of an editor (left/right/top/bottom). While dragging, you will see the grey dark rectangle aligned to one of the sides of the editor, or around the entire editor window. If you drop it to one of the sides it will dock to that side of the editor. If you drop it when the rectangle is around the entire window of the editor it will get stacked on top of that editor. You can also grab one of the stacked editors and tile it to one of the sides.

To scroll through the tiled editors in the same time, enable the **Synchronous scrolling** action (from the same **Window** menu).

You can divide the editing area vertically and horizontally using the actions available in the **Split** toolbar and in the **Window** menu: **Split horizontally**, **Split vertically**, and **Unsplit**. To maximize or restore the editors, go to **Window** > **Maximize/Restore Editor Area**.

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

To watch our video demonstration about dockable and floating views and editors in Oxygen XML Author, go to *http://oxygenxml.com/demo/Dockable_Views.html*.

Chapter

5

Editing Modes

Topics:

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

- *Text* - this mode presents the source of an XML document.
- *Grid* - this mode displays an XML document as a structured grid of nested tables.
- *Author* - this mode enables you to edit in a WYSIWYG like editor.
- **Design** - this mode is found in the schema editor and represents the schema as a diagram.

Text Editing Mode

The **Text** mode of Oxygen XML Author 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.

The Undo/Redo Actions

The typical undo and redo actions are available in Oxygen XML Author:

Undo Ctrl+Z (Command+Z on OS X) - menu Edit > Undo or the ↩ **Undo toolbar button**

Reverses a maximum of 200 editing actions (configurable in the *Editor preferences page*) to return to the preceding state.

> 📄 **Note:** Complex operations like **Replace All** or **Indent selection** count as single undo events.

Redo Ctrl+Y (Command+Z on OS X) - menu Edit > Redo or the ↪ **Redo toolbar button**

Recreates a maximum of 100 editing actions that were undone by the **Undo** function.

Copying and Pasting Text

The typical copying and pasting actions are available:

Edit > Cut Ctrl+C (Command+C on OS X) or the toolbar button ✂ **Cut**

Removes the current selected node from the document and places it in the clipboard as RTF. All text attributes such as color, font or syntax highlight are preserved when pasting into another application.

Edit > Copy Ctrl+C (Command+C on OS X) or the toolbar button 📋 **Copy**

Places a copy of the current selection in the clipboard as RTF. All text attributes such as color, font or syntax highlight are preserved when pasting into another application.

Edit > Paste Ctrl+V (Command+V on OS X) or the toolbar button 📋 **Paste**

Places the current clipboard content into the document at the cursor position.

Edit > Select All Ctrl+A (Command+A on OS X)

Selects the entire body of the current document, including whitespace preceding the first and following the last character.

Finding and Replacing Text in the Current File

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

You can use a number of advanced views depending on what you need to find in the document you are editing and in even in your entire project. The *Find/Replace dialog* searches through the current project or selected resources and offers a set of options to improve your search. The *Find All Elements/Attributes dialog* searches through the structure of the current document for elements and attributes.

As an alternative to the dedicated search dialogs, use the *Quick Find toolbar*.

The Find/Replace Dialog

To open the **Find / Replace** dialog, do either one of the following:

• go to **Find** > **Find / Replace...**
• click the 🔍 **Find / Replace** toolbar button
• press **Ctrl+F (Command+F on OS X)** keys on your keyboard

You can use the **Find/Replace** dialog to perform the following operations:

- Replace occurrences of target defined in the **Find** area with a new fragment of text defined in **Replace with** area.
- Find all the occurrences of a word or string of characters (that can span over multiple lines) in the document you are editing. This operation also takes into account all the white spaces contained in the fragment you are searching for.

> **Note:** The **Find/Replace** dialog counts the number of occurrences of the text you are searching for and displays it at the bottom of the dialog, above the **Close** button. This number is also displayed in *the Results view*.

The *find* operation works on multiple lines, meaning that a find match can cover characters on more than one line of text. To input multiple-line text in the **Find** and **Replace with** areas, do one of the following:

- press **Ctrl+Enter (Command+Enter on OS X)** keys on your keyboard
- use the **Insert newline** contextual menu action

You can use *Perl-like regular expressions syntax* to define patterns. A content completion assistant window is available in the **Find** and **Replace with** areas to help you edit regular expressions. It is activated every time you type \(backslash key) or on demand if you press **Ctrl+Space (Command+Space on OS X)** keys on your keyboard.

The *replace* operation can bind regular expression capturing groups ($1, $2, etc.) from the find pattern.

> To replace the `tag-name` start tag and its attributes with the `new-tag-name` tag use as **Find** the expression *<tag-name(\s+)(.*)>* and as **Replace with** the expression *<new-tag-name$1$2>*.

The dialog contains the following options:

- **Find** - The target character string to search for. You can search for Unicode characters specified in the \uNNNN format. Also, hexadecimal notation (\xNNNN) and octal notation (\0NNNN) can be used. In this case you have to select the **Regular expression** option. For example, to search for a space character you can use the \u0020 code.
- **Replace with** - The character string with which to replace the target. The string for replace can be on a line or on multiple lines. It can contain Perl notation capturing groups, only if the search expression is a regular expression and the **Regular expression** option is selected.

  > **Note:** Some regular expressions can block indefinitely the application. If the execution of the regular expression does not end in about 5 seconds, the application displays a dialog that allows you to interrupt the operation.

  > **Note:** Special characters like *newline* and *tab* can be inserted in the **Find** and **Replace with** text boxes using dedicated actions in the contextual menu (**Insert newline** and **Insert tab**).

  Unicode characters in the \uNNNN format can also be used in the **Replace with** area.

- The history buttons ⟳ - Contain lists of the last find and replace expressions. Use the ✳ **Clear history** action from the bottom of the lists to remove these expressions.
- **XPath** - The XPath 2.0 expression you input in this combo is used for restricting the search scope.

  > **Note:** The **Content Completion Assistant** helps you input XPath expressions, valid in the current context.

- **Direction** - Specifies if the search direction is from current position to end of file (**Forward**) or to start of file (**Backward**).
- **Scope** - In **Author** mode, the search scope is restricted to the entire document only.
- **Find** - Executes a find operation for the next occurrence of the target. It stops after highlighting the find match in the editor panel.
- **Replace** - Executes a replace operation for the target followed by a find operation for the next occurrence.
- **Find All** - Executes a find operation and displays all results in the *Results view*. The results are *displayed in the Results view*.
- **Replace All** - Executes a replace operation in the entire scope of the document.
- **Replace to End** - Executes a replace operation starting from current target until the end of the document, in the direction specified by the current selection of the **Direction** switch (**Forward** or **Backward**).
- **Case sensitive** - When checked, the search operation follows the exact letter case of the **Find**.

- **Whole words only** - Only entire occurrences of a word are included in the search operation.
- **Incremental** - The search operation is started every time you type or delete a letter in the **Find** text box.
- **Regular expression** - When this option is enabled, you can use regular expressions in *Perl-like regular expressions syntax* to look for specific pieces of text.
- **Dot matches all** - A dot used in a regular expression matches also end of line characters.
- **Wrap around** - When the end of the document is reached, the search operation is continued from the start of the document, until its entire content is covered.
- **Enable XML search options** - Provides access to a set of options that allow you to search specific XML component types when editing in **Text** mode:

    - **Element names** - Only the element names are included in the search operation which ignores XML-tag notations ('<', '/', '>'), attributes or white-spaces.
    - **Element contents** - Search in the text content of XML elements.
    - **Attribute names** - Only the attribute names are included in the search operation, without the leading or trailing white-spaces.
    - **Attribute values** - Only the attribute values are included in the search operation, without single quotes(') or double quotes(").
    - **Comments** - Only the content of comments is included in the search operation, excluding the XML comment delimiters ('<!--', '-->').
    - Processing Instructions (**PIs**) - Only the content are searched, skipping '<?', '?>'. e. g.: <?processing instruction?>
    - **CDATA** - Searches inside content of CDATA sections.
    - **DOCTYPE** - Searches inside content of DOCTYPE sections.
    - **Entities** - Only the entity names are searched.

    The two buttons **Select All** and **Deselect All** allow a simple activation and deactivation of all types of XML components.

    > **Note:** Even if you enable all options of the **Enable XML search options** section, the search is still XML-aware. If you want to perform the search over the entire file content, disable **Enable XML search options**.

- **Find All Elements / Attributes ...** - While editing in **Author** mode, press the **Find All Elements / Attributes ...** to extend the search scope to XML-specific markup (names and values of both attributes and elements).

The Find All Elements / Attributes Dialog

To open the **Find All Elements / Attributes** dialog, go to **Find** > **Find All Elements...**(**Ctrl+Shift+E (Command+Shift+E on OS X)**) or from the shortcut **Find All Elements / Attributes** that is available in *the Find / Replace dialog* . It assists you in defining XML elements / attributes search operations on the current document.
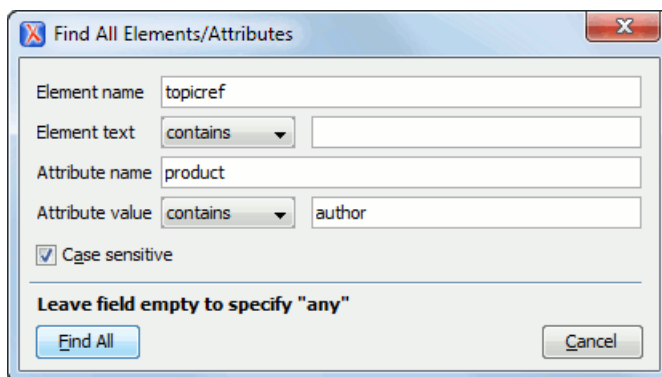
Figure 7: Find All Elements / Attributes dialog

The dialog can perform the following actions:

- Find all the elements with a specified name;

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

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

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

The following fields are available in the dialog:

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

The Quick Find Toolbar

A reduced version of *the Find / Replace dialog* is available as *a dockable toolbar*. To display it press the **Alt+Shift+F (Command+Shift+F on OS X)** key combination or invoke the **File** > ⌕ **Quick Find...** action. The toolbar is displayed by default at the bottom of the Oxygen XML Author window, above the status bar, but can be changed at any time by dragging (and docking) it to a different location. To hide the toolbar, press the **ESC** key.

All matches are highlighted in the current editor.

The toolbar offers the following controls:

- A search input box where you insert the text you want to search for. The input box keeps a history of the last used search texts. The background color of the input box turns red when no match is found.
- **Next** and **Previous** buttons. They allow you to advance to the next or previous match.
- **All** button. Highlights in the current document all matches of the search string.
- **Incremental** check box. The search operation is started every time you type or delete a character in the search input box.
- **Case sensitive** check box. When selected, the search operation follows the exact letter case of the search text.
- Shortcuts to 🔍 *Find / Replace* and 🔍 *Find / Replace in Files* dialogs.

Keyboard Shortcuts for Finding the Next and Previous Match

Navigating from one match to the next or previous one is very easy to perform using the **F3** and **Shift+F3** keyboard shortcuts. They are useful to quickly repeat the last find action performed in *the Find / Replace dialog*, taking into account the same find options.

Regular Expressions Syntax

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

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

Other notable difference:

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

Finding and Replacing Text in Multiple Files

To open the **Find / Replace in Files** dialog do one of the following:

- go to **Find** > **Find / Replace in Files...** ;
- click the [icon] **Find / Replace in Files** toolbar button;
- go to the contextual menu of the **DITA Maps Manager** view;
- go to the contextual menu of the **Project** view;
- go to the contextual menu of the **Data Source Explorer** view for Documentum xDb (X-Hive/DB), eXist and WebDAV connections. This action is available for Documentum (CMS), but lacks the *replace* feature.

The operation works on both local and remote files from an (S)FTP, WebDAV or *CMS* server.

It enables you to define *Search for* or *Search for and Replace* operations across a number of files. You can use *Perl-like regular expression syntax* to match patterns in text content. The *replace* operation can bind regular expression capturing groups ($1, $2, etc.) from the find pattern.

> To replace the tag-name start tag and its attributes with the new-tag-name tag use as **Text to find** the expression *<tag-name(\s+)(.*)>* and as **Replace with** the expression *<new-tag-name$1$2>*.

The encoding used to read and write the files is detected from the XML header or from the BOM. If a file does not have an XML header or BOM Oxygen XML Author uses by default the UTF-8 encoding for files of type XML, that is for files with one of the extensions: .xml, .xsl, .fo, .xsd, .rng, .nvdl, .sch, .wsdl or *an extension associated with the XML editor type*. For the other files it uses *the encoding configured for non-XML files.*

You can cancel a long operation at any time by pressing the **Cancel** button of the progress dialog displayed when the operation is executed.

Because the content of read-only files cannot be modified, the **Replace** operation is not processing those files. For every such file, a warning message is displayed in the message panel.
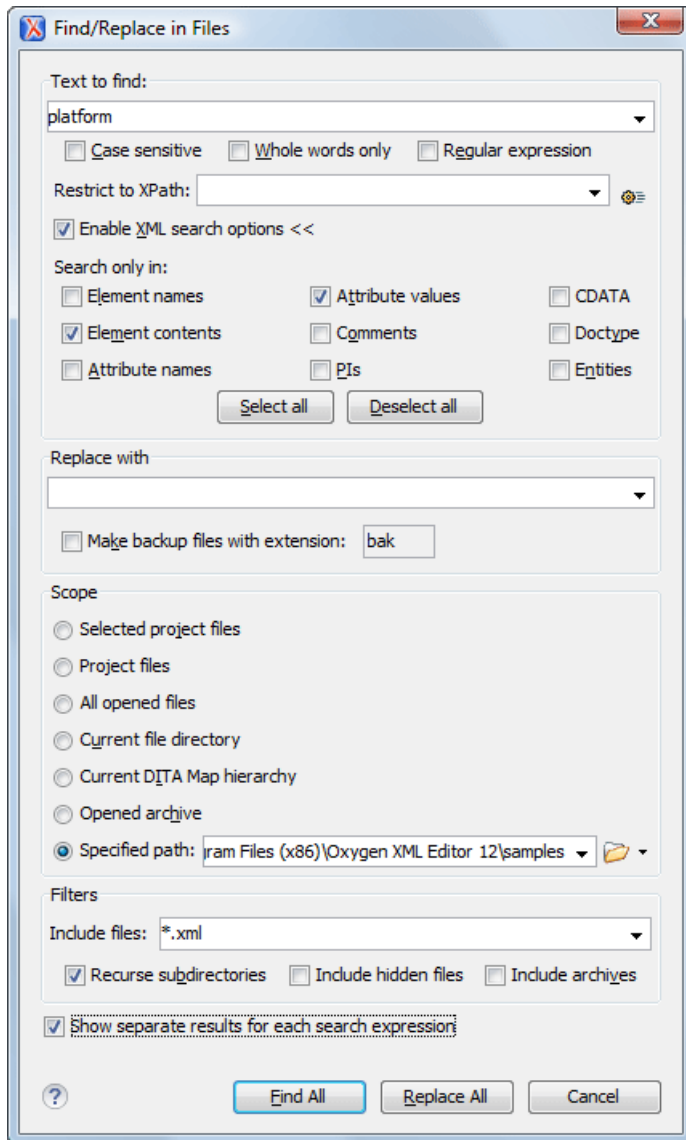
Figure 8: Find / Replace in Files

The dialog contains the following options:

- **Find** - The target character string to search for. You can search for Unicode characters specified in the `\uNNNN` format. Also, hexadecimal notation (`\xNNNN`) and octal notation (`\0NNNN`) can be used. In this case you have to select the **Regular expression** option. For example, to search for a space character you can use the `\u0020` code.
- **Case sensitive** - When checked, the search operation follows the exact letter case of the **Find**.
- **Whole words only** - Only entire occurrences of a word are included in the search operation.
- **Regular expression** - When this option is enabled, you can use regular expressions in *Perl-like regular expressions syntax* to look for specific pieces of text.
- **XPath** - The XPath 2.0 expression you input in this combo is used for restricting the search scope.

  > **Note:** The **Content Completion Assistant** helps you input XPath expressions, valid in the current context.

- **Enable XML search options** - Provides access to a set of options that allow you to search specific XML component types when editing in **Text** mode:
  - **Element names** - Only the element names are included in the search operation which ignores XML-tag notations ('<', '/', '>'), attributes or white-spaces.
  - **Element contents** - Search in the text content of XML elements.

- **Attribute names** - Only the attribute names are included in the search operation, without the leading or trailing white-spaces.
- **Attribute values** - Only the attribute values are included in the search operation, without single quotes(') or double quotes(").
- **Comments** - Only the content of comments is included in the search operation, excluding the XML comment delimiters ('<!--', '-->').
- Processing Instructions (**PIs**) - Only the content are searched, skipping '<?', '?>'. e. g.: <?<u>processing instruction</u>?>
- **CDATA** - Searches inside content of CDATA sections.
- **DOCTYPE** - Searches inside content of DOCTYPE sections.
- **Entities** - Only the entity names are searched.

The two buttons **Select All** and **Deselect All** allow a simple activation and deactivation of all types of XML components.

> **Note:** Even if you enable all options of the **Enable XML search options** section, the search is still XML-aware. If you want to perform the search over the entire file content, disable **Enable XML search options**.

- **Replace with** - The character string with which to replace the target. It may contain regular expression group markers if the search expression is a regular expression and the **Regular expression** checkbox is checked.
- **Make backup files with extension** - In the replace process Oxygen XML Author makes backup files of the modified files. The default extension is `.bak` but you can change the extension as you prefer.
- **Selected project files** - Searches only in the selected files of the current opened project. Not displayed when dialog is started from contextual menu of *DITA Maps view* and *Archive Browser* view.
- **Project files** - Searches in all files from the current project. Not displayed when dialog is started from contextual menu of *DITA Maps view* and *Archive Browser* view.
- **All opened files** - Searches in all files opened in Oxygen XML Author (regular files or DITA Maps). You are prompted to save all modified files before any operation is performed. Not displayed when dialog is started from contextual menu of *DITA Maps view* and *Archive Browser* view.
- **Current file directory** - The search is done in the directory of the file opened in the current editor panel. If there is no opened file, this option is disabled in the dialog. Not displayed when dialog is started from contextual menu of *DITA Maps view* and *Archive Browser* view.
- **Current DITA Map hierarchy** - The search is done in all maps and topics referenced by the current edited DITA Map. If **Recurse referenced maps** option is checked, the references from the maps referenced in the main DITA map are also searched in. If there is no opened DITA Map, this option is disabled in the dialog. You are prompted to save all modified files before any operation is performed. Displayed only when dialog is opened from *DITA Maps view*.
- **Opened archive** - The search is done in archive opened in *Archive Browser* view. Displayed only when dialog is opened from *Archive Browser* view.
- **Specified path** - Chooses the search path.
- **Include files** - Narrows the scope of the operation only to the files that match the given filters.
- **Recurse subdirectories** - When checked, the search is performed recursively in the sub directories found in the specified directory path.
- **Include hidden files** - When checked, the search is performed also in the hidden files.
- **Include archives** - When checked, the search is also done in all individual file entries from all supported ZIP-type archives.
- **Show separate results for each search expression** - When checked, the application opens a new tab to display the result of each new search expression. When the option is unchecked, the search results are displayed in the *Find in Files* tab, replacing any previous search results.
- **Find All** - Executes a find operation and returns the result list to the message panel. The results are *displayed in a view* that allows grouping the results as a tree with two levels.
- **Replace All** - Replaces all occurrences of the target contained in the specified files.

When you replace a fragment of text, Oxygen XML Author provides a preview of the changes you make. The **Preview** dialog is divided in two sections. The first section presents a list of all the documents containing the fragment of text

you want to modify. The second section offers a view of the original file and a view of the final result. It also allows you to highlight all changes using the vertical bar from the right side of the view. The **Next change** and **Previous change** buttons allow you to navigate through the changes displayed in the **Preview** dialog.

> ◆ **Caution:** Use this option with caution. Global search and replace across all project files does not open the files containing the targets, nor does it prompt on a per occurrence basis, to confirm that a replace operation must be performed. As the operation simply matches the string defined in the find field, this may result in replacement of matching strings that were not originally intended to be replaced.

Changing the Font Size

The font size of the editor panel can be changed with the following actions:

- **Document** > **Font size** > **Increase editor font** (<u>**Ctrl+NumPad+"+" (Command+NumPad+"+" on OS X)**</u>) - increases the font size with one point for each execution of the action;
- **Document** > **Font size** > **Decrease editor font** (<u>**Ctrl+NumPad+"-" (Command+NumPad+"-" on OS X)**</u>) - decreases the font size with one point for each execution of the action;
- **Document** > **Font size** > **Normal editor font** (<u>**Ctrl+0 (Command+0 on OS X)**</u>) - resets the font size to *the value of the editor font set in Preferences*.

Word/Line Editor Actions

The **Text** editor implements the following actions:

<u>**Ctrl+Delete (Command+Delete on OS X)**</u>

Deletes the next word.

<u>**Ctrl+Backspace (Command+Backspace on OS X)**</u>

Deletes the previous word.

<u>**Ctrl+W (Command+W on OS X)**</u>

Cuts the previous word.

<u>**Ctrl+K (Command+K on OS X)**</u>

Cuts to end of line.

Dragging and Dropping the Selected Text

To move a whole region of text to other location in the same edited document just select the text, drag the selection by holding down the left mouse button and drop it to the target location.

Inserting a File at Caret Position

The action from menu **Document** > **File** > **Insert file...** inserts in the current document the content of the file with the file path at the current position of the caret.

Opening the File at Caret in System Application

The **Document** > **File** > **Open File at Caret in System Application** action opens the file (identified by its link) or web page (identified by a web link) found at caret position. The target is open in the default system application associated with that file type.

Opening the File at Caret Position

The action from menu **Document** > **File** > **Open File at Caret** opens in a new panel the file with the file path at the 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** button which starts the **New document** wizard. This allows you to choose the type or the template for the file. If the action succeeds, the file is created with the referred location and name and is opened in a new editor panel. This is useful when you decide first on the file name and after that you want to create it in the exact location specified at the current caret position.

Printing a File

Printing is supported in **Text**, **Author,** and **Grid** modes of the XML editor panel. The action from menu **File** > **Print** > **Ctrl+P (Command+P on OS X)** displays the **Page Setup** dialog used for defining the page size and orientation properties for printing.

A **Print Preview** action is available in the **File** menu. It allows you to manage the format of the printed document.

> **Note:** If you are printing in the Author visual editing mode to change the styling of the document in the printed output you can use the *CSS print media type*.
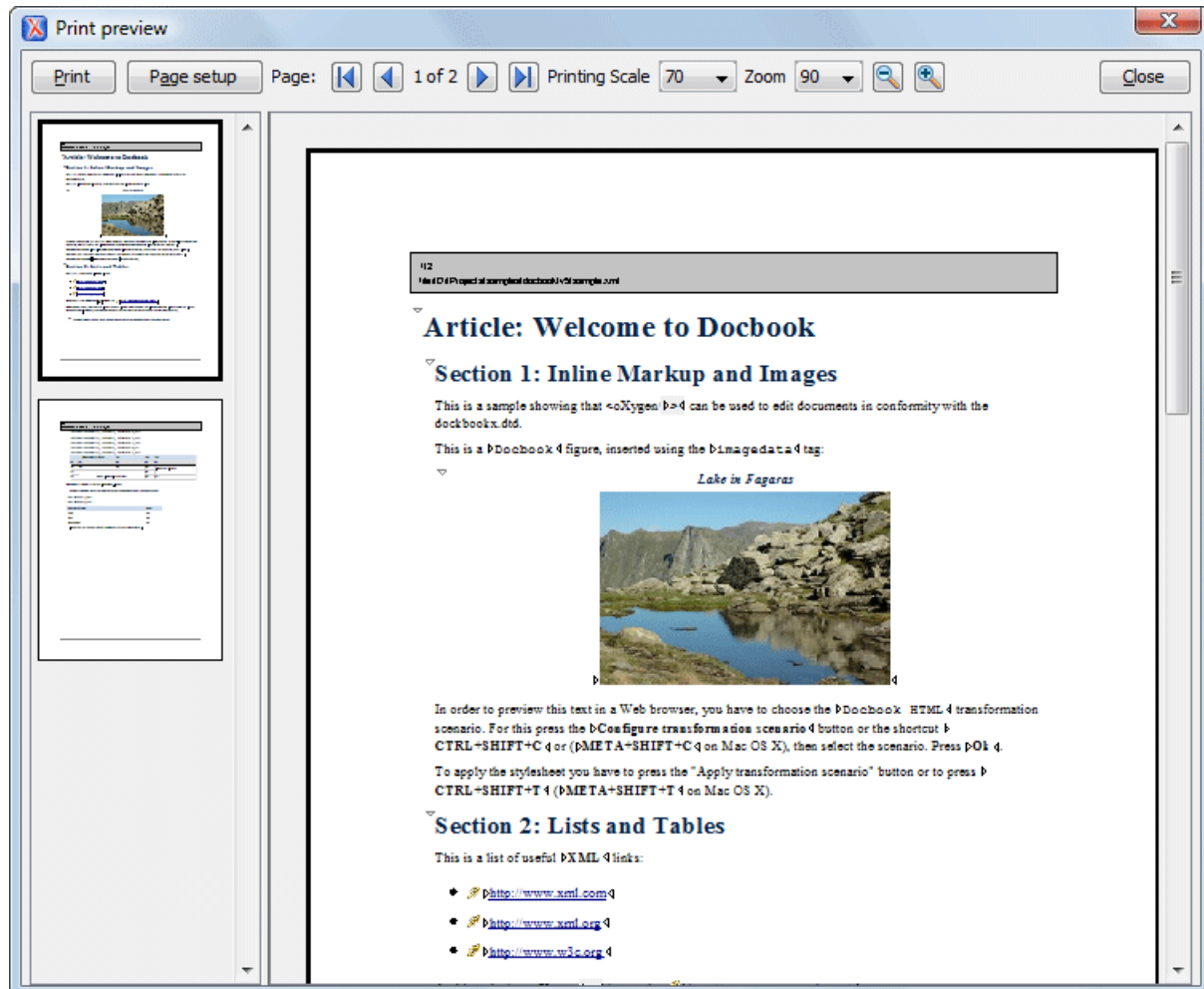
Figure 9: Print Preview Dialog

The main window is split in three sections:

- **Preview area** - Displays the formatted document page as it will appear on printed paper.
- **Left stripe** - The left-hand side stripe which displays a list of thumbnail pages. Clicking any of them displays the page content in the main preview area.
- **Toolbar** - The toolbar top area which contains controls for printing, page settings, page navigation, print scaling, and zoom.

> **Note:** If your document is open in Author mode and contains *Track Changes*, you can print (or print preview) a copy of the document as if all changes have been accepted by switching the *Track Changes Visualization Modes* to **View Final**.

Grid Editing Mode

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

In case you are a non-technical user, you are able to modify the text content of the edited document without working with the XML tags directly. You can expand and collapse the tables using the mouse cursor and also display or hide the elements of the document as nested. The document structure can also be changed easily with drag and drop operations on the grid components. To zoom in and out, use **Ctrl+"+" (Command+"+" on OS X)**, **Ctrl+"-" (Command+"-" on OS X)**, **Ctrl+0 (Command+0 on OS X)** or **Ctrl+Scroll Forward (Command+Scroll Forward on OS X)** /**Ctrl+Scroll Backwards (Command+Scroll Backwards on OS X)**.
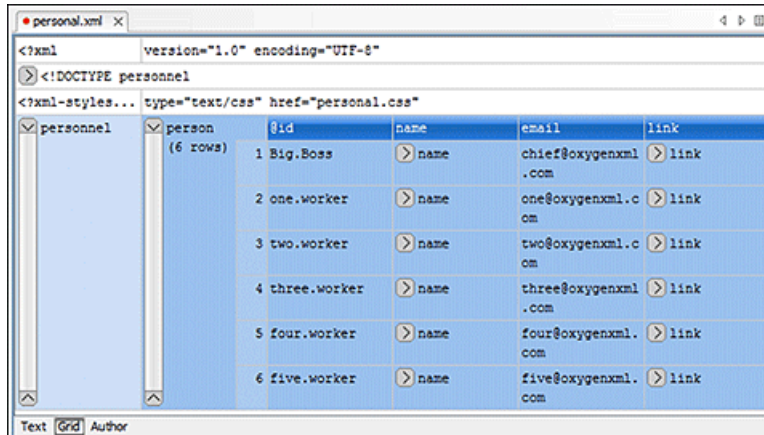
Figure 10: The Grid Editor

To switch back from the **Grid** mode to the **Text** or **Author** mode, use the **Text** and **Grid** buttons from the bottom of the editor. You are also able to perform this switch from **Document** > **Edit Mode** > **Grid** and **Document** > **Edit Mode** > **Text**.

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

To display the content completion pop-up, you have to start editing (for example, double click a cell). Pressing **Ctrl+Space (Command+Space on OS X)** on your keyboard also displays the pop-up.
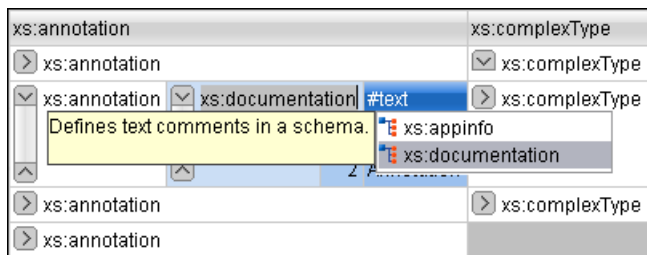
Figure 11: Content Completion in Grid Editor

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

Layouts: Grid and Tree

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

Figure 12: Grid Layout

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

Figure 13: Tree Layout

To switch between the two modes, go to **Document** > **Grid Layout** > **Grid mode/Tree mode**.

Grid Move Navigation

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

Table 2: Shortcuts in the Grid Mode

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

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

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

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

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

Expand All

Expands the selection and all its children.

Collapse All

Collapses the selection and all its children.

Expand Children

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

Collapse Children

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

Collapse Others

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

Specific Grid Actions

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

Sorting a Table Column

You can sort the table by a specific column. The sorting can be either ascending or descending. The icons for this pair of actions are ![A-Z ascending] and ![Z-A descending] .

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

Inserting a Row in a Table

You can add a new row using the **Copy**/**Paste** row operation, or by selecting ![icon] **Insert row** from the **Table** contextual menu.

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

Inserting a Column in a Table

You can insert a column after the selected one, using the ![icon] **Insert column** action from the **Table** contextual menu.

Clearing the Content of a Column

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

Adding Nodes

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

The sub-menus containing detailed actions are:

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

Duplicating Nodes

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

Refresh Layout

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

Start Editing a Cell Value

You can simply press **(Enter)** after you have selected the grid cell or you can use the **Start Editing** action found in the **Document** > **Grid Edit** menu.

Stop Editing a Cell Value

You stop editing a cell value when you press **(Enter)** or use the **End Editing** action found in the **Document** > **Grid Edit** menu.

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

Drag and Drop in the Grid Editor

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

With drag and drop you can:

- copy or move a set of nodes;
- change the order of columns in the tables;
- move the rows from the tables.

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

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

Copy and Paste in the Grid Editor

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

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

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

The same action can be found in the menu **Document** > **Grid Edit** > **Paste as Child**.

The nodes copied from the **Grid** editor can also be pasted into the **Text** editor or other applications. When copying from the **Grid** into the **Text** editor or other text based applications, the inserted string represents the nodes serialization. The nodes from tables can be copied using HTML or RTF in table format. The resulting cells contain only the concatenated values of the text nodes.
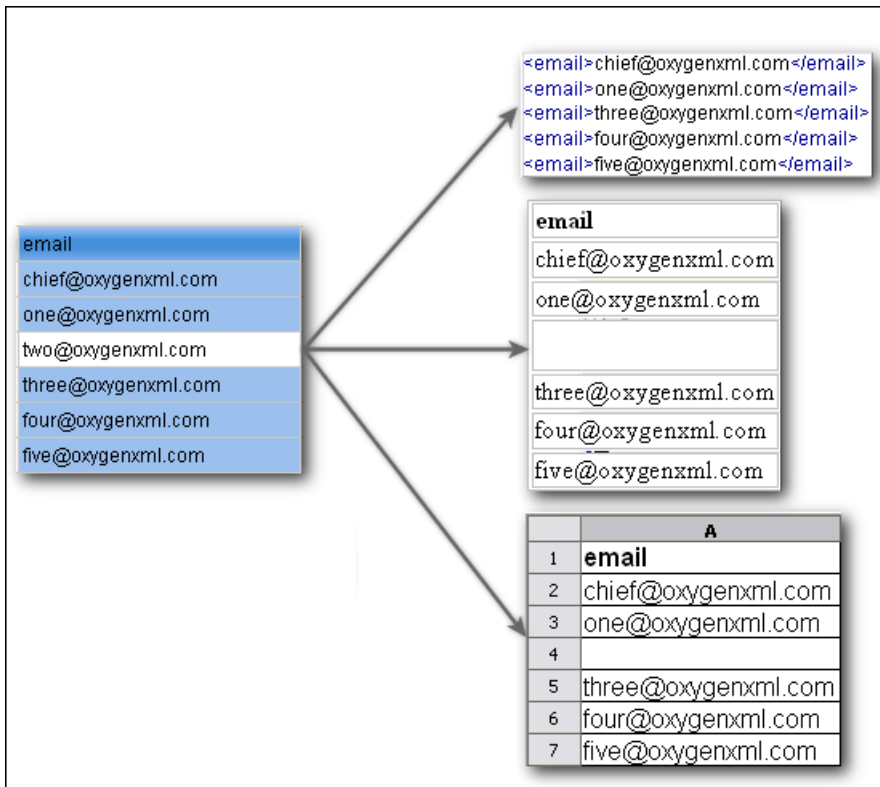
Figure 14: Copying from Grid to Other Editors

In the **Grid** editor you can paste well-formed XML content or tab separated values from other editors. If you paste XML content, the result will be the insertion of the nodes obtained by parsing this content.
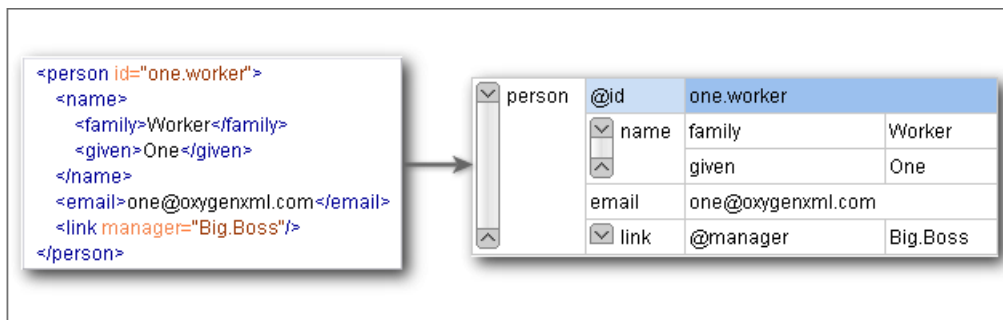
Figure 15: Copying XML Data into Grid

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.

Figure 16: Copying Tab Separated Values into Grid

Bidirectional Text Support in Grid Mode

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

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

Figure 17: Default left to right text orientation

Figure 18: Right to left text orientation

Author Editing Mode

This chapter presents the WYSIWYG like editor, called **Author** editor, targeted to content authors.

Tagless XML Authoring

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

This tagless editor is available as the **Author** mode of the XML editor. To enter the **Author** mode, click the **Author** button at the bottom of the editing area. The **Author** mode renders the content of the XML document visually based on a CSS stylesheet associated with the document. Many of the actions and features available in **Text** mode are also available in **Author** mode.
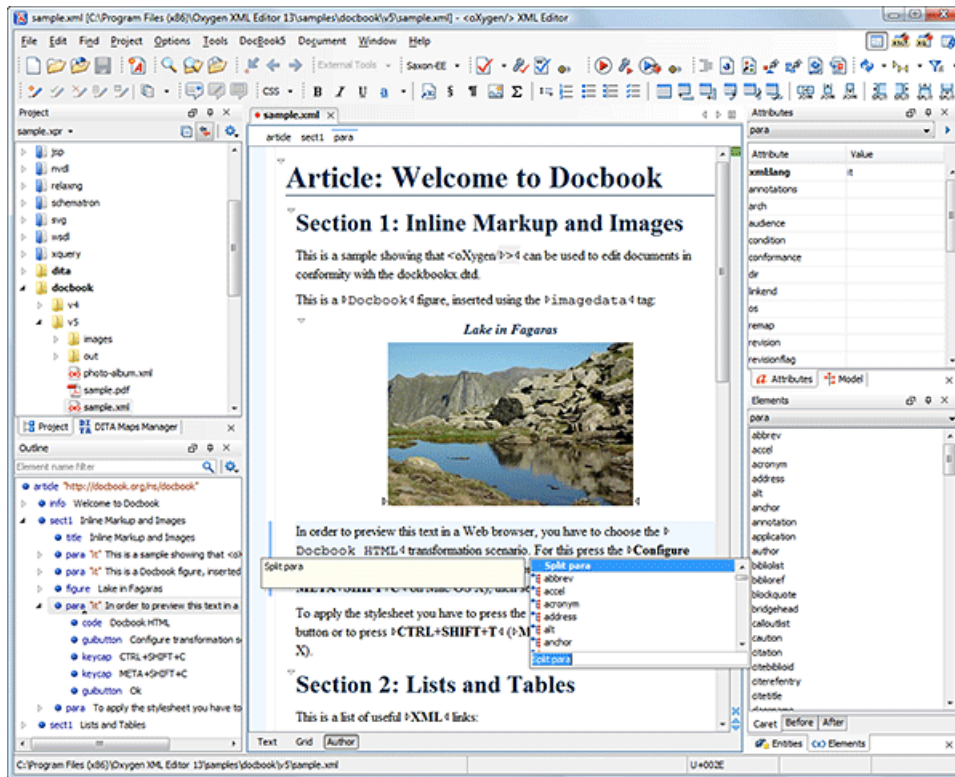
Figure 19: Author Editing Mode

Associating a CSS with an XML Document

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

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

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

Author Mode User Roles

There are two main types of users of the **Author** mode: *framework developers* and *content authors*. A *framework developer* is a technical person with advanced XML knowledge who defines the framework for authoring XML documents in the tagless editor. Once the framework is created or edited by the developer, it is distributed as a deliverable component ready to plug into the application to the content authors. A *content author* does not need to have advanced knowledge about XML tags or operations like validation of XML documents or applying an XPath expression to an XML document.

The author just plugs the framework set-up by the developer into the application and starts editing the content of XML documents without editing the XML tags directly.

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

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

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

To watch our video demonstration about the basic functionality of the **Author** mode, go to *http://oxygenxml.com/demo/WYSIWYG_XML_Editing.html*.

General Author Presentation

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

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

The syntax of such a processing instruction is:

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

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

When the document has no CSS association or the referred stylesheet files cannot be loaded, a default one is used. A warning message is also displayed at the beginning of the document presenting the reason why the CSS cannot be loaded.
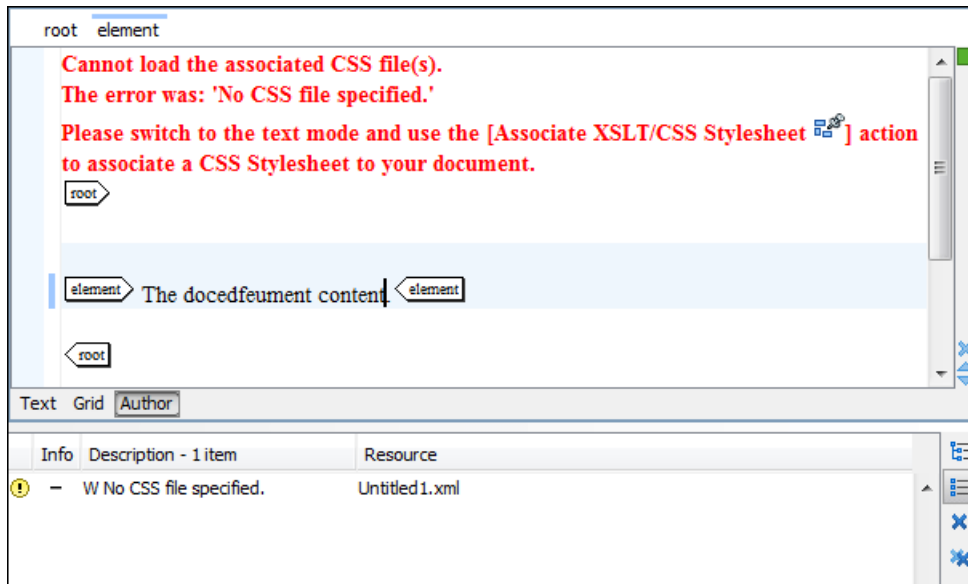
Figure 20: Document with no CSS association default rendering

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*.
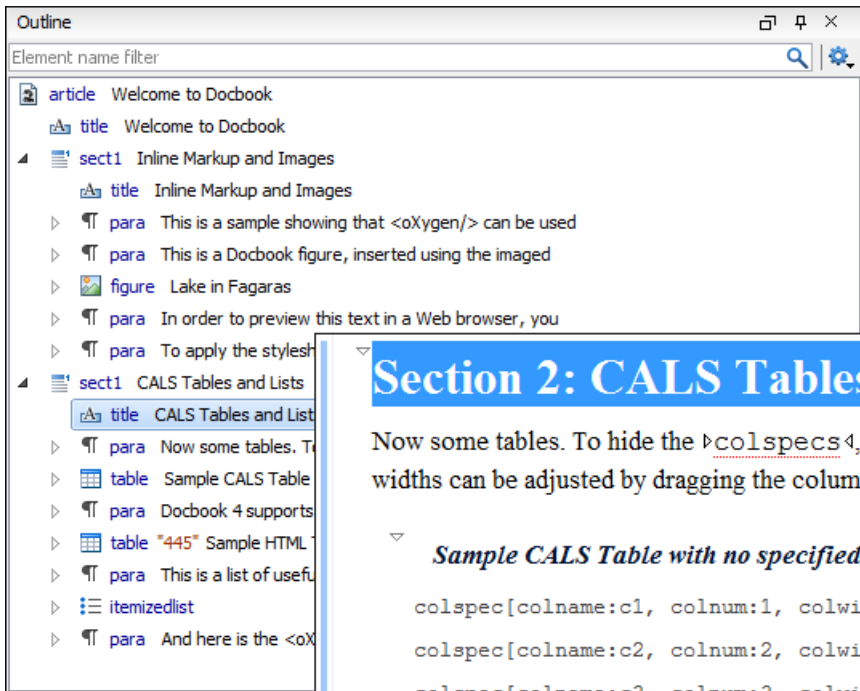
Figure 21: The Outline View

XML Document Overview

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

The **Outline** view allows you to:

- insert or delete nodes using pop-up menu actions;
- move elements by dragging them to a new position in the tree structure;
- highlight elements in the **Author** editor area.

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

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

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

Modification Follow-up

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

Document Structure Change

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

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

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

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

Outline Filters

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

Flat presentation mode of the filtered results

When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions

Show/hide comments and processing instructions in the **Outline** view.

Show element name

Show/hide element name.

T **Show text**

Show/hide additional text content for the displayed elements.

Show attributes

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

Configure displayed attributes

Displays the *XML Structured Outline preferences page*.

The 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

Opens a dialog box that allow you to see and edit the attributes of the selected node.

Append child, Insert before and Insert after

Submenus that allow you 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.

Cut, Copy and Paste

Usual text manipulation actions. The content retains text attributes like color and font.

Paste

Pastes the clipboard content in the currently selected node. The **Paste Before** and **Paste After** variants of the *Paste* action allow you to paste the clipboard content before and after the currently selected node.

Delete

Deletes the currently selected node.

Toggle Comment

If the currently selected element is not commented, this action encloses it in an XML comment. Otherwise, it removes the comment.

Rename Element

Allows you to rename the selected element. Alternatively, you can choose to rename all its siblings with the same name or all elements with the same name in the entire document.

Expand More / Collapse All

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.
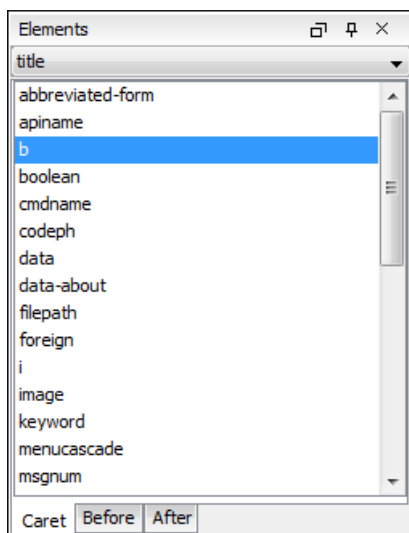
Figure 22: The Elements View

Three tabs present information relative to the caret location:

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

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

Attributes View

The **Attributes** view presents all the attributes of the current element determined by the schema of the document. It allows you to insert attributes in the current element or change the value of the attributes already inserted. The attributes are rendered differently depending on their state:

- the names of the attributes with a specified value are rendered with a bold font, and their value with a plain font;

> **Note:** The names of the attributes with an empty string value are also rendered bold.

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

Double-click a cell in the **Value** column to edit the value of the corresponding attribute. In case the possible values of the attribute are specified as list in the schema of the edited document, the **Value** column acts as a combo box that allows you to insert the values in the document.

You can sort the attributes table by clicking the **Attribute** column header. The table contents can be sorted as follows:

- by attribute name in ascending order;
- by attribute name in descending order;
- custom order, where the used attributes are displayed at the beginning of the table sorted in ascending order, followed by the rest of the allowed elements sorted in ascending order.
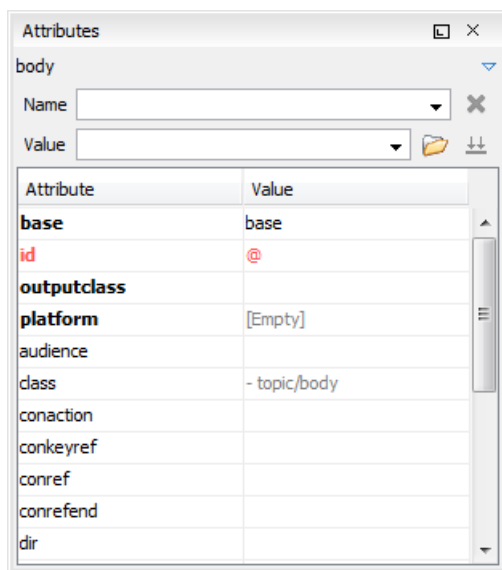
Figure 23: The Attributes View

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

The following actions are available in the contextual menu:

Add

Allows you to insert a new attribute. Adding an attribute that is not in the list of all defined attributes is not possible when the *Allow only insertion of valid elements and attributes* schema aware option is enabled.

Set empty value

Specifies the current attribute value as empty.

Remove

Removes the attribute (action available only if the attribute is specified). You can invoke this action by pressing the **(Delete)** or **(Backspace)** keys;

Copy

Copies the `attrName="attrValue"` pair to the clipboard. The `attrValue` can be:

- the value of the attribute;
- the value of the default attribute, in case the attribute does not appear in the edited document;
- empty, in case the attribute does not appear in the edited document and has no default value set.

Paste

> This action is available in the contextual menu of the **Attributes** view, in the **Text** and **Author** modes. Depending on the content of the clipboard, the following cases are possible:
>
> - in case the clipboard contains an attribute and its value, both of them are introduced in the **Attributes** view. The attribute is selected and its value is changed if they exist in the **Attributes** view;
> - in case the clipboard contains an attribute name with an empty value, the attribute is introduced in the **Attributes** view and you can start editing it. The attribute is selected and you can start editing it if it exists in the **Attributes** view;
> - in case the clipboard only contains text, the value of the selected attribute is modified.

In-place Attributes Editor

To edit in-place the attributes of an XML element, do one of the following:

- completely select the element or place the caret inside it and then press the **Alt+Enter** keyboard shortcut;
- double click any named start tag when the document is edited in one of the following *display modes*.: 📑 **Full Tags with Attributes**, 📑 **Full Tags**, ▷ **Block Tags**, ▷ **Inline Tags**.

This shortcut pops up a small window with the same content as the **Attributes** view. The default form of the pop-up window presents the **Name** and **Value** fields, with the list of all the possible attributes collapsed.
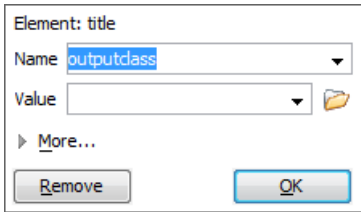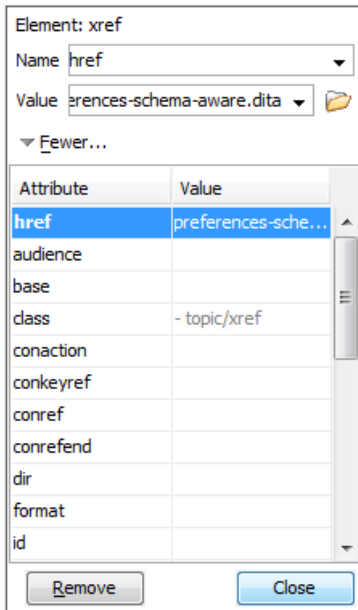
Figure 24: Edit attributes in place

The small right arrow button expands the list of possible attributes allowed by the schema of the document as in the **Attributes** view.
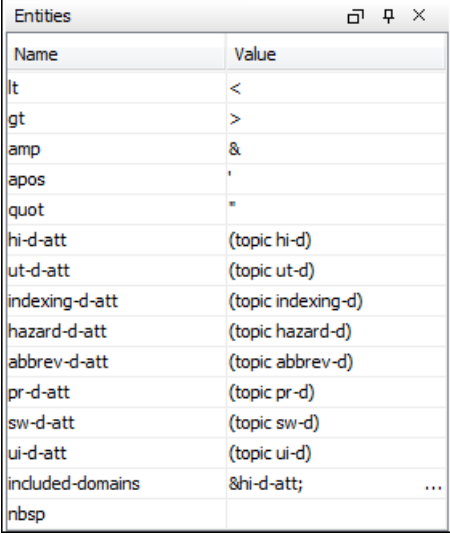
Figure 25: Edit attributes in place - full version

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.

Entities	⊡ ᵗ ×
Name	Value
lt	<
gt	>
amp	&
apos	'
quot	"
hi-d-att	(topic hi-d)
ut-d-att	(topic ut-d)
indexing-d-att	(topic indexing-d)
hazard-d-att	(topic hazard-d)
abbrev-d-att	(topic abbrev-d)
pr-d-att	(topic pr-d)
sw-d-att	(topic sw-d)
ui-d-att	(topic ui-d)
included-domains	&hi-d-att; ...
nbsp	

Figure 26: The Entities View

The Author Editor

This section explains the features of the tag-less WYSIWYG-like editor for XML documents.

Navigating the Document Content

Using the Keyboard

Oxygen XML Author allows you to quickly navigate through a document using **Tab** to go to the next XML node and **Shift+Tab** to go to the previous one. The caret is moved to the next / previous editable position. When the caret is positioned in a space preserve element, press a key on your keyboard and then use **Tab** to arrange the text. You can also arrange the text using **Tab** if you position the cursor in a space preserve element using your mouse. In case you encounter a space preserve element when you navigate through a document and you press no other key, the next **Tab** continues the navigation.

To navigate one word forward or backwards, use **Ctrl+Right Arrow (Command+Right Arrow on OS X)**, and **Ctrl+Left Arrow (Command+Left Arrow on OS X)**, respectively. Entities and hidden elements are skipped. To position the cursor at the beginning or at the end of the document you can use **Ctrl+Home (Command+Home on OS X)**, and **Ctrl+End (Command+End on OS X)** respectively.

Using the Navigation Toolbar

The locations of selected text are stored in an internal list which allows you to navigate between them with the ⬅ **Back** (**Ctrl+Alt+[(Command+Alt+[on OS X)**) and ➡ **Forward** (**Ctrl +Alt+] (Command+Alt+] on OS X)**) buttons from the **Navigation** toolbar. The 💾 **Last Modification** (**Ctrl+Alt+G (Command+Alt+G on OS X)**) button automatically takes you to the latest edited text.

Using the Breadcrumb Helpers

A top stripe called *breadcrumb* indicates the path from document root to the current element.

Figure 27: 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. Also, each element provides a contextual popup menu with access to the following actions:

- **Edit Attributes** action which opens the *in-place attributes editor*.
- The **Edit Profiling Attributes** action allows you to select the *profiling attributes* that apply to a certain element.
- The **Append child**, **Insert before** and **Insert after** submenus of the 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 selecting an element name from the popup menu offered by *the content completion assistant.* The **Insert before** and **Insert after** submenus list the elements which are allowed by the schema associated with the current document as siblings of the current element inserted immediately before respectively after the current element.
- The **Cut**, **Copy**, **Paste** and **Delete** items of the popup menu execute *the same actions as the **Edit** menu items with the same name* on the elements currently selected in the stripe. The **Cut** and **Copy** operations (like the `display:block` property or the tabular format of the data from a set of table cells) preserve the styles of the copied content. The **Paste before**, **Paste after** and **Paste as Child** actions allow the user to insert an well-formed element before, after or as a child of the currently selected element.
- The **Toggle Comment** item of the **Outline** tree popup menu encloses the currently selected element of the top stripe in an XML comment, if the element is not commented, or removes the comment if it is commented.
- Using the **Rename Element** action the selected element and the elements that have the same name as the current element can be renamed according with the options from the **Rename** dialog.

> **Tip:** The tag names displayed in the breadcrumb can be customized with an Author extension class that implements `AuthorBreadCrumbCustomizer`. See the *Oxygen SDK* for more details.

Using the Folding Support

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

Toggle Fold

Toggles the state of the current fold.

Close Other Folds (Ctrl+NumPad +/ (Command+NumPad+/ on OS X))

Folds all the elements except the current element.

Collapse Child Folds (Ctrl+NumPad + . (Command+NumPad+. on OS X))

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 + * (Command+NumPad+* on OS X))

Unfolds all elements in the current document.

Using the Linking Support

When working on a suite of documents that refer to one another (references, external entities, XInclude, DITA conref, etc), the **linking support** is useful for navigating between the documents. In the predefined customizations that are bundled with Oxygen XML Author 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 the link to open the referred resource in an editor. The same effect can

be obtained by using the action **Document** > **File** > **Open file at caret (<u>Ctrl+Enter (Command+Enter on OS X)</u>)** when the caret is in a link element.

> **Note:** Depending on the referred file type, the **Open file at caret** action opens the target link either in the Oxygen XML Author or in the default system application. If the target file does not exist, Oxygen XML Author prompts you to create it.

Displaying the Markup

In the **Author** mode, you can control the amount of displayed markup using the following dedicated actions from the toolbar:

Full Tags with Attributes

> Displays full name tags with attributes for both block level as well as in-line level elements.

Full Tags

> Displays full name tags without attributes for both block level as well as in-line level elements.

Block Tags

> Displays full name tags for block level elements and simple tags without names for in-line level elements.

Inline Tags

> Displays full name tags for inline level elements, while block level elements are not displayed.

Partial Tags

> Displays simple tags without names for in-line level elements, while block level elements are not displayed.

No Tags

> No tags are displayed. This is the most compact mode.

To set a default mode of the tags mode, go to *Author preferences page* and configure the **Tags display mode** mode. However, if the document opened in Author editor does not have an associated CSS stylesheet, then the **Full Tags** mode will be used.

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

Bookmarks

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

A bookmark can be placed with:

- one of the menu items available on the menu **Edit** > **Bookmarks** > **Create**;
- the menu item **Edit** > **Bookmarks** > **Bookmarks Quick Creation (<u>F9</u>)**;
- 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**.

Visual Hints of the Caret's Position in the Document

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.
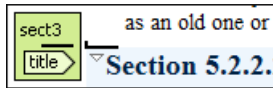
Figure 28: Before first block

- The caret is positioned between two block elements.
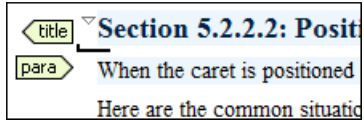
Figure 29: Between two block elements

- The caret is positioned after the last block element child of the current node.
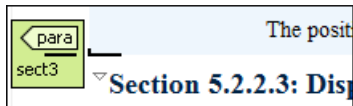
Figure 30: After last block

- The caret is positioned inside a node.
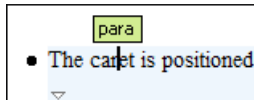
Figure 31: Inside a node

- The caret is positioned inside an element, before an inline child element.
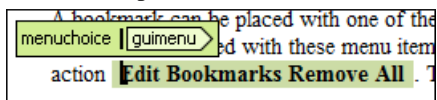
Figure 32: Before an inline element

- The caret is positioned between two inline elements.
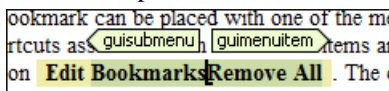
Figure 33: Between two inline elements

- The caret is positioned inside an element, after an inline child element.
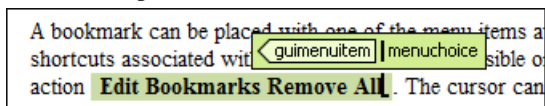
Figure 34: After an inline element

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

To deactivate this feature *open the **Preferences** dialog* and go to **Editor / Author** > **Show caret position tooltip**. 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.

Location Tooltip

When editing XML documents in a visual environment you might find it difficult to position the caret between certain tags that do not have a visual representation. To counterbalance this, Oxygen XML Author displays a transparent preview of the Position Information Tooltip, called *Location Tooltip*:
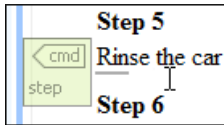
Figure 35: Location Tooltip

Oxygen XML Author displays a location tooltip when the following conditions are met:

- you are editing the document in one of the following *tags display modes*: **Inline Tags**, **Partial Tags**, **No Tags**
- the mouse pointer is moved between block elements

To activate or deactivate this feature, use the **Show location tooltip on mouse move** option from the *Caret Navigation preferences page*.

Displaying Referred Content

The references to entities, XInclude, and DITA conrefs are expanded by default in Author mode and the referred content is displayed. You can control this behavior from the *Author preferences page*. The referred resources are loaded and displayed inside the element or entity that refers them, however the displayed content cannot be modified directly.
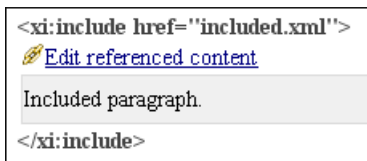
Figure 36: XInclude reference
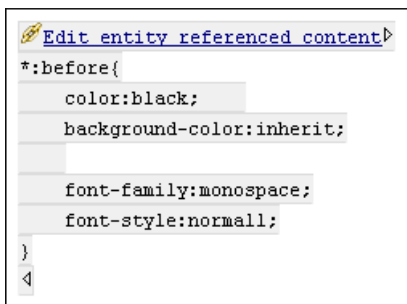
Figure 37: External entity reference

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

If you want to make modifications to the referred content, you must open the referred resource in an editor. The referred resource can be opened quickly by clicking 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 Author;
- on demand, by using the *Refresh references action*. Useful when the referred content is modified outside the Oxygen XML Author scope.

Finding and Replacing Text

You can search for a specific word or string of characters using the following features:

- *Find/Replace dialog box*
- *Find/Replace in Files dialog box*
- *Quick Find toolbar*
- *Find all Elements dialog box*

Complex search operations may take some time to complete. If a search operation takes more than 5 seconds, you are prompted to decide whether you want to continue the operation or stop it.

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).

The generic actions are:

Quick Fix (Alt + 1 (Meta + Alt + 1 on OS X))

Available when the contextual menu is invoked on an error where *Oxygen XML Author can provide a quick fix*.

Open Image

Available when the contextual menu is invoked on an image. This action allows you to open an image in the Oxygen XML Author's *Image Viewer* or in a default system application associated with the current image type.

a **Edit Attributes**

A pop-up window is displayed allowing you to *manage in-place the element's attributes*.

Edit Profiling Attributes

Allows you to change the *profiling attributes* defined on all selected elements.

Cut, Copy, Paste

Common edit actions with the same functionality as those found in the text editor.

Paste special > Paste As XML

Similar to **Paste** operation, except that the clipboard's content is considered to be XML.

Paste special > Paste As Text

Pastes the clipboard content, ignoring any structure or styling markup, if any.

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.

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.

Text

Contains the following actions:

Text > To Lower Case

Converts the selected content to lower case characters.

Text > To Upper Case

Converts the selected 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 XML structure:

Toggle Comment

Encloses the currently selected text in an XML comment, or removes the comment if it is commented.

Move Up

Moves the current node or selected nodes in front of the previous node.

Move Down

Moves the current node or selected nodes after the successive node.

Split Element

Splits the content of the closest element that contains the caret's position. Thus, if the caret is positioned at the beginning or at the end of the element, the newly created sibling will be empty.

Join Elements

Joins two adjacent elements that have the same name. The action is available only when the caret position is between the two adjacent elements. Also, joining two elements can be done by pressing the **Delete** or **Backspace** keys and the caret is positioned between the boundaries of these two elements.

Surround with Tag...

Selected text in the editor is marked with the specified start and end tags.

Surround with <Tag name>

Selected text in the editor is marked with start and end tags used by the last **Surround with Tag...** action.

Rename Element

The element from the caret position and the elements that have the same name as the current element can be renamed according with the options from the **Rename** dialog.

Delete Element Tags

Deletes the tags of the closest element that contains the caret's position. This operation is also executed if the start or end tags of an element are deleted by pressing the **Delete** or **Backspace** keys.

Review

Provides access to *Track Changes* and *Manage Comments* actions.

Manage IDs

Provides access to *searching and refactoring actions for ID/IDREFS*.

Insert > Insert Entity

Allows the user to insert a predefined entity or a character entity. Surrogate character entities (range #x10000 to #x10FFFF) are also accepted. Character entities can be entered in one of the following forms:

- #*<decimal value>* - e. g. #65;
- &#*<decimal value>*; - e. g. A
- #x*<hexadecimal value>* - e. g. #x41;
- &#x*<hexadecimal value>*; - e. g. A

Options

Opens the *Author options page*.

Document type actions are specific to some document type. Examples of such actions can be found in the *DocBook 4 Author Extensions* and *DITA Author Extensions* sections.

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 (Command+Space on OS X)** in the editor panel.

Content completion window offers the following types of actions:

- inserting allowed elements for the current context according to the associated schema, if any;
- inserting element values if such values are specified in the schema for the current context;
- inserting new undeclared elements by entering their name in the text field;
- inserting CDATA sections, comments, processing instructions;
- inserting *code templates*.
- if the **Show all possible elements in the content completion list** option from the *Schema aware preferences page* is enabled, the content completion pop-up window will present all the elements defined by the schema. When choosing an element from this section, the insertion will be performed using the schema aware smart editing features.
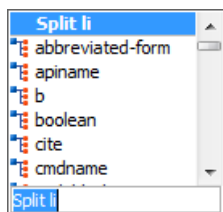
Figure 38: Content completion window

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

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

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

Note: The content completion list of proposals contains elements depending on the elements inserted both before and after the caret position.

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

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

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

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

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

The same action can be triggered in the next situations:

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

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

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

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

Code Templates

Code templates are code fragments that can be inserted quickly at the editing position. Oxygen XML Author comes with a set of ready-to use templates for XSL, XQuery, and XML Schema. You can also *define you own code templates and share them with others*.

To get a complete list of available code templates, press **Ctrl+Shift+Space (Command+Shift+Space on OS X)**. To enter the code template, select it from the list or type its shortcut code and press **Enter**.

If you know the template shortcut, type it and press **Ctrl+Shift+Space (Command+Shift+Space on OS X)**.

Code templates are also displayed when you press the content completion keys,**Ctrl+Space (Command+Space on OS X)** or **Enter** in *Author mode*.

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

Editing the XML Content

By default you can type only in elements which accept text content. So if the element is declared as empty or element only in the associated schema you are not allowed to insert text in it. This is also available if you try to insert CDATA inside an element. Instead a warning message is shown:
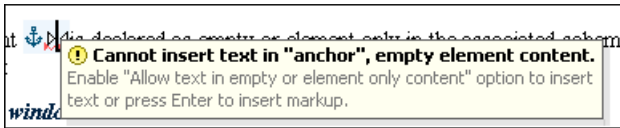
Figure 39: Editing in empty element warning

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.

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 XML Author 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 IDs when copying content in the same document** allows you to control if a pasted element should retain its ID.

Table Layout and Operations

Oxygen XML Author provides support for editing data in a tabular form. The following operations are available:

- **Adjusting column width**

  You are able to manage table width and column width specifications from the source document. These specifications are supported both in fixed and proportional dimensions. The predefined frameworks (DITA, DocBook, and XHTML) also support this feature. The layout of the tables for these document types takes into account the table width and the column width specifications particular to them. To adjust the width of a column or table, drag the border of the column. The changes you make to a table are committed into the source document.

Figure 40: Resizing a Column in Oxygen XML Author Author Editor

- **Column and row selection**

To select a row or a column of a table, place the mouse cursor above the column or in front of the row you want to select, then click. When hovering the mouse cursor in front of rows or above column headers, the cursor changes to

for row selection and to for column selection and that specific row or column is highlighted.

- **Cell selection**

  To select a cell in a table, press and hold the **Ctrl** key and click anywhere inside the cell. You can use this action to select one or more cells, and also to deselect cells from a selection. Alternatively, you can click one of the left corners

  of a cell (right corners in case you are editing a *RTL document*). The cursor changes to when it hovers over the corners of the cell.

- **Rectangular selection**

  To select a rectangular block of cells do one of the following:

  - click a cell and drag to expand the selection
  - click a cell, then press the **Shift** key and use the arrow keys to expand the selection

- **Drag and drop**

  You can use the drag and drop action to edit the content of a table. You are able to select a column and drag it to another location in the table you are editing. When you drag a column and hover the cursor over a valid drop position, Oxygen XML Author decorates the target location with bold rectangles. The same drag and drop action is also available for rows.

- **Copy-paste and cut for columns and rows**

  In Oxygen XML Author, you are able to copy entire rows or columns of the table you are editing. You can paste a copied column or row both inside the source table and inside other tables. The cut operation is also available for rows and columns. You can use the cut and the copy-paste actions for tables located in different documents as well.

  When you paste a column in a non-table content, Oxygen XML Author introduces a new table which contains the fragments of the source column. The fragments are introduced starting with the header of the column. When you copy a column of a CALS table, Oxygen XML Author preserves the width information of the column. This information is then used when you paste the column in another CALS table.

- **Content deletion**

  To delete a group of cells (can be columns, rows, or rectangular block of cells), select them and do one of the following:

  - press either **Delete**, or **Backspace** on your keyboard to delete the cells' content. Press again **Delete**, or **Backspace** to remove the selected table structure
  - if the selection is a column or a row, you can use the **Delete a table row** or **Delete a table column** actions to delete both the content and table structure

DocBook Table Layout

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

In the CALS table model, you can specify column widths using the `colwidth` attribute of the associated `colspec` element. The values can be fixed or proportional. By default, when you insert, drag and drop, or copy/paste a column, the value of the `colwidth` attribute is `1*`.

Also the `colsep` and `rowsep` attributes are supported. These control the way separators are painted between the table cells.

Figure 41: CALS table in Docbook

Editing Table Component Properties

To customize the look of a table, place the caret anywhere in a table and invoke the ▦ **Table Properties (Ctrl+T (Command+T on OS X))** action from one of the following locations:

- **contextual menu** > **Table** menu
- **main menu** > **Docbook** > **Table**
- ▦ **Table Properties** toolbar action

The **Table properties** dialog allows you to set specific properties to the table elements.

> 📄 **Note:** Depending on the context, some options or values are filtered out;

> 📄 **Note:** If you want to remove a property, set its value to **<not set>**;

> 📄 **Note:** Choose the **<preserve>** setting if you want to:
> - keep the current non-standard value for a particular property;
> - keep the values already set. This happens when you select multiple elements having the same property set to different values.

For a **CALS** table you can format any of the following:

- **Table** - set the horizontal alignment, row and column separators and the table's frame;
- **Row** - set the row type, vertical alignment and row separator;
- **Column** - set the horizontal alignment, and column and row separators;
- **Cell** - set the horizontal and vertical alignment, column and row separators.

For an **HTML** table you can customize any of the following:

- **Table** - set the frame attribute;
- **Row** - set the row type, horizontal and vertical alignment;
- **Column** - set the horizontal and vertical alignment;
- **Cell** - set the horizontal and vertical alignment.

XHTML Table Layout

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

x	y	Spans ▷Horizontally◁	
Spans ▷Vertically ◁	b		
	Spans ▷Both◁	d	
		e	f
g	h	i	k

A table with merged cells, fixed column widths, and fixed total width.

▶ colspecs...

Figure 42: HTML table

DITA Table Layout

Depending on the context, the DITA table layout accepts CALS tables, simple tables, and choice tables.

In the CALS table model, you can specify column widths using the `colwidth` attribute of the associated `colspec` element. The values can be fixed or proportional. By default, when you insert, drag and drop, or copy/paste a column, the value of the `colwidth` attribute is `1*`.

Also the `colsep` and `rowsep` attributes are supported. These control the way separators are painted between the table cells.

Sample CALS Table with no specified width and proportional column widths

▽ colspecs...

column name c1	number 1	width 0.32*	align ▼	☐ colsep	☐ rowsep
column name c2	number 2	width 1.49*	align ▼	☐ colsep	☐ rowsep
column name c3	number 3	width 1.15*	align ▼	☐ colsep	☐ rowsep
column name c4	number 4	width 0.4*	align ▼	☐ colsep	☐ rowsep
column name c5	number 5	width 1.67*	align ▼	☐ colsep	☐ rowsep

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

Figure 43: CALS table in DITA

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

Header 1	**Header 2**
Column 1	Column 2

Figure 44: DITA simple table

You can insert choice tables in DITA tasks either using the **Content Completion Assistant** or using the toolbar and contextual menu actions.

Editing Table Component Properties

To customize the look of a table, place the caret anywhere in a table and invoke the 🔲 **Table Properties (Ctrl+T (Command+T on OS X))** action from one of the following locations:

- **contextual menu** > **Table** menu

- **main menu** > **DITA** > **Table**
- 📇 **Table Properties** toolbar action

The **Table properties** dialog allows you to set specific properties to the table elements.

📄 **Note:** Depending on the context, some options or values are filtered out;

📄 **Note:** If you want to remove a property, set its value to **<not set>**;

📄 **Note:** Choose the **<preserve>** setting if you want to:

- keep the current non-standard value for a particular property;
- keep the values already set. This happens when you select multiple elements having the same property set to different values.

For a **CALS** table you can format any of the following:

- **Table** - set the horizontal alignment, row and column separators and the table's frame;
- **Row** - set the row type, vertical alignment and row separator;
- **Column** - set the horizontal alignment, and column and row separators;
- **Cell** - set the horizontal and vertical alignment, column and row separators.

For a *simple* table you can customize any of the following:

- **Table** - set the frame attribute;
- **Row** - set the row type.

Sorting Content in Tables and List Items

Oxygen XML Author offers support for sorting the content of tables and list items of ordered and unordered lists.

What do you want to do?

- *Sort an entire table*;
- *Sort a selection of rows in a table*;
- *Sort a table that contains cells merged over multiple rows*;
- *Sort a table based on multiple sorting criteria*;
- *Sort list items*.

Sorting an Entire Table

To sort an entire table either right click the table and select ⌛ **Sort**, or select the table and click ⌛ **Sort** on the main toolbar. Any of the two opens the **Sort** dialog box.
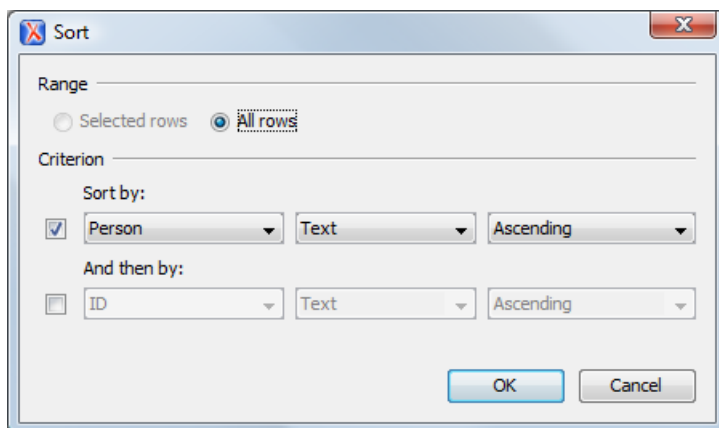
Figure 45: The "Sort" Dialog Box

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

📑 **Note:** When you invoke the sorting operation over an entire table, the **Selected rows** option is disabled.

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

- a name, which is collected from the column heading;
- the type of the information that is sorted (either text, numeric, or date);
- the sorting direction (either ascending or descending).

The sort criteria is set automatically to the column where the caret is located at the time when the sorting operation is invoked.

📑 **Note:** The sorting mechanism of Oxygen XML Author recognizes multiple date formats like *short*, *medium*, *long*, *full*, *xs:date*, and *xs:dateTime*.

After you finish configuring the options in the **Sort** dialog box, click **OK** to complete the sorting operation. In case you want to go back to the initial order of your content, press **Ctrl+Z (Command+Z on OS X)** on your keyboard.

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

Sorting a Selection of Rows

To sort a selection of rows in a table, select the rows that you want to sort and either right click the selection and choose ⇅ **Sort**, or click ⇅ **Sort** on the main toolbar. Any of the two opens the **Sort** dialog box.
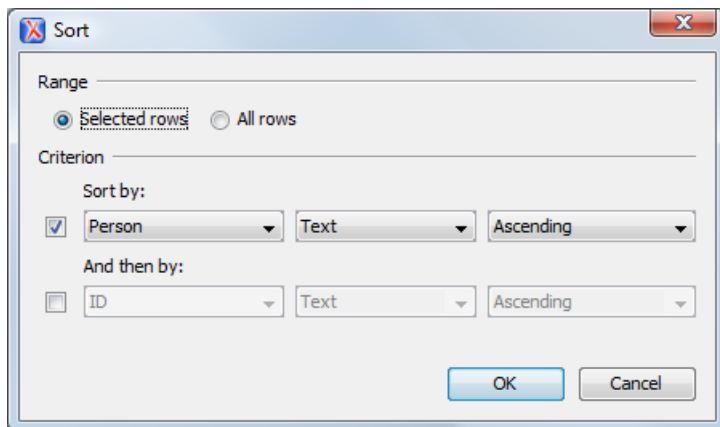
Figure 46: Sort Selected Rows

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

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

- a name, which is collected from the column heading;
- the type of the information that is sorted (either text, numeric, or date);
- the sorting direction (either ascending or descending).

The sort criteria is set automatically to the column where the caret is located at the time when the sorting operation is invoked.

📑 **Note:** The sorting mechanism of Oxygen XML Author recognizes multiple date formats like *short*, *medium*, *long*, *full*, *xs:date*, and *xs:dateTime*.

After you finish configuring the options in the **Sort** dialog box, click **OK** to complete the sorting operation. In case you want to go back to the initial order of your content, press **Ctrl+Z (Command+Z on OS X)** on your keyboard.

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

Sorting a Table that Contains Merged Cells

In case a table contains cells that span over multiple rows, you can not perform the sorting operation over the entire table. Still, the sorting mechanism works over a selection of rows that do not contain rowspans.

> **Note:** For this type of table, the **Sort** dialog keeps the **All rows** option disabled even if you perform the sorting operation over a selection of rows.

Sorting Using Multiple Criteria

You can sort both an entire table or a selection of its rows based on multiple sorting criteria. To do so, enable the rest of the criteria in the **Sort** dialog, configure the items of each criterion and click **OK** to complete the sorting operation.
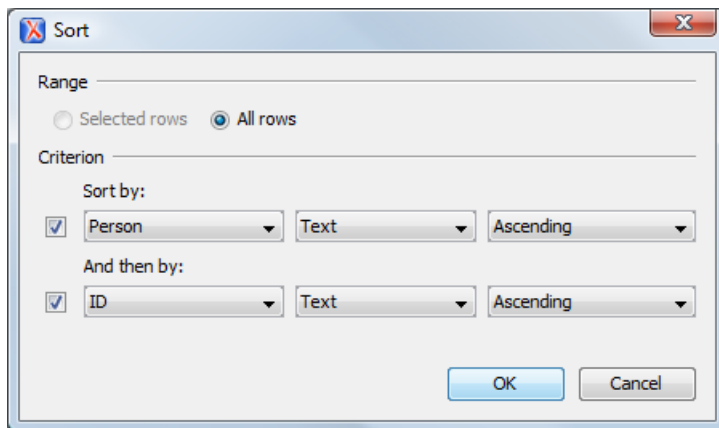
Figure 47: Sorting Based on Multiple Criteria

Sorting List Items

You can perform the sorting operation over list items of ordered and unordered lists.
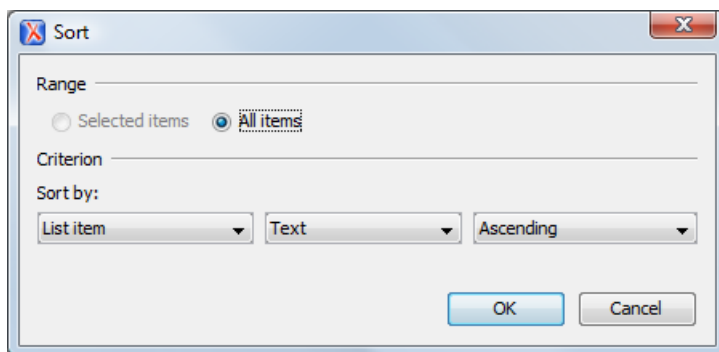
Figure 48: Sorting List Items

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

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

Image Rendering

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

Table 3: Supported Image Formats

Image Type	Support	Additional Information
GIF	built-in	Animations not yet supported
JPG, JPEG	built-in	*JPEG images with CMYK color profiles* are properly rendered only if color profile is inside the image.
PNG	built-in	
SVG, SVGZ, WMF	built-in	Rendered using the open-source Apache Batik library which supports SVG 1.1.
BMP	built-in	
TIFF	built-in	Rendered using a part of the Java JAI Image library.
EPS	built-in	Renders the preview TIFF image inside the EPS.
AI	built-in	Renders the preview image inside the Adobe Illustrator file.
JPEG 2000, WBMP	plug-in	Renders by *installing the Java Advanced Imaging (JAI) Image I/O Tools plug-in*.
CGM	plug-in	Renders by *installing an additional library*.
PDF	plug-in	Renders by *installing the Apache PDF Box library*.

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

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

Scaling Images

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

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

📄 **Note:**

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

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

Follow this procedure:

1. Start Oxygen XML Author and open the **Help** > **About** dialog. Open the **System properties** tab and look for *java.runtime.name* and *java.home* properties. Keep their values for later use.
2. *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.

OS X Workaround

There is no native implementation of JAI Image I/O for OS X 10.5 and later. However, the JAI Image I/O has a Java implementation fallback which also works on 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_DIR]/lib` directory create a directory named `endorsed` e.g. `[OXYGEN_DIR]/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_DIR]/lib/endorsed` directory.

4. Restart the application and the JAI Image I/O support will be up and running.

Customize Oxygen XML Author to Render CGM Images (Experimental Support)

Oxygen XML Author provides experimental support for CGM 1.0 images.

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

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

The CGM rendering support is based on a third party library. In its free of charge variant it renders the images watermarked with the string `Demo`, painted across the panel. You can find more information about ordering the fully functioning version here: *http://www.bdaum.de/cgmpanel.htm*.

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

1. Download the `CGMPANEL.ZIP` from *http://www.bdaum.de/CGMPANEL.ZIP*.

2. Unpack the ZIP archive and copy the `cgmpanel.jar` into `[OXYGEN_DIR]\lib` directory.

3. Restart the application.

Customize Oxygen XML Author to Render PDF Images (Experimental Support)

Oxygen XML Author provides experimental support for PDF images using the Apache PDFBox library.

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

1. Go to *http://pdfbox.apache.org/downloads.html* and download the pre-built PDFBox standalone binary `pdfbox-app-1.8.6.jar`.

2. Copy `pdfbox-app-1.8.6.jar` into `[OXYGEN_DIR]\lib` directory.

3. Restart the application.

Customize Oxygen XML Author to Render EPS and AI Images

Most EPS and AI image files include a preview picture of the content. Oxygen XML Author tries to render this preview picture. The following scenarios are possible:

- the EPS or AI image does not include the preview picture. Oxygen XML Author cannot render the image.
- the EPS image includes a TIFF preview picture.

  📄 **Note:** Some newer versions of the TIFF picture preview are rendered in gray-scale.

- the AI image contains a JPEG preview picture. Oxygen XML Author renders the image correctly.

Adding an Image

To insert an image in a document while editing in **Author** mode, use one of the following methods:

- Click the **Insert Image Reference** action from the toolbar and choose the image file you want to insert. Oxygen XML Author tries to reference the image with a path that is relative to that of the document you are currently editing. For example, if you want to add the `file:/C:/project/xml/dir/img1.jpg` image into `file:/C:/project/xml/doc1.xml` document, Oxygen XML Author inserts a reference to `dir/img1.jpg`. This is useful when multiple users work on a common project and they have it stored in different locations in their computers.

  > **Note:** The **Insert Image Reference** action is available for the following document types: DocBook 4, DocBook 5, DITA, TEI P4, TEI P5, XHTML.

- Drag an image from other application and drop it in the **Author** editor. If it is an image file, it is inserted as a reference to the image file. For example, in a DITA topic the path of the image file is inserted as the value of the `href` attribute in an `image` element:

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

  > **Note:** To replace an image, just drag and drop a new image over the existing one. Oxygen XML Author will automatically update the reference to the new image.

- Copy the image from other application (like an image editor) and paste it in your document. Oxygen XML Author prompts you to first save it. After saving the image to a file, a reference to that file path is inserted at the drop position.

Editing MathML Notations

The **Author** editor includes a built-in editor for *MathML* notations. To start the *MathML* editor, either double click a *MathML* notation, or select the **Edit Equation** action from its contextual menu. In the *MathML* editor you are able to edit the mathematical symbols of a *MathML* notation. You can open a *MathML* file of your current project directly in the *MathML* editor. To do this, select **Open with** > **MathML editor** from the contextual menu in the **Project** view.
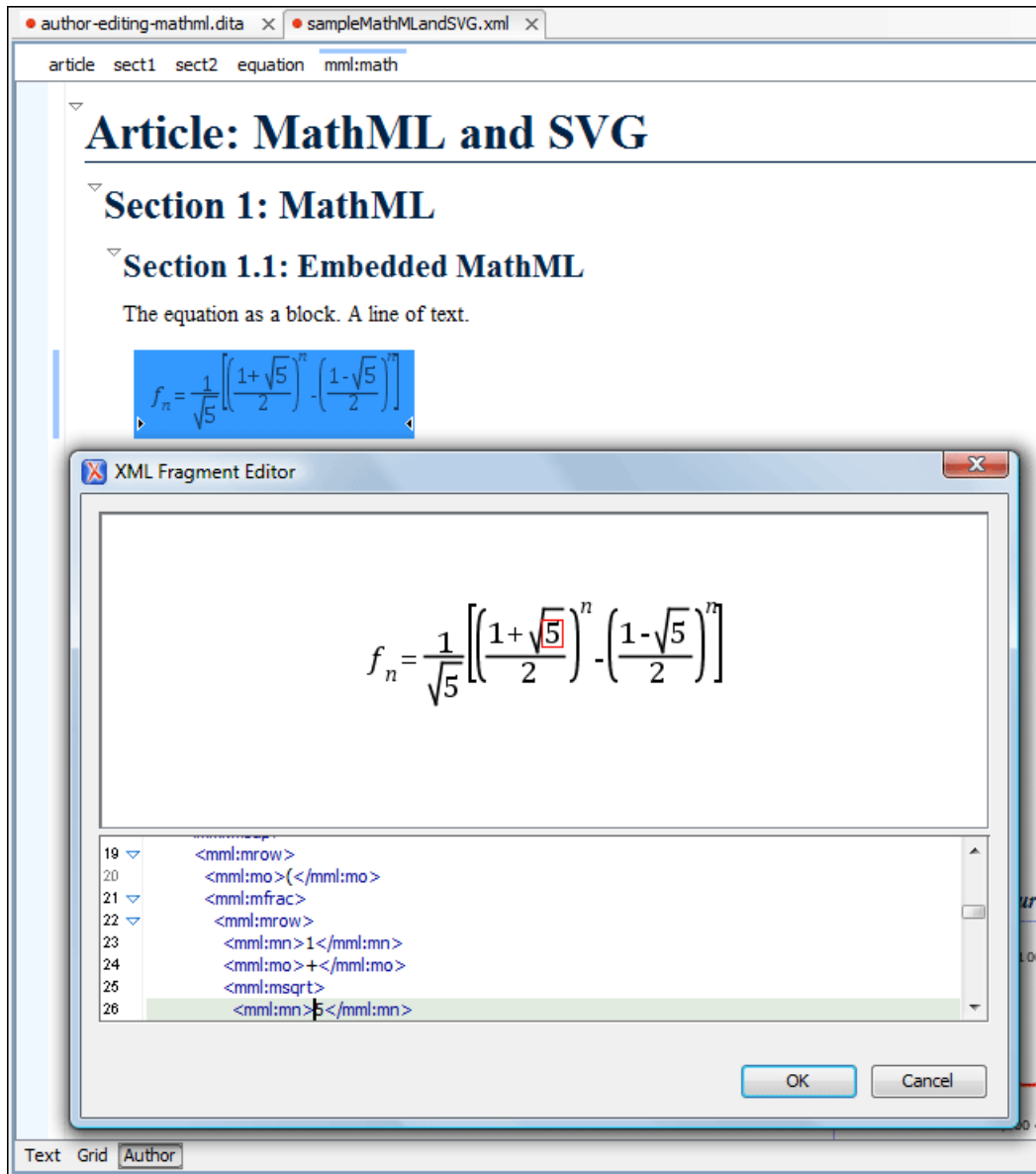
Figure 49: The default *MathML* editor

To customize the font size of the mathematical symbols and the *MathFlow* SDK, *open the Preferences dialog* and go to **Editor** > **Edit modes** > **Author** > **MathML**.

Configure the MathFlow Editor

The MathFlow Components (the MathFlow SDK) can replace the default MathML editor with a specialized MathML editor. You have to *purchase a MathML component from Design Science* and configure it in Oxygen XML Author with the following procedure:
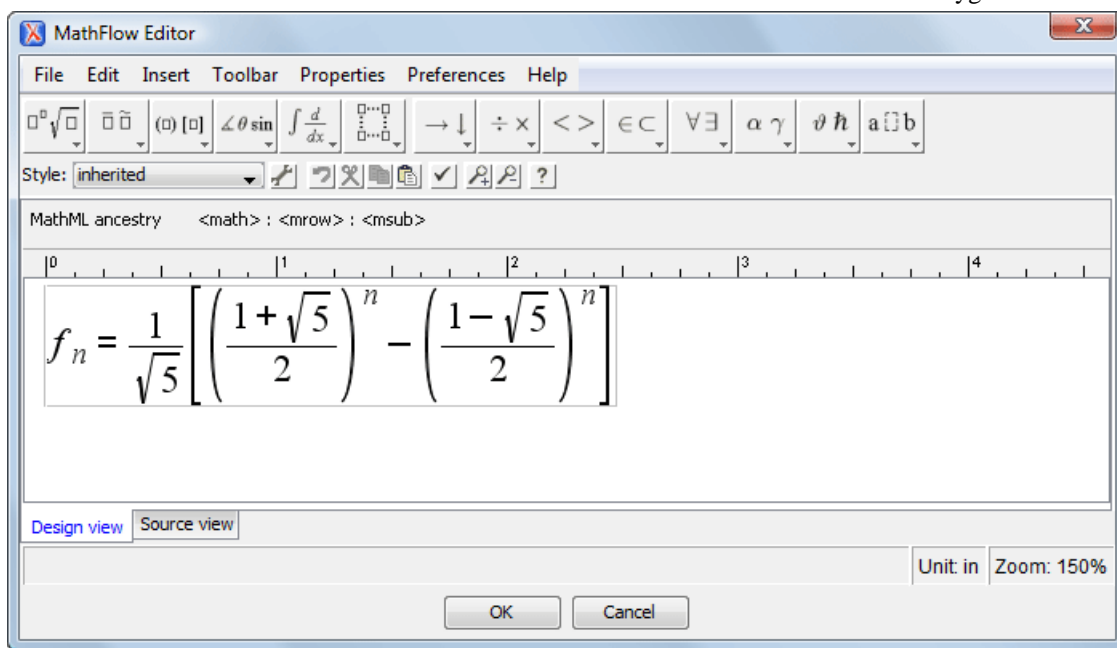
Figure 50: The default MathML editor

1. Install MathFlow Components (the MathFlow SDK).
2. On Windows make sure there is a copy of the FLEXlm DLL, that is the file `[MathFlow-install-folder]/resources/windows/lmgr10.dll`, in a folder that is added to the *PATH* environment variable.
3. Set the path to the MathFlow install folder *in the **Preferences***.
4. Set the path to the MathFlow license file *in the **Preferences***.

Refreshing the Content

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

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

Validation and Error Presenting

Automatic validation as well as validate on request operations are available while editing documents in the Author editor. A detailed description of the document validation process and its configuration is described in section *Validating Documents*.
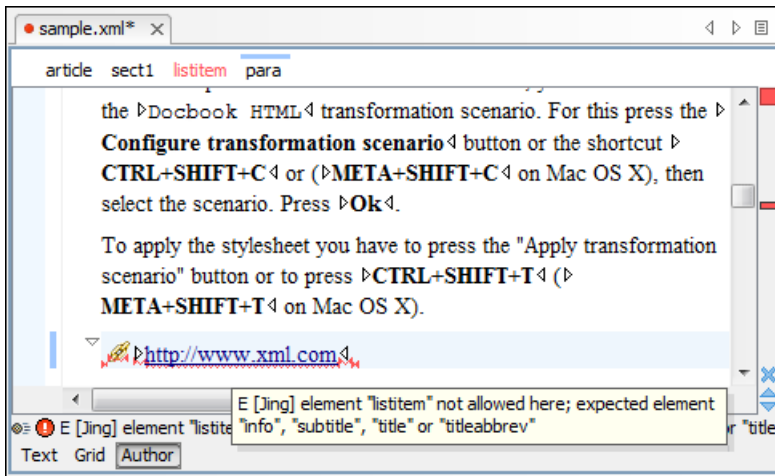
Figure 51: Error presenting in Oxygen XML Author Author editor

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

- The top area - a success validation indicator that will turn green in case the validation succeeded or red otherwise. A more detailed report of the errors is displayed in the tool tip. In case there are errors, only the first three of them will be presented in the tool tip.
- The middle area - the errors markers are depicted in red (with a darker color tone for the current selected one). To limit the number of markers shown, *open the Preferences dialog* and go to **Editor** > **Document checking** > **Limit error markers to**.

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

  The *Document checking user preferences* are easily accessible from the button displayed at the beginning of the error message on the bottom of the editor panel.

- The bottom area - two navigation arrows that will go to the next or to the previous error and a button for clearing all the error markers from the ruler. The same actions can be triggered from **Document** > **Automatic validation** > **Next error** (**Ctrl+. (Command+. on OS X)**) and **Document** > **Automatic validation** > **Previous error** (**Ctrl+, (Command+, on OS X)**).

The validation status area is the line at the bottom of the editor panel that presents the message of the current validation error. Clicking on ⚙ opens the *Document checking* page in Oxygen XML Author user preferences.

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

Author Whitespace Handling

When you edit a document in Author mode, Oxygen XML Author must serialize the resulting document as XML. Oxygen XML Author serializes the document when you save it or switch to another editing mode. When the document is serialized, Oxygen XML Author *formats and indents the XML document* according to the current *format and indent settings*.

Minimizing whitespace differences between versions

When serializing a document to XML, Author mode will only format and indent those elements of the document that have been edited. Any element that has not been edited will be serialized exactly as it was loaded from disk. This is useful when your content is managed in a version control systems, as it avoids introducing insignificant whitespace differences between version, which in turn makes diff output easier to read.

Entering whitespace in Author mode

Oxygen XML Author controls the entry of whitespace characters in Author mode according the *XML whitespace rules*, which means it will not let you insert insignificant whitespace. This means that it will not let you insert extra line-breaks or spaces inside a typical paragraph element, for instance. (Any such whitespace would be normalized away when the document was serialized to XML, so Oxygen XML Author is saving you from any surprises when this happens.)

Of course, you will legitimately want to enter additional spaces and returns in some cases, such as code samples. Oxygen XML Author will allow this in elements that are configured as preserve space elements according to the XML whitespace rules. For all of its *predefined document types*, Oxygen XML Author is *correctly configured to recognize preserve space elements* and to allow you to enter additional spaces in them.

If you are using a predefined document type and you are unable to enter additional whitespace, make sure that you are using an element from that document type that is intended to be a preserve-space element.

If you are using a custom document type, make sure that it is *configured correctly* so that Oxygen XML Author recognizes that the current element is a preserve-space element.

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 Author 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 Author 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. To activate track changes for the current document, either choose **Edit** > **Review** > **Track Changes** or click the **Track Changes** button on the **Author Review** toolbar. When **Track Changes** is enabled, your modifications are highlighted using a distinctive color. The name of the author who is currently making changes and the colors can be customized from the *Review preferences page*.

Figure 52: Change Tracking in Author Mode

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

Track Changes highlights textual changes and also changes that you make to the attributes in a document. Here is the list of tracked changes:

- Inserting, deleting content (text or elements)
- Drag and drop content (text or elements)
- Cutting, or pasting content (text or elements)
- Inserting, deleting and changing the structure of tables
- Inserting and editing lists and their content
- Inserting and deleting entities
- Deleting element tags
- Editing attributes
- Performing a **Split** operation
- Performing a **Surround with** operation

If the selection in the **Author** contains track changes and you are copying it, the clipboard contains the selection with all the *accepted* changes. This filtering is performed only if the selection is not entirely inside a tracked change. The changes are stored in the document as processing instructions and they do not interfere with validating and transforming it. For each change the author name and the modification time are preserved. The following processing instructions are examples of storing *insert* and *delete* changes in the document:

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

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

Adding Document Comments

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

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

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

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

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

Note: Oxygen XML Author presents the tracked changes in DITA conrefs and XInclude fragments.

Managing Changes

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

Track Changes

Enables or disables the track changes support for the current document.

Accept Change(s)

Accepts the change located at the caret position. If you select a part of a delete or insert change, then only the selected content is accepted. If you select multiple changes, all of them are accepted. For an insert change, it means keeping the inserted text and for a delete change it means removing the content from the document.

Reject Change(s)

Rejects the change located at the caret position. If you select a part of a delete or insert change, then only the selected content is rejected. If you select multiple changes, all of them are rejected. For an insert change, it means removing the inserted text and for a delete change it means preserving the original content from the document.

Comment Change

You can decide to add additional comments to an already existing change. The additional description appears in the tooltip when hovering over the change and in the **Manage Tracked Changes** dialog when navigating changes.

Highlight

Enables the *Highlight tool*.

Colors

Opens the colors palette of the *Highlight tool*.

Stop Highlighting

Disables the *Highlight tool*.

Add Comment

Inserts a comment in the document you are editing, at the caret position.

Edit Comment

Edits a selected comment from the edited document.

Remove Comment

Removes a selected comment from the edited document.

 Manage Reviews

Opens the *Review view*.

Track Changes Visualization Modes

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

- **View All Changes/Comments** - This mode is active by default. When you use this mode, all tracked changes are represented in the Author mode.
- **View only Changes/Comments by** - Only the tracked changes made by the author you select are presented.
- **View Final** - This mode offers a preview of the document as if all tracked changes (both inserted and deleted) were accepted.
- **View Original** -this mode offers a preview of the document as if all tracked changes (both inserted and deleted) were rejected. You cannot edit the document in this mode. Attempting to do so switches the view mode to **View All Changes**.

All four actions are available only in a drop-down list in the **Author Review** toolbar. If you use **View Final** mode and **View Original** mode, highlighted comments are not displayed. To display highlighted comments, use **View All Changes/Comments**.

To watch our video demonstration about the Track Changes support, go to *http://oxygenxml.com/demo/Change_Tracking.html*.

Track Changes Behavior

This section explains the behaviour of the **Track Changes** feature depending on the context and whether it is activated.

You can use the **Track Changes** feature to keep track of multiple actions.

Possible change tracking scenarios:

- *Keep tracking of inserted content*
- *Keep tracking of surrounded content*
- *Keep tracking of deleted characters*
- *Keep tracking of deleted content*
- *Keep tracking of copied content*
- *Keep tracking of pasted content*
- *Keep tracking of attribute changes*

Keep Tracking of Inserted Content

When **Track Changes** is disabled and you insert content, the following cases are possible:

- making an insertion in a **Delete** change - the change is split in two and the content is inserted without being marked as change
- making an insertion in an **Insert** change - the change is split in two and the content is inserted without being marked as change
- making an insertion in regular content - regular insertion

When **Track Changes** is enabled and you insert content, the following cases are possible:

- making an insertion in a **Delete** change - the change is split in two and the current inserted content appears marked as an INSERT
- making an insertion in an **Insert** change:
  - if the original insertion was made by another user, the change is split in two and the current inserted content appears marked as an INSERT by the current author
  - if the original **Insert** change was made by the same user, the change is just expanded to contain the inserted content. The creation time-stamp of the previous insert is preserved

- if we insert in regular content, the current inserted content appears marked as an **Insert** change

Keep Tracking of Surrounded Content

When **Track Changes** is enabled and you surround content in a new XML element, the following cases are possible:

- making a surround in a **Delete** change - nothing happens
- making a surround in an **Insert** change:
  - if the original insertion was made by another user, the change is split in two and the surround operation appears marked as being performed by the current author
  - if the original **Insert** change was made by the same user, the existing change is just expanded to contain the surrounded content
- making a surround in regular content - the operation is marked as a surround change.

Keep Tracking of Deleted Characters

When **Track Changes** is disabled and you delete content character by character, the following cases are possible:

- deleting content in an existing **Delete** change - nothing happens;
- deleting content in an existing **Insert** change - the content is deleted without being marked as a deletion and the INSERT change shrinks accordingly;
- deleting in regular content - regular deletion.

When **Track Changes** is enabled and you delete content character by character, the following cases are possible:

- deleting content in an existing **Delete** change:
  - if the same author created the **Delete** change, the previous change is marked as deleted by the current author;
  - if another author created the **Delete** change, nothing happens.

- deleting content in an existing **Insert** change:
  - if the same author created the **Insert** change, the content is deleted and the **Insert** change shrinks accordingly;
  - if another author created the **Insert** change, the **Insert** change is split in two and the deleted content appears marked as a **Delete** change by the current author.

- deleting in regular content - the content is marked as **Delete** change by the current author.

Keep Tracking of Deleted Content

When **Track** changes is disabled and you delete selected content, the following cases are possible:

- the selection contains an entire **Delete** change - the change disappears and the content is deleted;
- the selection intersects with a **Delete** change (starts or ends in one) - nothing happens;
- the selection contains an entire **Insert** change - the change disappears and the content is deleted ;
- the selection intersects with an **Insert** change (starts or ends in one), the **Insert** change is shrieked and the content is deleted.

When **Track** changes is enabled and you delete selected content, the following cases are possible:

- the selection contains an entire **Delete** change - the change is considered as rejected and then marked as deleted by the current author, along with the other selected content;
- the selection intersects a **Delete** change (starts or ends in one) - the change is considered as rejected and marked as deleted by the current author, along with the other selected content;
- the selection contains an entire **Insert** change:
  - if the **Insert** is made by the same author, the change disappears and the content is deleted;
  - if the **Insert** is made by another author, the change is considered as accepted and then marked as deleted by the current author, along with the other selected content;

- if the selection intersects an **Insert** change (starts or ends in one), the **Insert** change shrinks and the part of the **Insert** change that intersects with the selection is deleted.

Keep Tracking of Copied Content

When **Track Changes** is disabled and you copy content the following cases are possible:

- if the copied area contains **Insert** or **Delete** changes, these are also copied to the clipboard.

When **Track Changes** is enabled and you copy content the following cases are possible:

- if the copied area contains **Insert** or **Delete** changes, these are all accepted in the content of the clipboard (the changes will no longer be in the clipboard).

Keep Tracking of Pasted Content

When **Track Changes** is disabled and you paste content the following cases are possible:

- if the clipboard content contains INSERT OR DELETE changes, they will be preserved on paste.

When **Track Changes** is enabled and you paste content the following cases are possible:

- if the clipboard content contains **Insert** or **Delete** changes, all the changes are accepted and then the paste operation proceeds according to the insertion rules.

Keep Tracking of Attribute Changes

The **Track Changes** feature is able to keep the track of changes you make to attributes in a document. If the *Callouts support is enabled*, all the attribute changes are presented as callouts in the document you are editing. The changes are also presented in the *Review view* and *Attributes view*.

When you copy a fragment that contains tracked attribute changes, the following cases are possible:

- if you perform the copy operation with **Track Changes** enabled, all the attribute changes in the fragment are accepted;
- if you perform the copy operation with **Track Changes** disabled, the fragment holds the attribute changes inside it.

When you paste a fragment that contains tracked attribute changes, the following cases are possible:

- if you perform the paste operation with **Track Changes** enabled, the changes are accepted before the paste operation;
- if you perform the paste operation with **Track Changes** disabled, the changes are pasted in the document.

Track Changes Limitations

Recording changes has limitations and there is no guarantee that rejecting all changes will return the document to exactly the same state in which it originally was. Recorded changes are not hierarchical, a change cannot contain other changes inside. For example, if you delete an insertion made by another user, then reject the deletion, the information about the author who made the previous insertion is not preserved.

Track Changes Markup

Depending on the type of your edits, the following track changes markup appears in a document when you activate the **Track Changes** feature:

Edit Type	Processing Instruction Start Marker	Processing Instruction End Marker	Attributes
Insertion	`<?oxy_insert_start?>`	`<?oxy_insert_end?>`	`author`, `timestamp`
Split	`<?oxy_insert_start?>`	`<?oxy_insert_end?>`	`author`, `timestamp`, `type="split"`
Surround	`<?oxy_insert_start?>`	`<?oxy_insert_end?>`	`author`, `timestamp`, `type="surround"`
Deletion	`<?oxy_delete?>`	_	`author`, `timestamp`, `content`
Comment	`<?oxy_comment_start?>`	`<?oxy_comment_end?>`	`author`, `timestamp`, `comment`, `mid`

Edit Type	Processing Instruction Start Marker	Processing Instruction End Marker	Attributes
Attribute Change	`<?oxy_attributes?>`	_	`id`, `type`, `oldValue`, `author`, `timestamp`

In case a comment intersects another, the `mid` attribute is used to correctly identify start and end processing instruction markers.

Intersecting Comments Markup

```
<?oxy_comment_start author="Andrew" timestamp="20130111T151520+0200" comment="Do we have a
task about pruning trees?"?>Unpruned
    <?oxy_comment_start author="Matthew" timestamp="20130111T151623+0200" comment="What time
of the year do they flower?" mid="3"?>lilacs<?oxy_comment_end?>
        flower reliably every year<?oxy_comment_end mid="3"?>
```

Managing Comments

A comment is marked in the **Author** mode with a background that is configured for each user name.
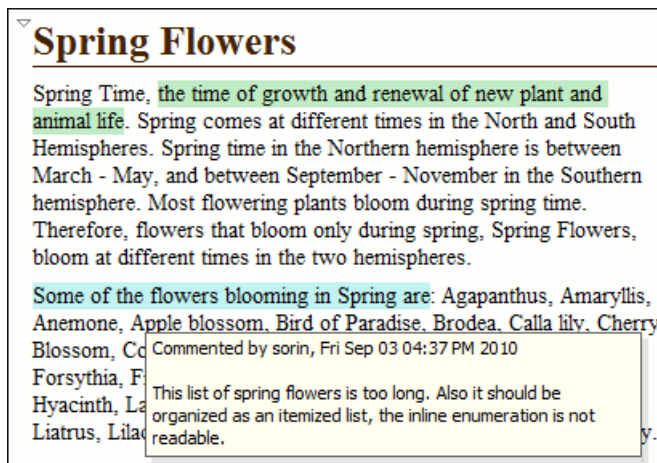
Figure 53: Manage Comments in Author Editor

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 in the Author toolbar.

Edit Comment...

Allows you to change an existing content. The action is available both in the Author toolbar and the contextual menu.

Remove Comment(s)...

Removes the comment at the cursor position or all comments found in the selected content. The action is available in the Author contextual menu, **Review** sub-menu.

Managing Highlights

Use the **Highlight** tool to mark the text in your document using different colours.

You can find the **Highlight** option on the main toolbar, in the **Edit** > **Review** menu, or in the contextual menu of a document, in the **Review** st of options.

What do you want to do?

- *Mark selected text*;
- *Mark fragments of the document you are editing*;
- *Remove highlighting*.

    Tip: In case the   **Highlight** tool is not available on your toolbar, enable **Author Comments** in the contextual menu of the toolbar.

    Note: Oxygen XML Author keeps the highlighting of a document between working sessions.

To watch our video demonstration about using the **Highlight** tool, go to *http://oxygenxml.com/demo/Highlight_Tool.html*.

Mark Selected Text

To mark the text you select in a document:

1. Select the text you want to highlight.

       Note: To mark more than one part of the document you are editing, press and hold **Ctrl (Meta on Mac OS)** and using you cursor select the parts you want to highlight.

2. Click the small arrow next to the   **Highlight** icon and select the colour that you want to use for highlighting. The selected text is highlighted.

3. Click the **Highlight** icon to exit the highlighting mode.

Mark Document Fragments

To mark fragments in a document, follow these steps:

1. Click the   **Highlight** icon on the toolbar.
   The highlighting mode is on. The cursor changes to a dedicated symbol that has the same color with the one set in the **Highlight** palette.

2. Select the text you want to highlight with your cursor.

3. To highlight different fragments using multiple colors, click the small arrow next to the   **Highlight** icon, choose the colour that you want to use for highlighting, and repeat **step 2**.
   The fragments are highlighted.

4. To exist the highlighting mode, press **Esc** on your keyboard, click the   **Highlight** icon, or start editing the document.

Remove Highlighting from the Entire Document or Part of It.

To remove highlighting from the document you are editing, follow these steps:

1. Either select the text you want to remove highlighting from using your cursor, or press **Ctrl+A (Command+A on OS X)** in case you want to select all of the text.

2. Click the small arrow next to the   **Highlight** icon and select **No color (erase)**, or right click the highlighted content and select **Remove highlight(s)**.
   The highlighting is removed.

3. Click the **Highlight** icon to exit the highlighting mode.

Author Callouts

A *callout* is a vertical stripe, with a balloon-like look, that Oxygen XML Author displays in the right side of the editing area. Callouts are decorated with a colored border and also have a colored background. A horizontal line, which has the same color as the border, connects text fragments with their corresponding callouts. Oxygen XML Author assigns an individual color for the callouts depending on the user who is editing the document. To customize the list of these colors, *open the **Preferences** dialog* and go to **Editor** > **Edit Modes** > **Author** > **Review**. You are able to add, edit, or remove colors in this list. You can choose to use the same color for any user who modifies the content or inserts a comment. To

do this, select the **fixed** option and choose a color from the color box. Once you set a fixed color for a user you are able to edit it. Press the color box and select a different color from the **Choose color** dialog box.

Oxygen XML Author uses callouts to provide an enhanced view of the changes you, or other authors make to a document. They hold specific information depending on their type. In addition, Oxygen XML Author uses callouts to display *comments* that you associate with fragments of the document you are editing. For more information about editing comments, go to *Managing Comments*. To enable callouts, *Open the Preferences dialog* and go to **Editor** > **Edit Modes** > **Author** > **Review** > **Callouts**. Enable the following options:

- **Comments** - Oxygen XML Author displays comment callouts when you insert a comment. You can use two types of comments in Oxygen XML Author:

  - author review comments: comments that you associate with specific fragments of text;
  - change comments: comments that you add in an already existing insertion or deletion callout.

  By default, the fragment of text that you comment is highlighted and a horizontal line connects it with the comment callout. A comment callout contains the name of the author who inserts the callout and the comment itself. To customize how comments are displayed, *open the Preferences dialog* , go to **Editor** > **Edit Modes** > **Author** > **Review** > **Callouts**, and enable **Show review time**;
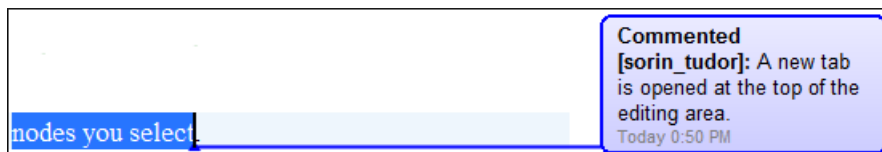
Figure 54: Comment Callouts

- **Track Changes deletions** - Oxygen XML Author displays deletion callouts when you delete a fragment of text. By default, a deletion callout contains the type of callout (*Deleted*) and the name of the author that makes the deletion. You are able to customize the content of a deletion callout to display the date and time of the deletion and the deleted fragment itself. To do this, *open the Preferences dialog* , go to **Editor** > **Edit Modes** > **Author** > **Review** > **Callouts**, and enable **Show review time** and **Show deleted content in callout**;
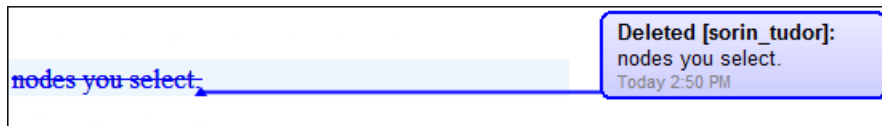
Figure 55: Deletion Callouts

- **Track Changes insertions** - Oxygen XML Author displays insertion callouts when you insert a fragment of text. By default, an insertion callout contains the type of callout (*Inserted*) and the name of the author that makes the insertion. You are able to customize the content of an insertion callout to contain the date and time of the insertion and the inserted fragment itself. *Open the Preferences dialog* and go to **Editor** > **Edit Modes** > **Author** > **Review** > **Callouts** and enable **Show review time** and **Show inserted content in callout**.
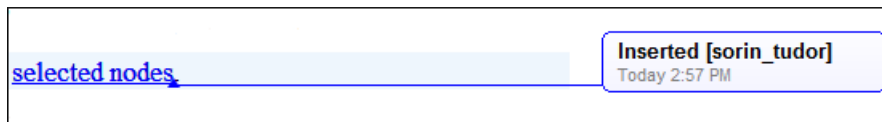
Figure 56: Insertion Callouts
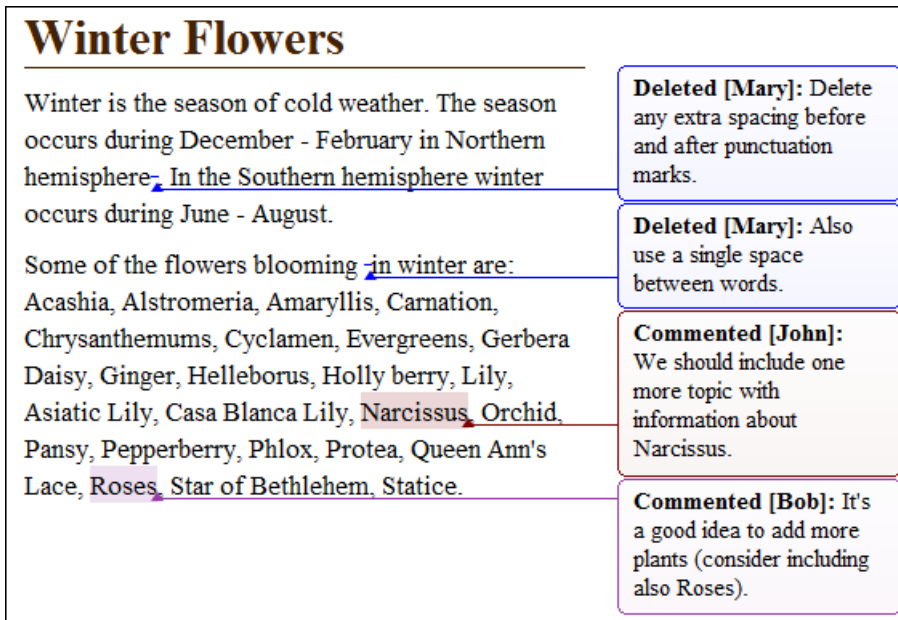
Figure 57: Multiple Authors Callouts

📝 **Note:** Oxygen XML Author displays callouts only if 🗐 **View All Changes/Comments** or **View Only Changes/Comments by** is selected. Oxygen XML Author does not display callouts in 🗐 **View Final** and 🗐 **View Original** modes.

To select a callout, either click the callout or its source. Selected callouts have a more intense background and a bold border. The connecting line between the source and the callout is also rendered in bold font. If you select a fragment of text which is associated with one or more callouts, the callouts are highlighted.

⚠ **Important:** The callouts are displayed in the right side of the editing area. However, in some cases, the text you are editing can span into the callouts area. For example, this situation can appear for callouts associated with wide images or space-preserve elements (like *codeblock* in DITA or *programlisting* in DocBook) which contain long fragments. To help you view the text under the covered area, Oxygen XML Author applies transparency to these callouts. When the caret is located under a callout, the transparency is enhanced, allowing you to both edit the covered content and access the contextual menu of the editing area.
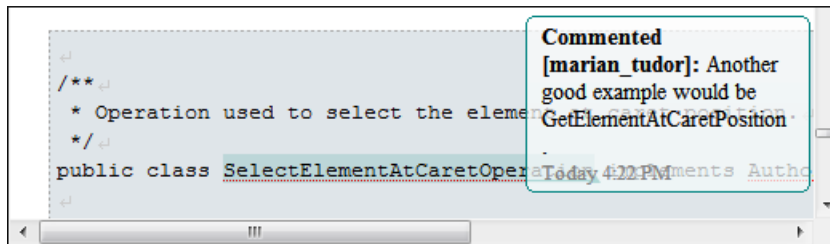
Figure 58: Transparent Callout

📝 **Note:** Oxygen XML Author does not display callouts located in folded areas of the edited document.

The following actions are available in the contextual menu of an insertion, or deletion callout:

- **Accept Change** - select this option to accept the changes you or other authors make to a document;
- **Reject Change** - select this option to reject the changes you or other authors make to a document;
- **Comment Change** - select this option to comment an existing change in your document. You are also able to add a comment to a change from the **Comment Change** 📝 button available on the **Author Review** toolbar;

- **Edit Reference** - in case the fragment that contains callouts is a reference, use this option to go to the reference and edit the callout;
- **Callouts Options** - select this option to open the *preferences page* of the callouts.

The following options are available in the contextual menu of the comment callouts:

- **Edit Comment** - select this option to modify the content of a comment callout;

> 📋 **Note:** The text area is disabled if you are not the author which inserted the comment.

- **Remove Comment** - select this option to remove a comment callout;
- **Edit Reference** - in case the fragment that contains callouts is a reference, use this option to go to the reference and edit the callout;
- **Callouts Options** - select this option to open the *preferences page* of the callouts.

When you print a document from Oxygen XML Author, all callouts you, or other authors added to the document are printed. For a preview of the document and its callouts, go to **File** > **Print preview...**.

To watch our video demonstration about the Callouts support, go to *http://oxygenxml.com/demo/CalloutsSupport.html*.

The Review View

The **Review** view is a framework-independent panel, available both for built-in, and custom XML document frameworks. It is designed to offer an enhanced way of monitoring all the changes that you make to a document. This means you are able to view and control highlighted, commented, inserted, and deleted content, or even changes made to attributes, using a single view.

The **Review** view is useful when you are working with documents that contain large quantities of edits. The edits are presented in a compact form, in the order they appear in the document. Each edit is marked with a type-specific icon.
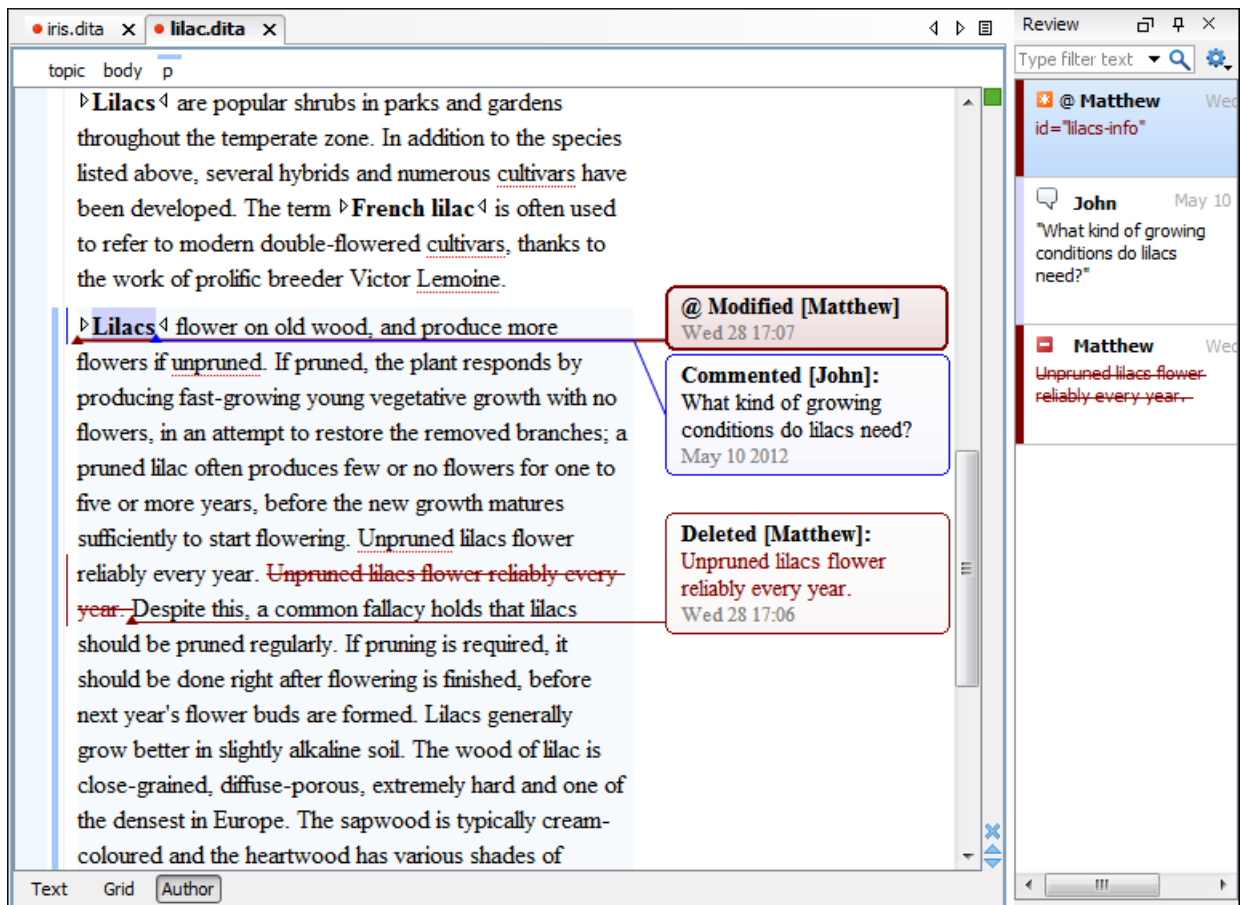
Figure 59: The Review View

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

- click the **Manage reviews** button on the **Author Review** toolbar;
- right click in a document and from the contextual menu go to **Review**, **Manage reviews**;
- go to **Window** > **Show View** > **Review**.

This view and the editing area are synchronized. When you select an edit listed in the **Review** view, its corresponding fragment of text is highlighted in the editing area and. The reverse is also true. For example, when you place the caret inside an area of text marked as inserted, its corresponding edit is selected in the list.

The upper part of the view contains a filtering area which allows you to search for specific edits. Use the small arrow symbol from the right side of the search field to display the search history. The **Settings** button allows you to:

- **Show highlights** - controls whether the **Review** view displays the highlighting in your document;
- **Show comments** - controls whether the **Review** view displays the comments in the document you are editing;
- **Show track changes** - controls whether the **Review** view displays the inserted and deleted content in your document;
- **Show review time** - displays the time when the edits from the **Review** view were made.

The following actions are available when you hover the edits in the **Review** view, using the cursor:

Remove

Action available for highlights and comments presented in the **Review** view. Use this action to remove these highlights or comments from your document;

Accept

Action available for inserted and deleted content presented in the **Review** view. Use this action to accept the changes in your document;

Reject

Action available for inserted and deleted content presented in the **Review** view. Use this action to reject the changes in your document.

Depending on the type of an edit, the following actions are available in its contextual menu in the **Review** view:

Show comment

This option is available in the contextual menu of changes not made by you and of any comment listed in the **Review** view. Use this option to view a comment in the **Show comment** dialog.

Edit comment

This option is available in the contextual menu of your comments, listed in the **Review** view. Use this action to start editing the comment.

Remove comment

This option is available in the contextual menu of a comment listed in the **Review** view. Use this action to remove the selected comment.

Show only reviews by

This option is available in the contextual menu of any edit listed in the **Review** view. Use this action to keep visible only the edits of a certain author in the view.

Remove all comments

This option is available in the contextual menu of any comment listed in the **Review** view. Use this action to remove all the comments that appear in the edited document.

Change color

Opens a palette that allows you to choose a new color for the highlighted content.

Remove highlight

Removes the selected highlighting.

Remove highlights with the same color

Removes all the highlighting with the same color from the entire document.

Remove all highlights

Clears all the highlighting in your document.

Accept change

Accepts the selected change.

Reject change

Rejects the selected change.

Comment change

This option is available in the contextual menu of an insertion or deletion that you made. Use this option to open the **Edit comment** dialog and comment the change you made.

Accept all changes

Accepts all the changes made to a document.

Reject all changes

Rejects all the changes made to a document.

To watch our video demonstration about the **Review** view, go to *http://oxygenxml.com/demo/Review_Panel.html*.

Profiling / Conditional Text

Conditional text is a way to mark blocks of text meant to appear in some renditions of the document, but not in others. It differs from one variant of the document to another, while unconditional text appear in all document versions.

For instance you can mark a section of a document to be included in the manual designated for the *expert* users, other for the *novice* users manual while unmarked sections are included in any rendition.

You can use conditional text when you develop documentation for:

• a series of similar products
• different releases of a product
• various audiences

The benefits of using conditional text include reduced effort for updating and translating your content and an easy way to customize the output for various audiences.

Oxygen XML Author 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 Preferences dialog* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text** .
2. In the **Profiling Attributes** area, press the **New** button.

The following dialog is opened:
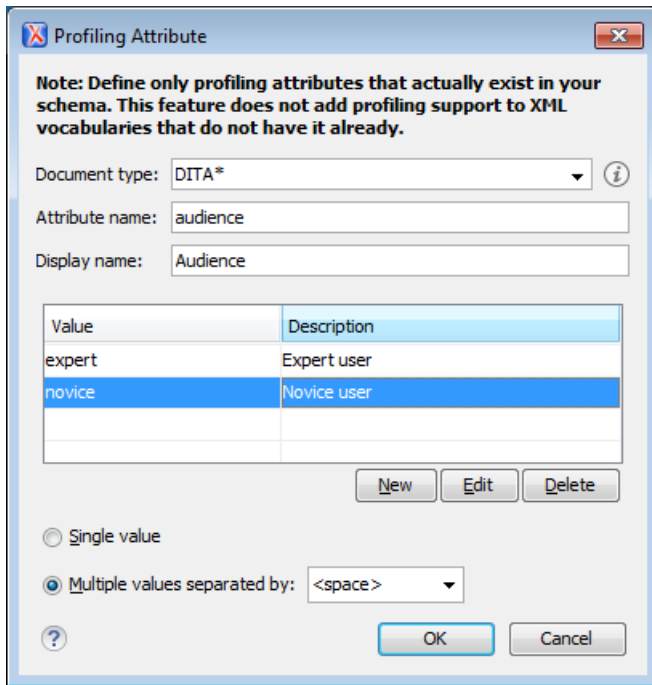
3. Fill-in the dialog as follows:

   a) Choose the document type on which the profiling attribute is applied. ***** and **?** are used as wildcards, while **,**(comma character) can be used to specify more patterns. For example use *DITA** to match any document type name that starts with *DITA*.

   b) Set the attribute name.

   c) Set a display name. This field is optional, being used only as a more descriptive rendering in application's profiling dialogs.

   d) Use the **New**, **Edit**, **Delete** buttons to add, edit and delete possible values of the attribute. Each attribute value can have a description.

   e) Choose whether the attribute accepts a single value (**Single value** option checked) or multiple values. Multiple values can be separated by a default delimiter (*space*, *comma*, *semicolon*), or a custom one, that must be supported by the specified document type. For example, the DITA document type only accepts spaces as delimiters for attribute values.

4. Click **OK**.

5. Click **Apply** to save the profiling attribute.

Create Profiling Condition Sets

Several profiling attributes can be aggregated into a profiling condition set that allow you to apply more complex filters on the document content. A Profiling Condition Set is a very powerful and convenient tool used to preview the content that goes into the published output. For example, an installation manual available both in Windows and Linux variants can be profiled to highlight only the Linux procedures for more advanced users.

To create a new profiling condition set:

1. *Open the Preferences dialog* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text**.

2. In the **Profiling Condition Sets** area, press the **New** button.

   The following dialog is opened:
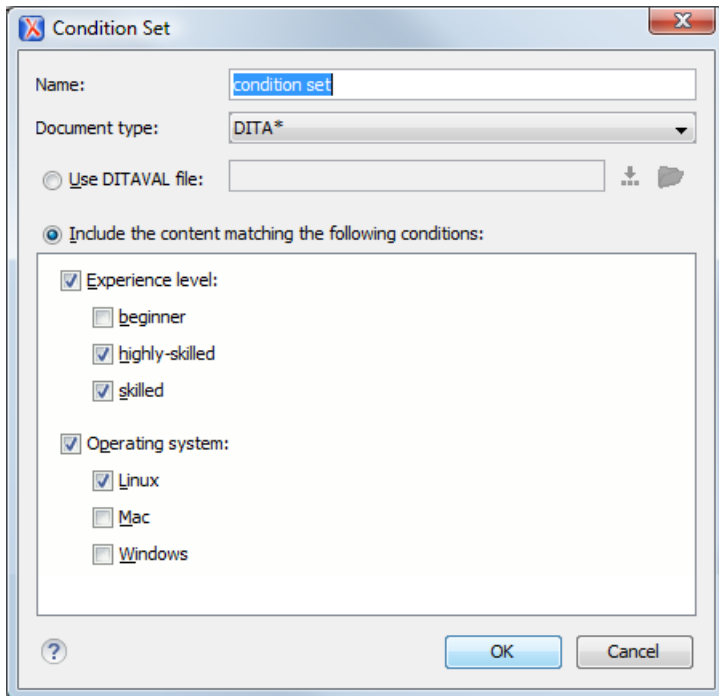
3. Fill-in the dialog as follows:

   a) Type the condition set's name.

      If you want the Profiling Condition Set to refer a DITAVAL file, enable the **Use DITAVAL file** option and select the DITAVAL file from your disk.

   b) Choose the document type for which you have previously defined profiling attributes.
      After choosing a document type, all profiling attributes and their possible values are listed in the central area of the dialog.

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

4. Click **OK**.

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

Apply Profiling Condition Sets

All defined Profiling Condition Sets are available as shortcuts in the Profiling / Conditional Text menu. Just click on a menu entry to apply the condition set. The filtered content is grayed-out in Author editor, Outline view and DITA Maps Manager view. An element is filtered-out when one of its attributes is part of the condition set and its value does not match any of the value covered by the condition set. As an example, let us suppose that you have the following document:

If you apply the following condition set it means that you want to filter-out the content written for non-expert audience and having the *Other* attribute value different than *prop1*.
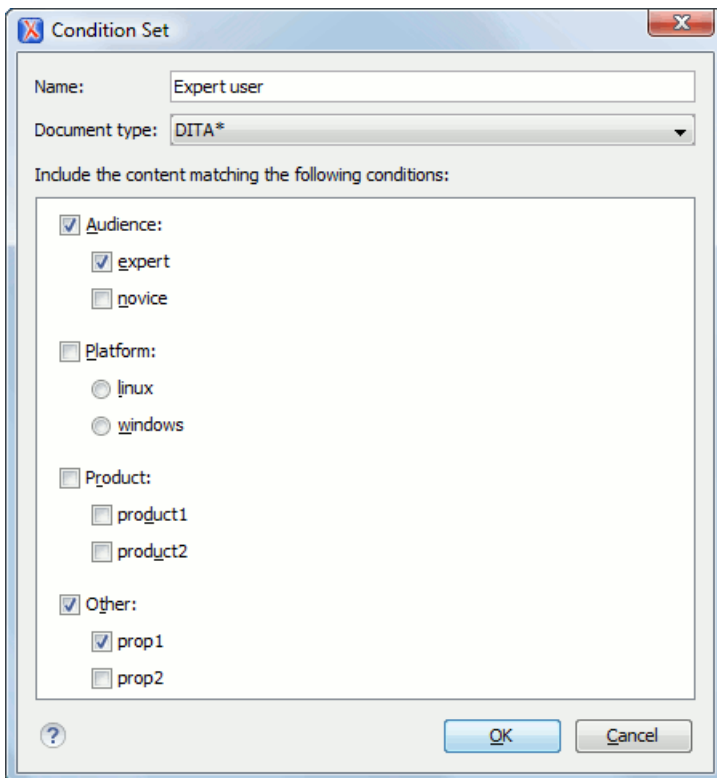
And this is how the document looks like after you apply the *Expert user* condition set:

Apply Profiling Attributes

Profiling attributes are applied on element nodes.

You can apply profiling attributes on a text fragment, on a single element, or on multiple elements in the same time. To profile a fragment from your document, select the fragment in the **Author** mode and follow these steps.

> **Note:** If there is no selection in your document, the profiling attributes are applied on the element at caret position.

1. Invoke the **Edit Profiling Attributes...** action from the contextual menu.

    The displayed dialog shows all profiling attributes and their values, as defined on the document type of the edited content. The checkboxes corresponding with the values already set in the profiled fragment are enabled.

2. In the **Edit Profiling Attributes** dialog, enable the checkboxes corresponding to the attribute values you want to apply on the document fragment. The profiling attributes having different values set in the elements of the profiled fragment are marked with a gray background and they are disabled by default. You can change the values of these attributes by choosing the **Change Now** option associated with all attributes.

3. Click **OK** to finish the profiling configuration.

    The attributes and attributes values selected in the **Edit Profiling Attributes** dialog are set on the elements contained in the profiled fragment.

    If you select only a fragment of an element's content, this fragment is wrapped in phrase-type elements on which the profiling attributes are set. Oxygen XML Author comes with predefined support for DITA and DocBook. For more developer-level customization options, see the *Customize Profiling Conditions* topic.

    If **Show Profiling Attributes** option (available in the ▼⃥ *Profiling / Conditional Text toolbar menu*) is set, a light green border is painted around profiled text, in the **Author** mode. Also, all profiling attributes set on the current element are listed at the end of the highlighted block and in its tooltip message. To edit the attributes of a profiled fragment, click one of the listed attributes. A form control pops up and allows you to add or remove attributes using their checkboxes.

Profiling / Conditional Text Menu

The ▼⃥ **Profiling / Conditional Text** toolbar menu groups the following actions:

Show Profiling Colors and Styles

Enable this option to turn on conditional styling.

Show Profiling Attributes

Enable this option to turn on conditional text markers. They are displayed at the end of conditional text blocks, as a list of attribute name and their currently set values.

Show Excluded Content

Controls if the content filtered out by a particular condition set is hidden or greyed-out in the editor area and in the **Outline** and **DITA Maps Manager** views. When this option is turned on, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, turn off this option.

> 📄 **Note:** To remind you that document content is hidden, Oxygen XML Author displays labels showing the currently applied condition set. These labels are displayed in the Author editing area, the **Outline** view and **DITA Maps Manager** view. Right click any of the labels to quickly access the **Show Excluded Content** action.

List of all profiling condition sets that match the current document type

Click on a condition set entry to activate it.

⚙ **Profiling Settings...**

Link to the *profiling options* preference pages, where you can manage profiling attributes, profiling conditions sets, as well as profiling styles and colors options.

All these settings are associated with the current project, being restored the next time you open it. For a new project all Profiling/Conditional Text menu actions states are reset to their default values.

Apply Profiling Colors and Styles

Applying profiling colors and styles allows you to customize the Author editing area to mark profiled content so you can instantly spot different variants of the output.

Mowing equipment needs regular checks and maintenance. Monthly, you should:

- Refill the oil:
    - Remove the oil fill cap;
    - Pour new oil gradually. Regularly check the dipstick to see if the oil level reached the maximum mark;

    product [Gasoline]

- Sharpen the blades:
    - Clamp the blade to a vice or to the edge of a solid surface;
    - Using ▷ an electric grinder or ◁ audience [ExpertUser] a file, grind the length of the blade until it is sharp;

- Check the electric cable for any signs of wear. Replace it if worn; product [Electric]
- Clean the mower's underside for debris; product [Electric, Gasoline]
- Inspect the general state of the mower. Use a ratchet to tighten any loose bolts;
- Lubricate the gears of the manual lawn mower; product [Manual]

Choosing the right style for a specific profiling attribute is a matter of personal taste, but you should keep in mind that:

- if the same block of text is profiled with two or more profiling attributes, their associated styles combine. Depending on the styling, this might result in an excessively styled content that may prove difficult to read or work with;
- profile only differences. There is no need to profile common content, since excessive profiling can visually pollute the document;
- a mnemonic associated with a style will help you spot instantly different types of content.

To set colors and styles to profiling attribute values:

- enable the **Show Profiling Colors and Styles** option from the *Profiling / Conditional Text toolbar menu*;

- go to **Profiling Settings** from the *Profiling / Conditional Text toolbar menu*. This is a shortcut to the *Profiling/Conditional Text options page*. Select the *Colors and Styles options page*;

- set a style to a profiling attribute value.

Note that the styling is now applied in the Author editor, the Outline view and DITA Maps Manager view. Also, to help you identify more easily the profiling you want to apply in the current context, the styling is applied in the **Edit Profiling Attributes** dialog and in the inline form control that allows you to quickly set the profiling attributes.

Smart Paste Support

You can paste content from various sources, such as web pages and office-type documents, and paste it into DITA, TEI, DocBook, and XHTML documents. Oxygen XML Author keeps the original text styling (like bold, italics) and formatting (like lists, tables, paragraphs), and helps you make the resulting document valid.

You can paste content from:

- Office applications (**Microsoft Word** and **Microsoft Excel**, **OpenOffice.org Writer** and **OpenOffice.org Calc**).
- Web browsers.
- The Oxygen XML Author **Data Source Explorer** view (where resources are available from WebDAV or CMS servers).

You can paste it into:

- *DITA*
- *DocBook 4*
- *DocBook 5*
- *TEI 4*
- *TEI 5*
- *XHTML*

The styles and general layout of the pasted content are transformed to the equivalent XML markup of the target document type.

Tables pasted in a DocBook file are automatically converted to CALS. If you want to overwrite this behaviour and instruct Oxygen XML Author to convert them to HTML tables, set the `docbook.html.table` parameter to `1`. You can find this parameter in:

- `[OXYGEN_DIR]/frameworks/docbook/resources/xhtml2db5Driver.xsl` stylesheet, for DocBook 5
- `[OXYGEN_DIR]/frameworks\docbook\resources\xhtml2db4Driver.xsl` stylesheet, for DocBook 4

You can disable smart paste by deselecting *Convert external content on paste* in the **Schema Aware** preferences.

If you paste the content in a location where the resulting XML would not be valid, Oxygen XML Author will attempt to place it in a valid location, and may prompt you with one or more choices for where to place it.

You can disable this location selection feature by deselecting *Smart paste and drag and drop* option, available in the **Schema Aware** preferences.

To watch our video demonstration about the Smart Paste support, go to *http://oxygenxml.com/demo/Smart_Paste_Copy_Paste_from_Web_Office_Documents_to_DITA_DocBook_TEI_XHTML_Documents.html*.

Bidirectional Text Support in Author Mode

Oxygen XML Author offers support for languages that require right to left scripts. This means that authors editing documents in the **Author** mode are able to create and edit XML content in Arabic, Hebrew, Persian and others. To achieve this, Oxygen XML Author implements the *Unicode Bidirectional Algorithm* as specified by the Unicode consortium. The text arrangement is similar to what you get in a modern HTML browser. The final text layout is rendered according with the directional CSS properties matching the XML elements and the Unicode directional formatting codes.

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

Controlling the Text Direction Using XML Markup

Oxygen XML Author Supports the following CSS properties:

Table 4: CSS Properties Controlling Text Direction

`direction`	Specifies the writing direction of the text. The possible values are `ltr` (the text direction is left to right), `rtl` (the text direction is right to left, and `inherit` (specifies whether the value of the direction property is inherited from the parent element).
`unicodeBidi`	Used with the `direction` property, sets or returns whether the text is overridden to support multiple languages in the same document. The possible values of this property are `bidi-override` (creates an additional level of embedding and forces all strong characters to the direction specified in the `direction`), `embed` (creates an additional level of embedding), `normal` (does not use an additional level of embedding), and `inherit` (the value of the `unicodeBidi` property is inherited from parent element).

For instance, to declare an element as being Right to Left, you could use a stylesheet like the one below:

XML File:

```
<article>
  <myRTLpara>RIGHT TO LEFT TEXT</myRTLpara>
</article>
```

Associated CSS File:

```
myRTLpara{
    direction:rtl;
    unicode-bidi:embed;
}
```

Oxygen XML Author recognizes the `dir` attribute on any XML document. The supported values are:

`ltr`	The text from the current element is Left to Right, embedded.
`rtl`	The text from the current element is Right to Left, embedded.
`lro`	The text from the current element is Left to Right, embedded.
`rlo`	The text from the current element is Right to Left, embedded.

The following XML document types make use of the `dir` attribute with the above values:

- DITA
- DocBook
- TEI

- XHTML

📄 **Note:** When the inline element tags are visible, the text in the line is arranged according to the BIDI algorithm after replacing the tags symbols with Object Replacement Characters. This makes it possible to get a different text arrangement when viewing a document in the **No Tags** mode versus viewing it in the **Full Tags** mode.

Controlling the Text Direction Using the Unicode Direction Formatting Codes

These Unicode Direction Formatting Codes codes can be embedded in the edited text, specifying a text direction and embedding. However, it is not recommended to use them in XML as they are zero width characters, making it hard to debug the text arrangement.

Table 5: Directional Formatting Codes

U+202A (LRE)	LEFT-TO-RIGHT EMBEDDING	Treats the following text as embedded left-to-right.
U+202B (RLE)	RIGHT-TO-LEFT EMBEDDING	Treats the following text as embedded right to left.
U+202D (LRO)	LEFT-TO-RIGHT OVERRIDE	Forces the following characters to be treated as strong left-to-right characters.
U+202E (RLO)	RIGHT-TO-LEFT OVERRIDE	Forces the following characters to be treated as strong right-to-left characters.
U+202C (PDF)	POP DIRECTIONAL FORMATTING CODE	Restores the bidirectional state to what it was before the last LRE, RLE, RLO, or LRO.
U+200E (LRM)	LEFT-TO-RIGHT MARK	Left-to-right strong zero-width character.
U+200F (RLM)	RIGHT-TO-LEFT MARK	Right-to-left strong zero-width character.

To insert Unicode Direction Formatting Codes, use the *Character Map* dialog. To easily find such a code, you can either enter directly the hexadecimal value, or use the **Details** tab to enter the codes name.

Oxygen XML Author offers the support for bi-directional text in all the side views (**Outline** view, **Attributes** view and so on) and text fields.

Chapter

6

Editing Documents

Topics:

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

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

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

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

Opening and Saving Unicode Documents

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

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

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

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

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

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

The byte order and the BOM of an existing document are preserved when the document is edited and saved. This behavior can be changed in Oxygen XML Author from the *Encoding preferences panel*.

Inserting Symbols

You can insert symbols either using the **Symbols** action from the toolbar or using the **Edit** > **Insert from Character Map...** action.

If the **Symbols** action is not available on the main toolbar, right click the toolbar and from the contextual menu select

Symbols. Clicking the Ω action displays a menu containing:

A list of recently used symbols

Click any item in the list to insert a symbol at caret position.

More symbols... action

Provides access to the **Character Map** dialog.

Clear all symbols action

Click this action to delete the list of recently used symbols.

The Character Map dialog

The **Character Map** dialog allows you to visualize all characters available in a font, pick the character you need and insert it in the document you are editing. You can also access this dialog if you use the **Edit** > **Insert from Character Map...** action.

Use the **Font** box to choose the font for which you want to display characters. To see only a certain range of characters, use the **Unicode Block** box. This will filter the number of characters displayed, showing only a contiguous range of characters corresponding to the selected block. Unassigned characters are displayed as empty squares.

The available characters are listed in two tabs:

*   **Compact** - matrix-like representation which displays only characters.
*   **Details** - displays the available characters in a tabular format, presenting their decimal and hexadecimal value along with their description.

Use the **Search** field to search for a character by one of the following attributes:**decimal** value, **hexadecimal** value or description. Selecting the **description** option places the focus on the **Details** tab. In case you enter a character description in the **Search** field, the **description** option is selected automatically. The searching operation starts as soon as you start typing characters in the **Search** field.

Press the **Insert** button to insert the selected character in the current editor at caret position. You will see the character in the editor if *the editor font* is able to render it. The **Copy** button copies it to the clipboard without inserting it in the editor. You can see the name and range name of a character either at the bottom of the dialog, or when hovering the mouse cursor over the character.

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

Creating, Opening, and Closing Documents

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

Creating Documents

This section details the procedures available for creating new documents.

The New Document Wizard

Oxygen XML Author supports a wide range of document types. The **New Document** wizard presents the default associations between a file extension and the type of editor that opens the file. To customize these default associations, *open the **Preferences** dialog* and go to *File Types*.

1. To create a document in Oxygen XML Author, either select **File** > **New** > **Ctrl+N (Command+N on OS X)**, or

   click the ☐ **New** button on the toolbar.
   Oxygen XML Author displays the **New Document** wizard and groups the supported document types in multiple categories:

   - **Recently used** - contains the list of the most recently used files;
   - **New Document** - contains the list of all supported document types. This list includes XML, CSS, Text, PHP, JavaScript.
   - **Global templates** - contains the list of predefined templates as well as templates defined in the *Document Templates* preferences page.
   - **Framework templates** - contains the list of templates defined in the *Document Type Association* preferences page, **Templates** tab.

2. Select a document type.

3. Click one of the following:

   - **Customize** - action available only for XML, XML Schema, Schematron, and XSL documents. Depending on the document type, you can set different properties before you create the file.
   - **Create** - uses default settings to create a file.

   If you select **Create**, Oxygen XML Author opens the new file in the editor view.

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

   -
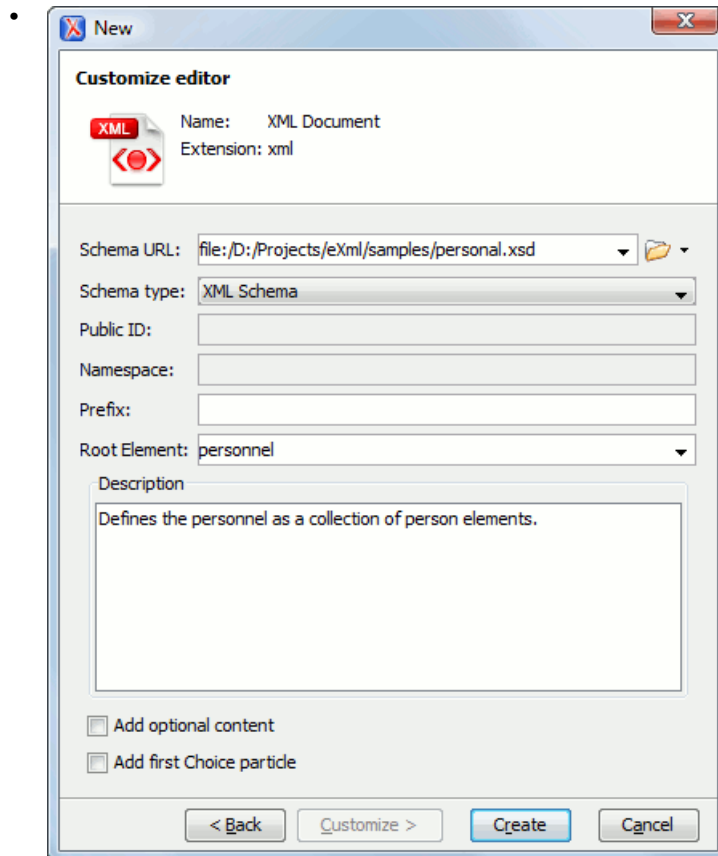
   Figure 60: New XML Document Dialog Box

   - **Schema URL** - specifies the path to the schema file. When you select a file, Oxygen XML Author analyzes its content and tries to fill the rest of the dialog box;

- **Schema type** - allows you to select the schema type. The following options are available: XML Schema, DTD, RelaxNG XML syntax, RelaxNG compact syntax, and NVDL;
- **Public ID** - specifies the PUBLIC identifier declared in the document prolog;
- **Namespace** - specifies the document namespace;
- **Prefix** - specifies the prefix for the namespace of the document root;
- **Root Element** - populated with elements defined in the specified schema, enables selection of the element used as document root;
- **Description** - shows a small description of the selected document root;
- **Add optional content** - if you select this option, the elements, and attributes defined in the XML Schema as optional, are generated in the skeleton XML document;
- **Add first Choice particle** - if you select this option, Oxygen XML Author generates the first element of an `xs:choice` schema element in the skeleton XML document. Oxygen XML Author creates this document in a new editor panel when you click **OK**.

Creating Documents Based on Templates

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

The list of templates presented in the dialog includes:

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

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

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

Document Templates

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

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

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

Saving Documents

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

- **File** > **Save**;
- The ⊞ **Save** toolbar button. If the document was not saved yet it displays the **Save As** dialog;
- **File** > **Save As** - displays the **Save As** dialog, used either to name and save an open document to a file or to save an existing file with a new name;
- **File** > **Save To URL** - displays the **Save to URL** dialog, which can be used to save a file identified by its URL (defined by a protocol, host, resource path, and an optional port). Use the drop down action list to choose one of the available save actions:

  - 📂 **Browse for local file** - Opens a local file browser dialog box allowing you to save the document locally.
  - 📂 **Browse for remote file** - Displays the **Save to URL** dialog which allows you to save the document to a remote location (accessible through FTP, SFTP or WebDAV).

- Browse for archived file - Displays the **Archive Browser Dialog**, which allows you to save the document inside an archive.
- Browse Data Source Explorer - Opens the **Data Source Explorer** which allows you to browse the data sources defined in the *Data Sources preferences page*.

  > **Tip:** You can get to the **Data Sources** preferences page, using the **Configure Database Sources** shortcut from the **Save to URL** dialog.

- Search for file - Displays the *Open/Find Resources dialog*.

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

Opening/Navigating Documents

To open a document in Oxygen XML Author, do one of the following:

- Go to **File** > **Open** <u>Ctrl+O (Command+O on OS X)</u>, to display the **Open** dialog. The start folder of the **Open** dialog can be either the last folder visited by this dialog or the folder of the currently edited file. This can be *configured in the user preferences.*
- Click the Open toolbar button to display the **Open** dialog.
- Go to **File** > Open URL ... or click the Open URL ... toolbar button to display the **Open URL** dialog which allows you to access any resource identified through an URL (defined by a protocol, host, resource path, and an optional port). The following actions are available in the drop down action list:

  - Browse for local file - Opens a local file browser dialog box allowing you to select a local file.
  - Browse for remote file - Displays *the Open using FTP/SFTP dialog* which allows you to open a remotely stored document.
  - Browse for archived file - Displays the **Archive Browser Dialog**, which allows you to browse the content of an archive a choose a file to open in Oxygen XML Author.
  - Browse Data Source Explorer - Opens the **Data Source Explorer** which allows you to browse the data sources defined in the *Data Sources preferences page*.

    > **Tip:** You can get to the **Data Sources** preferences page, using the **Configure Database Sources** shortcut from the **Open URL** dialog.

  - Search for file - Displays the *Open/Find Resources dialog*.

- Click the Open URL ... toolbar button to run the same action.
- Click the Open/Find Resource ... toolbar button to run the same action.
- Go to **File** > **Revert** to load the last saved file content. All unsaved modifications are lost.
- Go to **File** > **Reopen** to reopen one of the recently opened document files. The list containing recently opened files can be emptied by invoking the **Clear history** action.
- Select the **Open** action from the contextual menu of the **Project** view. This opens the selected file from the **Project** view.

The Open/Find Resource View

The **Open/Find Resource** view is designed to offer advanced search capabilities either by using a simple text search or by using the *Apache Lucene - Query Parser Syntax*. To open this view, go to **Window** > **Show View** > **Open/Find Resource**. The view is presented in the left side of the default Oxygen XML Author layout, next to the **Project** and **DITA Maps Manager** views.

Figure 61: The Open/Find Resource View

You can use this view to find a file in the current Oxygen XML Author project or in one of the DITA maps opened in *the DITA Maps Manager view* by typing only a few letters of the file name of a document or a fragment of the content you are searching for. The Open/Find Resource view also supports searching in document edits (comments, insertions, deletions, and highlighted content).

> **Note:** Full support for searching in document edits is available only in the Enterprise edition of Oxygen XML Author and Oxygen XML Editor. The Professional edition offers support to search through a maximum of 10 edits.

Search results are presented instantly, after you finish typing the content you are searching for. The matching fragments of text are highlighted in the results list displayed in the view. When you open one of the documents from the results list, the matching fragments of text are highlighted in the editing area. To remove the highlighting from your document close the *Results view*. To display the search history, position the caret in the search field and press **Ctrl+Down Arrow (Command+Down Arrow on OS X)** or **Ctrl+Up Arrow (Command+Up Arrow on OS X)** on your keyboard. Pressing only the **Down Arrow** key moves the selection to the list of results.

A contextual menu available on each search result provides actions applicable to the document that contains it. These actions allow you to:

- open the document in one of Oxygen XML Author's internal editors
- open the document in an external system application
- identify the document in the system file explorer
- copy the file's location

The content of the resources used to search in is parsed from an index. The indexing is performed both automatically and on request.

> **Note:** Automatic indexing is performed when you modify, add, or remove resources in the currently indexed project. In case the index was never initialized, the index in not updated on project changes.

> **Note:** To improve performance, the indexing process skips the following set of common English words (the so-called *stop words*): *a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the,*

their, then, there, these, they, this, to, was, will, with. This means that if you are searching for any of these words, **Open/Find Resource** view will not be able to match any of them. However, you can configure the list of *stop words* in the ***Open/Find Resources Preferences Page***.

Note: Searches are case insensitive. For example, if you search for *car* you get the same results as when you search for *Car*.

Note: Suffix searches are supported, both for searching in the content of your resources and in their name. For this, you can use wildcards. If you search **in content** for `*ing` you will find documents that contain the word *presenting*. If you search **in file paths** for **/samples/*.gif* you will find all the *gif* images from the `samples` directory.

Note: You can drag a resource from the **Open/Find Resource** view and drop it in a DocBook, DITA, TEI or XHTML document to create a link to that resource.

The **Open/Find Resource** view offers the following options:

- **Settings** - displays settings of the view;

  - **Clear Index** - clears the index;
  - **Show description** - presents the search results in a more compact form, displaying only the title and the location of the resources;
  - **Options** - opens the ***Open/Find Resource** preferences page*.

- ***In file paths*** - select this option to search for resources by their name or by its path (or a fragment of its path)
- ***In content*** - select this option to search through the content of your resources;
- ***In reviews*** - select this option to search through the comments, insertions, and deletion in your resources.
- **Reindex** - reindexes your resources.

The Open/Find Resource Dialog

To open the **Open/Find Resource** dialog, go to **Find** > **Open/Find Resource ... Ctrl+Shift+R (Command+Shift+R on OS X)**. You can also click the **Open/Find Resource ...** toolbar button or use the **Search for file** action, available for some URL input fields.
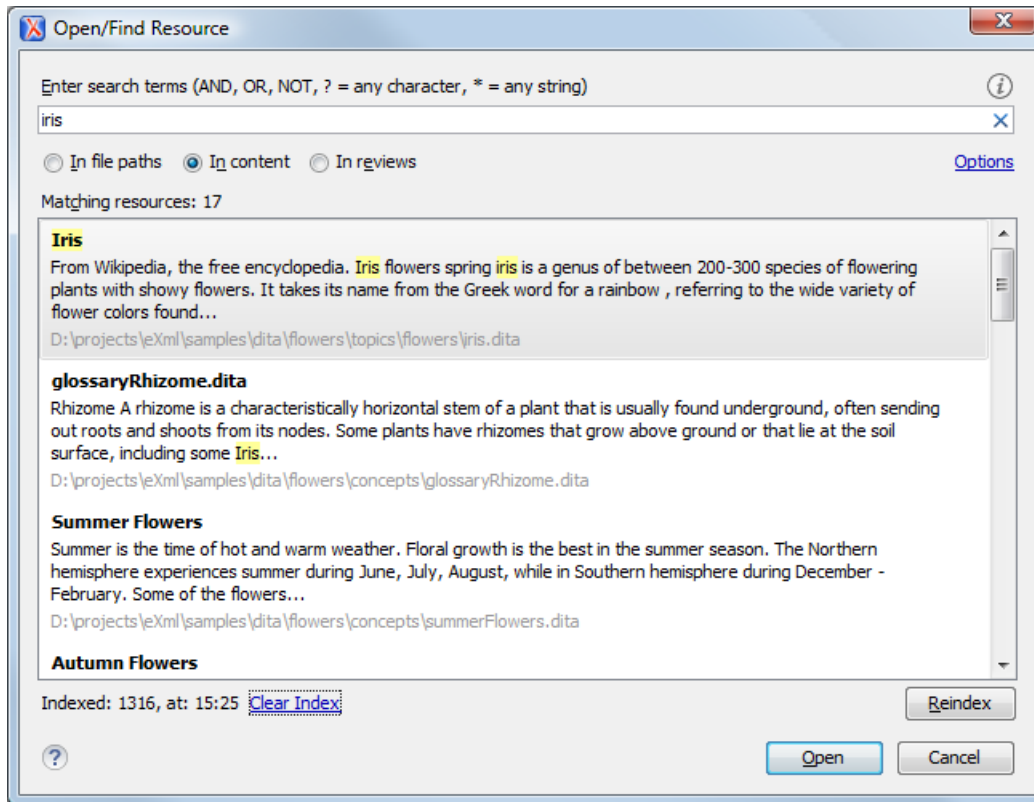
Figure 62: The Open/Find Resource Dialog

You can use this dialog to find a file in the current Oxygen XML Author project or in one of the DITA maps opened in *the DITA Maps Manager view* by typing a few letters of the file name of a document or a fragment of the content you are searching for. The Open/Find Resource dialog also supports searching in document edits (comments, insertions, deletions, and highlighted content).

> **Note:** Full support for searching in document edits is available only in the Enterprise edition of Oxygen XML Author and Oxygen XML Editor. The Professional edition offers support to search through a maximum of 10 edits.

Search results are presented instantly, after you finish typing the content. The matching fragments of text are highlighted in the results list displayed in the dialog. When you open one of the documents from the results list, the matching fragments of text are highlighted in the editing area. To remove the highlighting from your document close the *Results view*. To display the search history, position the caret in the search field and press **Ctrl+Down Arrow (Command+Down Arrow on OS X)** or **Ctrl+Up Arrow (Command+Up Arrow on OS X)** on your keyboard. Pressing only the **Down Arrow** key moves the selection to the list of results.

> **Note:** Searches are case insensitive. For example, if you search for *car* you get the same results as when you search for *Car*.

> **Note:** Suffix searches are supported, both for searching in the content of your resources and in their name. For this, you can use wildcards. If you search **in content** for `*ing` you will find documents that contain the word *presenting*. If you search **in file paths** for */samples/*.gif* you will find all the *gif* images from the `samples` directory.

The **Open/Find Resource** dialog offers the following options:

• *In file paths* - select this option to search for resources by their name or by its path (or a fragment of its path).
• *In content* - select this option to search through the content of your resources.
• *In reviews* - select this option to search through the comments, insertions, and deletion in your resources.

- **Options** - opens the *Open/Find Resource preferences page*.
- **Clear Index** - clears the index.
- **Reindex** - reindexes your resources.

A contextual menu available on each search result provides actions applicable to the document that contains it. These actions allow you to:

- open the document in one of Oxygen XML Author's internal editors
- open the document in an external system application
- identify the document in the system file explorer
- copy the file's location

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

⚠ **Important:** Files larger than 2GB are not indexed.

If there is no file found that matches your file pattern or text search, a possible cause is that the file you are searching for was added to the Oxygen XML Author project after the last caching operation. In this case, re-indexing the project files from the **Reindex** button enables the file to be found. The date and time of the last index operation are displayed below the file list.

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

To watch our video demonstration about the **Open/Find Resource** dialog and search capabilities, go to *http://oxygenxml.com/demo/Open_Find_Resource.html*.

Searching in Content

To perform a search through the content of your resources, open the **Open/Find Resources** dialog or the **Open/Find Resource** view, enable the **in content** option, and in the search field enter the terms that you want to search for.

The **Open/Find Resource** feature is powered by *Apache Lucene*. Apache Lucene is a free open source information retrieval software library.

You can use the **Open/Find Resource** dialog and the **Open/Find Resource** view either to perform a simple text search or to perform a more complex search using the *Apache Lucene - Query Parser Syntax*. Using the *Apache Lucene - Query Parser Syntax* means you can perform any of the following searches:

Term Searches

Use the **Open/Find Resource** view or dialog to search for plain text:

```
Garden Preparation
```

Element Specific Searches

Use the **Open/Find Resource** view or dialog to search for content that belongs to a specific element:

```
title:"Garden Preparation"
```

Wildcard Searches

Use wildcards to make your search more permissive:

```
Garden Prepar?tion
```

Fuzzy Searches

In case you are not sure what is the exact form of a term that you are interested in, use the fuzzy search to find the terms that are similar to what you introduce in the **Open/Find Resource** view or dialog. To perform a fuzzy search, use the ~ symbol after the word that you are not sure of:

```
Garden Preparing~
```

Proximity Searches

Use proximity searches to find words that are within a specific distance away. To perform a proximity search, use the ~ symbol at the end of your search. For example, to search for the word *Garden* and the word *Preparation* within 6 words of each other use:

```
"Garden Preparation"~6
```

Range Searches

Use range searches to match documents whose element values are between the lower and upper bound specified in the range query. For example, to find all documents whose titles are between *Iris* and *Lilac*, use:

```
title:{Iris TO Lilac}
```

The curly brackets denote an exclusive query. The results you get when using this query are all the documents whose titles are between *Iris* and *Lilac*, but not including *Iris* and *Lilac*. To create an inclusive query use square brackets:

```
title:[Iris to Lilac]
```

Term Prioritising Searches

Use term prioritising searches in case the fragment of text that you are searching for contains certain words that are more important to your search than the rest of them. For example, if you are searching for *Autumn Flowers*, a good idea is to prioritise the word *Autumn* since the word *Flowers* occurs more often. To prioritise a word use the ^ symbol:

```
Autumn^6 Flowers
```

Searches Using Boolean Operators

You are able to use the **AND**, +, **OR**, **-**, and **NOT** operators.

To search for documents that contain both the words *Garden* and *Preparation*, use:

```
Garden AND Preparation
```

To search for documents that must contain the word *Garden* and may contain the word *Preparation*, use:

```
+Garden Preparation
```

To search for documents that contain either the word *Garden* or the word *Preparation*, use:

```
Garden OR Preparation
```

To search for documents that contain *Garden Preparation* but not *Preparation of the Flowers*, use:

```
"Garden Preparation" - "Preparation of the Flowers"
```

Searches Using Grouping

To search either for the word *Garden* or *Preparation*, and the word *Flowers*, use:

```
(Garden OR Preparation) AND Flowers
```

Searches Using Element Grouping

To search for a title that contains both the word *Flowers* and the phrase *Garden Preparation*, use:

```
title:(+Flowers +"Garden Preparation")
```

Searching in File Paths

To perform a search in the file paths of your resources, open the **Open/Find Resources** dialog or the **Open/Find Resource** view, enable the **In file paths** option, and in the search field enter the terms that you want to search for.

The **Open/Find Resource** view and dialog allows you to search for a resource either by its name or by its path (or by a fragment of its path).

You can use wildcards when you perform such searches:

- Use "*" to match any sequence of characters;
- Use "?" to match any single character.

For example, if you search for **-preferences-page* you will find all the resources that contain the *-preferences-page* fragment in their name. If you search for **/samples/*.gif*, you will find all the `.gif` images from the `samples` directory.

Searching in Reviews

To perform a search in the edits of your resources, open the **Open/Find Resource** dialog or the **Open/Find Resource** view, enable the **In reviews** option, and in the search field enter the terms that you want to search for.

The following options are available:

- **Type** - specifies whether you want to search for content in comments, insertions, deletions, or highlighted content;
- **Author** - displays all the authors of the edits in your resources. The authors are collected when indexing. You can set a specific author for your search or all of them;
- **Time**- specifies the time when the edits that you are searching through were created.

Both the view and the dialog display the edits that contain the search results and their parent topics along with a short description. To hide this description, go to **Settings** and disable the **Show Description** option.

Technical Aspects

When Oxygen XML Author performs the indexing of your resources, the refereed content from your documents is not taken into account. For example, when DITA documents are indexed, the content from the `conref` elements is not

parsed. The files that make up the index are stored on the disk in the
`C:\Users\USER_NAME\AppData\Roaming\com.oxygenxml.author\lucene` folder.

Opening Local Files at Start-up from Command Line

To open a local file at start-up when you open Oxygen XML Author from the command line, add the paths for one or
more local files as parameters in the command line:

*   scriptName [pathToXMLFile1] [pathToXMLFile2] ... where scriptName is the name of the startup script for your
    platform (`oxygenAuthor.bat` on Windows, `oxygenAuthor.sh` on Unix/Linux, `oxygenAuthorMac.sh`
    on Mac OS) and *pathToXMLFileN* is the name of a local XML file.

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

Opening and Saving Remote Documents via FTP/SFTP/WebDAV/SharePoint

Oxygen XML Author supports editing remote files, using the FTP, SFTP ,WebDAV, SharePoint, and SharePoint Online
for Office 365 protocols. You can edit remote files in the same way you edit local files.For example, you are able to add
remote files a project, or make them subject of XSL and FO transformations.

You can open one or more remote files in *the Open URL dialog*.

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

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

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

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

The Open URL Dialog

To open this dialog, go to **File** > **Open URL ...** (or click the **Open URL ...** toolbar button), then choose the
 Browse for remote file option from the drop down action list.
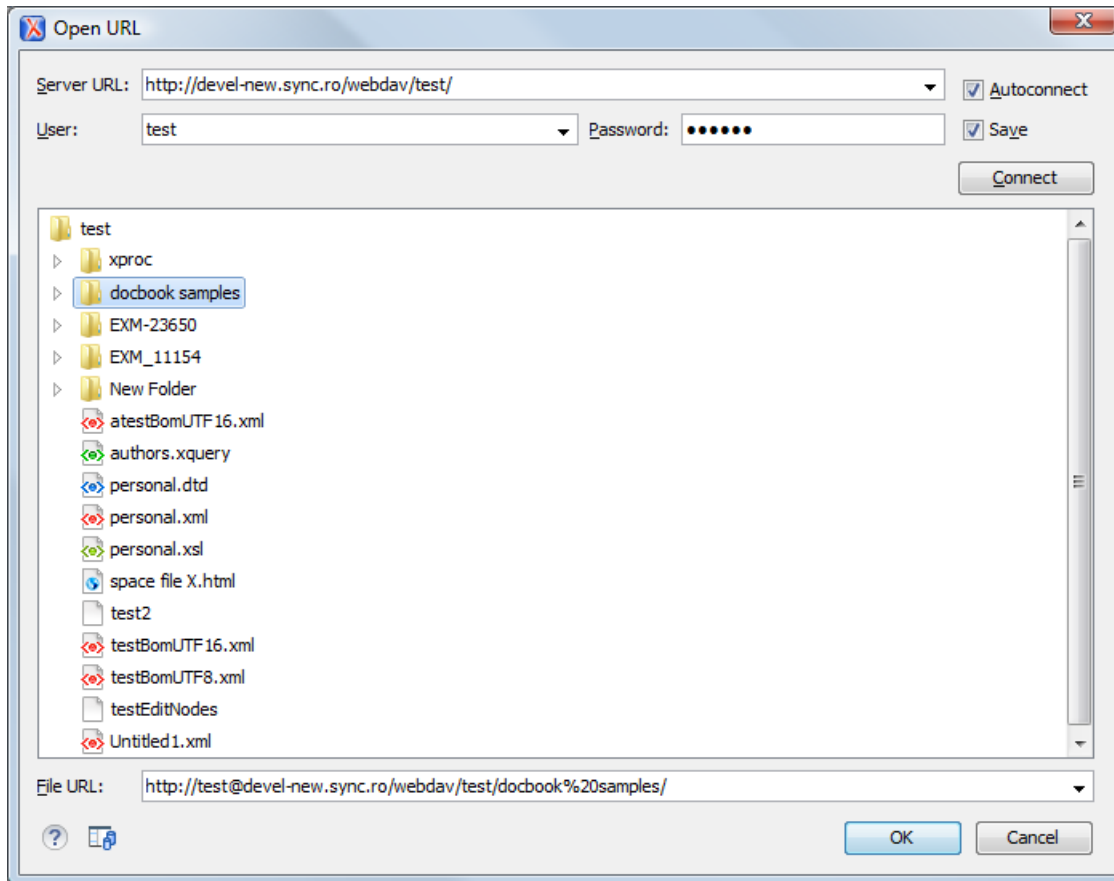
Figure 63: Open URL Dialog

The displayed dialog is composed of several parts:

- The *Identification* section contains the access credentials. 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 in 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.

- In the server combo you can specify the protocol (HTTP, HTTPS or FTP) and the host name or IP of the server.

  Tip: When specifying a URL, follow these rules:

  - to access an FTP server, write the protocol, host, and port (if using a non-standard one), like `ftp://server.com` or `ftp://server.com:7800/`
  - to access a WebDAV server, write the path to the directory of the WebDAV repository along with the protocol and the host name, like `https://www.some-webdav-server.com:443/webdav-repository/`

    Important:

    Make sure that the repository directory ends in a slash "/", like `https://www.some-webdav-server.com:443/webdav-repository/`

By pressing the **Connect** button the directory listing will be shown in the component below. If the input URL points to a SharePoint server, a dedicated SharePoint browsing component is presented. When **Autoconnect** is selected, the browse action is performed every time when you open the dialog.

- The browser view:

  - In case you are browsing a WebDAV or FTP repository, the items are presented in a tree-like fashion. You can browse the directories, and make multiple selections. Additionally, you may use the **Rename**, **Delete**, and **New Folder** actions to manage the file repository.

    📝 **Note:** The file names are sorted in a case-insensitive way.

  - When you browse a SharePoint repository, a specialized component renders the SharePoint site content.
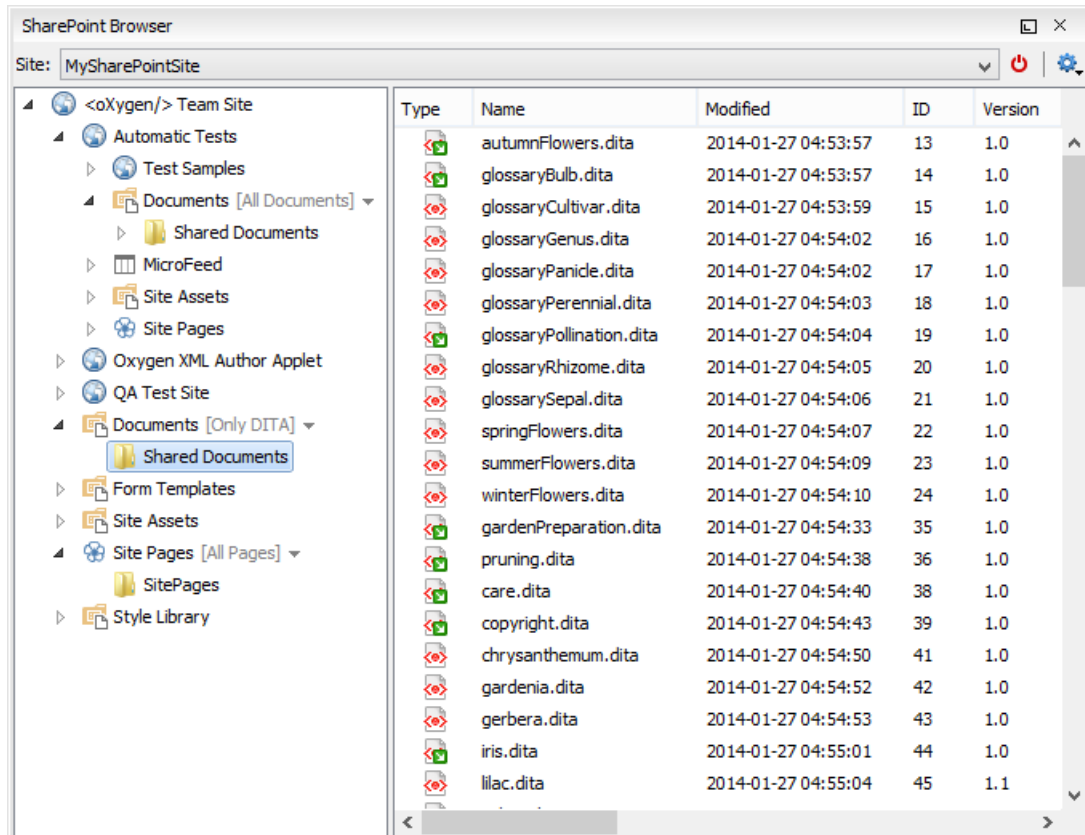
Figure 64: Browsing a SharePoint Repository

The left side navigation area presents the SharePoint site structure in a tree-like fashion displaying the following node types: *sites* (⬜), *libraries*, and *folders*.

Depending on a node's type, a contextual menu offers customized actions that can be performed on that node.

📝 **Note:** The contextual menu of a folder allows you to create new folders, new documents, and to rename and delete the folder.

📝 **Note:** The rename and delete actions are not available for library root folders (folders located at first level in a SharePoint library).

Each library node display next to its name a drop down box where you can select the current library *view*. This functionality is also available on the node's contextual menu, under the **Current View** submenu.

The content of a folder is displayed in a tabular form, where each row represents the properties of a folder or document. The list of columns and the way the documents and folders are organized depends on the currently selected view of the parent library.

You can filter and sort the displayed items. To display the available filters of a column, click the filter widget located on the column's header. You can apply multiple filters at the same time.

> **Note:** A column can be filtered or sorted only if it was configured this way on the server side.
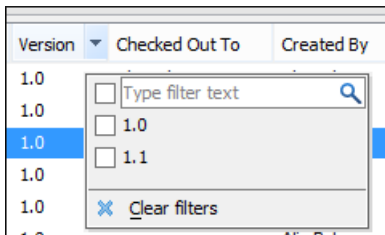
• The editable combo box, in which it can be specified directly the URL to be opened or saved.

> **Tip:**
>
> You can type in here an URL like `http://some.site/test.xml`, in case the file is accessible through normal HTTP protocol, or `ftp://anonymous@some.site/home/test.xml` if the file is accessible through anonymous FTP.

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

Changing File Permissions on a Remote FTP Server

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

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

WebDAV over HTTPS

If you want to access a WebDAV repository across an insecure network, Oxygen XML Author 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 Author 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 Author. This means that Oxygen XML Author can find the certificate in the key store of the Java Runtime Environment in which it runs. You know the server certificate is not in the JRE key store if you get the error *No trusted certificate found* when trying to access the WebDAV repository.

Troubleshooting HTTPS

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

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

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

      If this is your first visit to the repository it will be displayed a security alert stating that the security certificate presented by the server is not trusted.
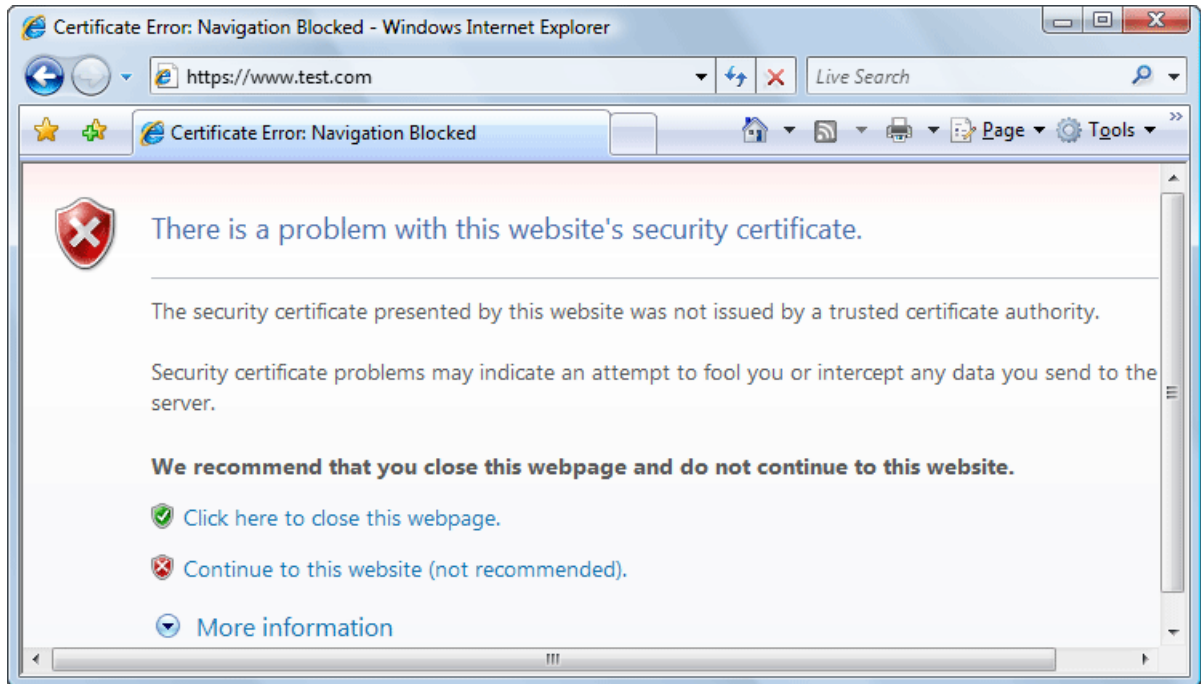
Figure 65: Security alert - untrusted certificate

   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 Author.

a) Open a text-mode console with administrative rights.

b) Go to the `lib/security` directory of the JRE running Oxygen XML Author. You find the home directory of the JRE in the *java.home* property that is displayed in the **About** dialog, **Installation Details**, **System properties** tab. On Mac OS X systems, the `lib/security` directory is usually located in `/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home` directory.

c) Run the following command:

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

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

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

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

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

3. Restart Oxygen XML Author.

Single Sign-on

Oxygen XML Author implements the *Single sign-on* property, meaning that you can log in once and gain access to multiple services without being prompted to log in for each of them. The implementation is based on the *Kerberos* protocol and relies on a *ticket-granting ticket (TGT)* that Oxygen XML Author obtains from the operating system.

To turn on the *Kerberos*-based authentication, you need to add the following system property in the `.vmoptions` configuration file:

```
-Djavax.security.auth.useSubjectCredsOnly=false
```

Opening the Current Document in System Application

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

Switching Between Opened Tabs

There are two actions for cycling through the opened file tabs:

Ctrl+Tab (Command+Tab on OS X)

Switches between the tabs with opened files in the order most recent ones first.

Ctrl +Shift+Tab (Command+Shift+Tab on OS X)

Switches between the tabs with opened files in the reverse order.

Closing Documents

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

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

The Contextual Menu of the Editor Tab

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

Close

Closes the current editor.

Close Other Files

Closes all opened editor but the one you are currently viewing.

Close All

Closes all opened editors.

Reopen last closed editor

Reopens the last closed editor.

Maximize/Restore Editor Area

Collapses all the side views and spans the editing are to cover the entire width of the main window.

Add to project

Adds the file you are editing to the current project.

Add all to project

Adds all the opened files to the current project.

Copy Location

Copies the disk location of the file.

Show in Explorer (Show in Finder on OS X)

Opens the Explorer to the file path of the file.

Viewing File Properties

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

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

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

Using the Project View

The Project view is designed to assist the user in organizing and managing related files grouped in the same XML project. The actions available on the context menu and toolbar associated to this panel, enable the creation of XML projects and shortcuts to various operations on the project documents.
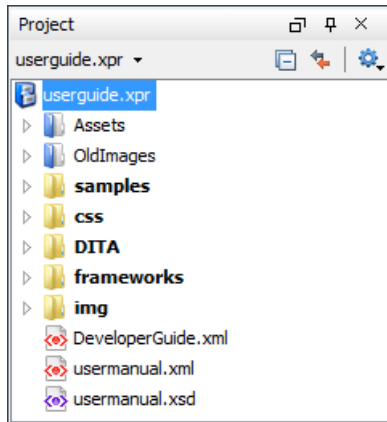
Figure 66: The Project View

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

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

Open Project ... (Ctrl (Meta on Mac OS)+F2)

Opens an existing project. An alternate way to open a project is to drop an Oxygen XML Author XPR project file from the file explorer in the **Project panel**.

New Project

Creates a new, empty project.

The following actions are grouped in the upper right corner:

Collapse All

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

Link with Editor

When selected, the project tree highlights the currently edited file, if that is found in the project files.

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

Settings

A sub-menu containing the following actions:

Filters

Allows you to filter the information displayed in the **Project View**. Click the toolbar button to set filter patterns for the files you want to show or hide. Also, you can set filter patterns for the linked directories that are hidden.

Show Full Path

When selected, linked files and folders are presented with a full file path.

Enable Master Files Support

Select this option to enable the *Master Files support*.

Change Search and Refactor operations scope...

Allows you to change the collection of documents that define the context of the *search and refactor operations*.

- **Use only Master Files, if enabled** - Restricts Oxygen XML Author to perform the search and refactor operations starting from the master files defined for the current resource. This option is available when you select **Project** in the **Select the scope for Search and Refactor operations** dialog and the **Master Files** support is enabled.
- **Working sets** - Allow you to specify exactly the set of files on which the search and refactor operations will act on.

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

- *Logical folders* - marked with a blue icon on Windows and Unix/Linux (📁) and a magenta icon on Mac OS X (📁). This folder type has no correspondent on the physical disk, being used as containers for related items. Creating and deleting them does not affect the file system on disk.
- *Linked folders* - marked with the operating system specific icon for folders (usually a yellow icon on Windows and a blue icon on Mac OS X). They content mirror a real folder existing in the file system on disk.

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.

Add Content to a Logical Folder

The project itself is considered a logical folder. You can add content to a logical folder using one of the actions available in the contextual menu:

📁 Add Folder

Adds a link to a physical folder, whose name and content mirror a real folder existing in the file system on disk.

The icon of this action is different on Mac OS X (📁) as the standard folder icon on Mac OS X is not the usual one from Windows and Unix/Linux systems.

📄 Add Files

Adds links to files on disk.

📄 Add Edited File

Adds a link to the current edited file to the project.

Managing Project Content

You can create linked folders by dragging and dropping a folder from the Windows Explorer / Mac OS X Finder over the project tree or by selecting in the contextual menu **Add Folder**.

📄 **Note:** The liked files presented in the **Project** view are marked with a special decorator.

Also, you can *move the resources of the project* with drag and drop operations on the files and folders of the tree.

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

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

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

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

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

• invoke the **Rename** action from the contextual menu
• press **F2**
• click the selected item

To finish editing the item name, press **Enter** on your keyboard.

📄 **Note:** The **Rename** action is available on logic files also.

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

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

Validate Files

The currently selected files in the **Project** view can be checked to be XML well-formed or validated against a schema of type DTD, XML Schema, Relax NG, Schematron or NVDL with one of the following contextual menu actions found in the **Validate** entry:

✓ **Check Well-Formedness**

Checks if the selected file or files are well-formed.

✓ **Validate**

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

Validate with Schema...

Validates the selected file of files against a specified schema.

⚒✓ **Configure Validation Scenario**

> Allows you to configure and run a *validation scenario*.

Applying Transformation Scenarios

The currently selected files in the **Project** view can be transformed in one step with one of the actions **Transform** >

▶ **Apply Transformation Scenario(s)**, **Transform** > ⚒▶ **Configure Transformation Scenario(s)...** and *Transform >*

⚒▶ *Transform with...* available as contextual actions inside the **Transform** sub menu. This, together with the logical folder support of the project allows you to group your files and transform them very easily.

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

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

Name	Value
Name	personal.xml
Line separator	Unix-like [\n]
Path of current file	file:/D:/Projects/samples/personal.xml
Content-type	text/xml
Encoding	UTF8
Number of characters	1528
BIDI	false
Content Completion	file:/D:/Projects/samples/personal.dtd
Indent size	2
Indent with tabs	false
Line width - Format and Ind...	100
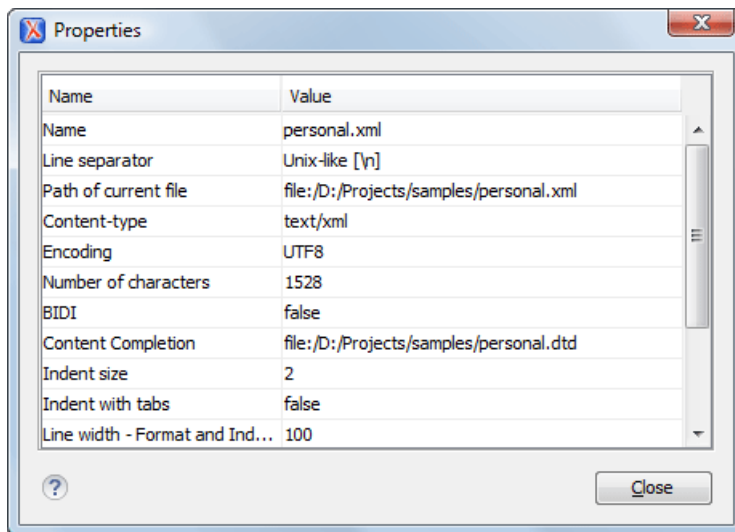
Figure 67: The Properties Dialog

You can also use drag and drop to arrange the files in folders, (but not in linked folders). Also, dragging and dropping files from the project tree to the editor area results in the files being opened.

Other Context-Dependent Actions

The contextual menu on the **Project View** contains also other actions:

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

> On Windows, the content of the selected folder, or the selected file is presented in a specific explorer window. On MAC OS X, the parent folder is opened and the selected folder is highlighted in a specific finder window. This window also presents selected files.

Open with

> Open selected file with one of internal tools: *SVG Viewer*, *Hex Viewer*, *Large File Viewer*, *MathML Editor*, WSDL/SOAP Analyzer, *DITA Maps Manager*, *Archive Browser*.

Open All Files

> Opens all the selected files with the corresponding editors.

♻ **Refresh**

> Refreshes the content of the **Project** view;

Note: When you refresh the content of the **Project** view, Oxygen XML Author also refreshes the dependencies between the resources in *the Master Files directory*.

Find/Replace in Files

Allows you to *find and replace text in multiple files.*

Check Spelling in Files

Allows you to *check the spelling of multiple files.*

Open in SVN Client

Syncro SVN Client tool is opened and it highlights the selected resource in its corresponding working copy.

Menu Level Actions

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

New Project

Creates a new, empty project.

Open Project ... (Ctrl (Meta on Mac OS)+F2)

Opens an existing project. An alternate way to open a project is to drop an Oxygen XML Author XPR project file from the file explorer in the **Project panel**.

Save Project As...

Allows you to save the current project under a different name.

Validate all project files

Checks if the project files are well-formed and their mark-up conforms with the specified DTD, XML Schema, or Relax NG schema rules. It returns an error list in the message panel.

Show Project View

Displays the project view.

Reopen Project

Contains a list of links of previously used projects. This list can be emptied by invoking the **Clear history** action.

Team Collaboration - Apache Subversion™

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

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

Project Level Settings

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

Moving/Renaming Resources in the Project View

The **Project** view allows you to move or rename a file from the current project.

To move a file or a directory, drag and drop it to the new location in the tree structure from the **Project** view. You can also right click the file or directory and select the **Refactoring** > **Move resource** action from its contextual menu, or use the usual ✂ **Cut** and 📋 **Paste** actions. Oxygen XML Author presents the **Move** dialog if you used the drag and drop or the **Cut/Paste** actions. The **Move resource** dialog is presented in case you used the refactoring actions. The following fields are available:

- **Destination** - available in the **Move resource** dialog only. Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location;

- **New name** - presents the current name of the moved resource and gives you the option to change it;
- **Update references of the moved resource(s)** - enable this option to update the references to the resource you are moving, in accordance with the new location and name.

To quickly rename a file or a directory, use the in-place editing either by pressing **F2** or by selecting **Rename** from the contextual menu of the resource. You can also right click a file or a directory and select the **Refactoring** > **Rename resource** action from its contextual menu. Oxygen XML Author presents the **Rename** dialog in case you used in-place editing. The **Rename resource** dialog is presented if you used the refactoring actions. The following fields are available:

- **New name** - presents the current name of the edited resource and allows you to modify it;
- **Update references of the renamed resource** - enable this option to update the references to the resource you are renaming;
- **Scope** - specifies the *scope of the rename operation*.

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

> **Note:** The support to update references is available for XML documents.

Problems with Updating References of Moved/Renamed Resources

In some case the references of a moved or a renamed resource can not be update. One case is when a resource is resolved through an XML catalog. Another problem can appear when the path to the moved/renamed resource contains entities. For these cases, Oxygen XML Author displays a warning dialog.
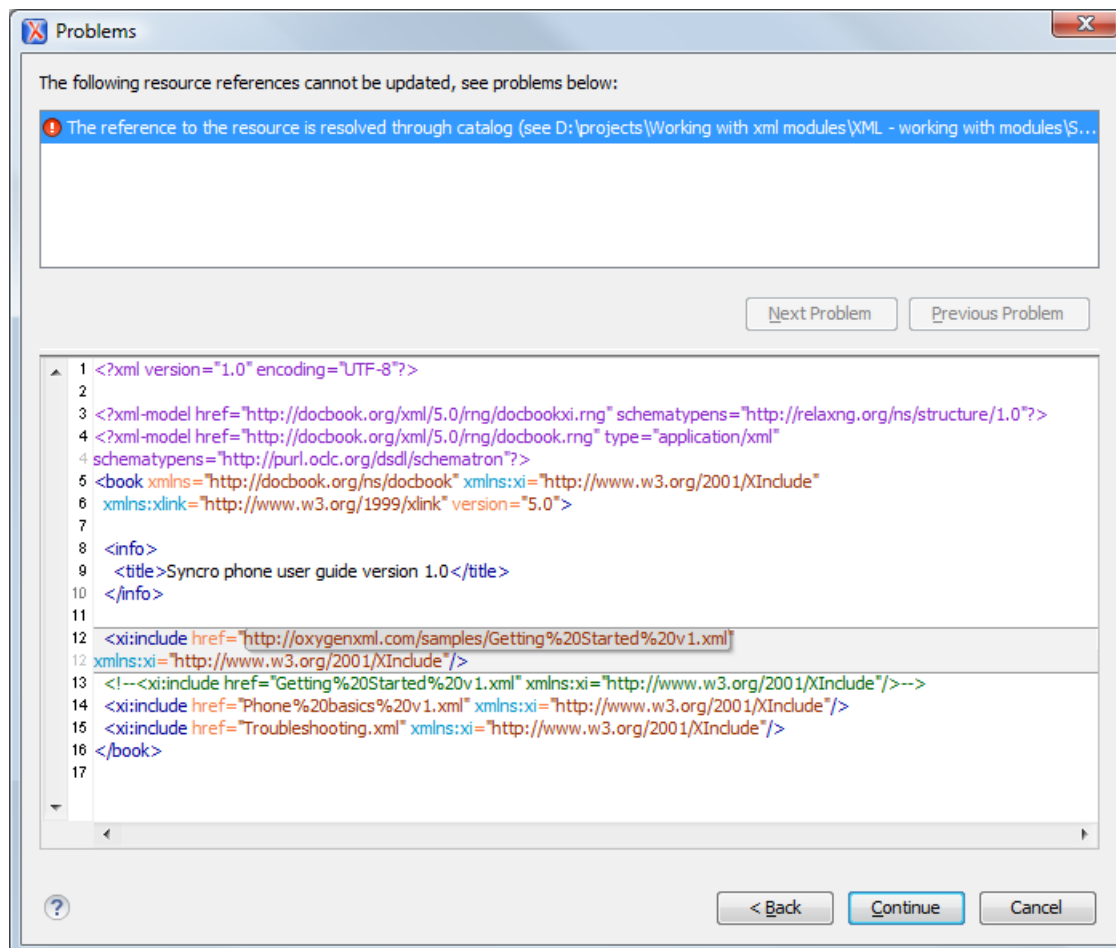
Figure 68: Problems Dialog

Defining Master Files at Project Level

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

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

Introduction

Oxygen XML Author allows you to define master files at project level. These master files are automatically used by Oxygen XML Author to determine the context for operations like validation, content-completion, refactoring or search for XML. Oxygen XML Author maintains the hierarchy of the master files, helping you to determine the editing context.

To watch our video demonstration about the **Master Files** support for XML documents, go to *Working with Modular XML Files*.

Master Files Benefits

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

- when the module is validated, Oxygen XML Author automatically identifies the master files which include that module and validates all of them;
- the *Content Completion Assistant* presents all the components collected starting from the master files towards the modules they include;
- the master files defined for the current module determines the *scope of the search and refactoring actions*. Oxygen XML Author performs the search and refactoring actions in the context that the master files determine, improving the speed of execution.

Enabling the Master Files Support

Oxygen XML Author stores the master files in a folder located in the **Project** view, as the first child of the project root. The **Master Files** support is disabled by default. To enable the **Master Files** support, go to the **Settings** menu of the **Project** view and select **Enable Master Files Support**. You can also select **Enable Master Files Support** from the contextual menu of the **Master Files** directory, or from the contextual menu of the project itself. Oxygen XML Author allows you to enable/disable the **Master Files** support for each project you are working on.

When the master files support is disabled Oxygen XML Author displays a window tip located at the bottom of the project view. This window contains an **Enable** button and a **read more...** option. Clicking the **read more** option takes you to the user guide. Clicking the **Enable** button opens the **Enable Master Files** dialog. This dialog contains general information about the **Master Files** support and allows you to enable the support. You can also use the **Detect and Enable** option to detect the master files from the current project.

> 📄 **Note:** Once you close this window tip, Oxygen XML Author hides it for all projects. To make the window tip reappear, restore Oxygen XML Author to its default settings.

> 📄 **Note:** Restoring Oxygen XML Author to its default settings causes loss of your customized options.

Detecting Master Files

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

- right click the root of the project and select 🔵 **Detect Master Files...**;
- use the 🔵 **Detect Master Files from Project...** option available in the contextual menu of the **Master Files** folder.

Both these options display the **Detect Master Files** wizard dialog. The detected master files are presented in a tree like fashion. The resources are grouped in three categories:

- Possible master files - the files presented on the first level in this category are not imported/included from other files. These files are most likely to be set as master files.

- Cycles - the files that are presented on the first level have circular dependencies between them. Any of the files presented on the first level of a cycle is a possible master file.
- Standalone - files that do not include/import other files and are also not included/imported themselves. No need to be set as master files.

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

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

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

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

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

> **Attention:** In case the **Master Files** support is disabled, the **Master Files** directory is rendered only if it contains master files.

Adding/Removing a Master File

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

- right click a file from your project and select **Add to Master Files** from the contextual menu;
- select **Add Files** or **Add Edited File** from the contextual menu of the **Master Files** directory;
- drag and drop files in the **Master Files** directory;
- from the contextual menu of the **Resource Hierarchy Dependencies** view, using the **Add to Master Files** action.

You can view the master files for the current resource in the **Properties** dialog of the **Project** view and the master files for the current editor in the **Properties** and **Information** views.

Project Validation and Transformation

The **Master Files** support is also useful for project level validation and transformation. When you hover the mouse cursor over a file in the **Master Files** directory, Oxygen XML Author displays the **Validate** and **Transform** buttons at the right of the file. Select one of these buttons to run a transformation, or validation scenario. In case the current node is selected, Oxygen XML Author executes a batch transformation and validation. In case the current node is not selected, Oxygen XML Author executes the validation and transformation for the current node only. The behavior of these actions is the same as the behavior of the transformation actions available in the contextual menu.

> **Note:** The tooltip of the **Validate** and **Transform** buttons displays the associated scenarios that you can execute.

When you hover the mouse cursor over the **Master Files** directory itself, apart from the **Validate** and **Transform** buttons, Oxygen XML Author displays a **Help** button. Either click this button, or press F1 on your keyboard to open the Help section regrading the **Master Files** support.

The Contextual Menu of the Master Files

The contextual menu of the **Master Files** directory contains the following actions:
New
 Allows you to create a file, logical folder, or project.

Add Files

Allows you to add master files to the **Master Files** directory.

Add Edited File

Use this option to add the file you are editing to the **Master Files** directory.

Open All Files

Opens all the files of the **Master Files** directory.

Paste

Pastes the files you copy in the **Master Files** directory.

Rename

Allows you to rename a file of the **Master Files** directory.

Refresh

Refreshes the content of the **Master Files** directory.

Find/Replace in Files...

Opens the *Find/Replace dialog*.

Open/Find Resource

Opens the *Open/Find Resource dialog*.

Check Spelling in Files

Opens the *Check Spelling in files dialog*.

Transform

Provides access to one of the following actions:

> **Apply Transformations Scenario(s)**
>
> Applies the transformation scenarios associated with the **Master Files** directory.
>
> **Configure Transformation Scenario(s)...**
>
> Opens the *Configure Transformation Scenario dialog*.
>
> **Transform with...**
>
> Opens the **Transform with** dialog which allows you to select the transformation scenario you want to execute.

Validate

Provides access to one of the following actions:

Check Well-Formedness

> Allows you to check if a document is *Namespace Well-Formed XML*.

Validate

> Oxygen XML Author performs the validation of the master files.

Validate with Schema

> Opens the **Validate with** dialog. Oxygen XML Author performs the validation of the master files using a schema.

Configure Validation Scenario

> Opens the *Configure validation scenario dialog*.

Detect Master Files from Project...

Enables automatic detection of master files.

Enable Master Files Support

Select this option to enable the **Master Files** support.

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 Author.

Supported Schema Types for XML Documents

The supported schema types are:

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

Setting a Default Schema

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

- the *validation scenario* associated with the document;
- the validation scenario associated with the document type (if defined);
- the document schema declaration;

  > **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);

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

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

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

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

> ⚠ **Important:**
>
> The editor is creating the content completion lists by analysing the specified schema and the current context (the position in the editor). If you change the schema, then the list of tags to be inserted is updated.
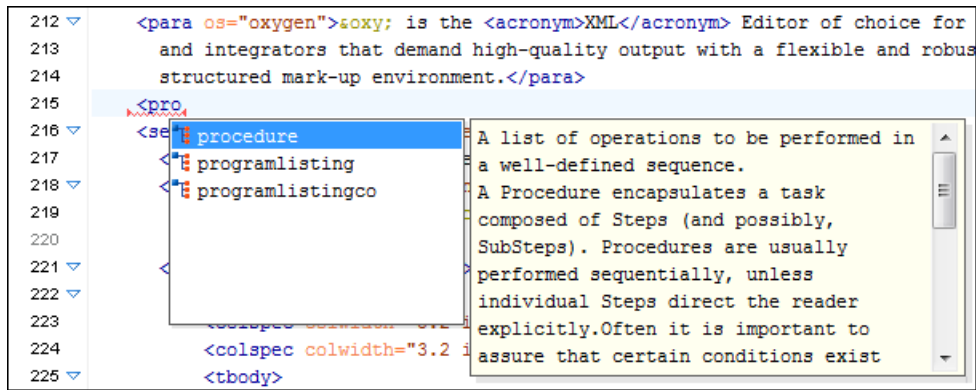
Figure 69: Content Completion Driven by DocBook DTD

Making the Schema Association Explicit in the XML Instance Document

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

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

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

Go to menu **Document** > **Schema** > **Associate schema...** or click the 🖈 **Associate schema** toolbar button to select the schema that will be associated with the XML document. The following dialog is displayed:
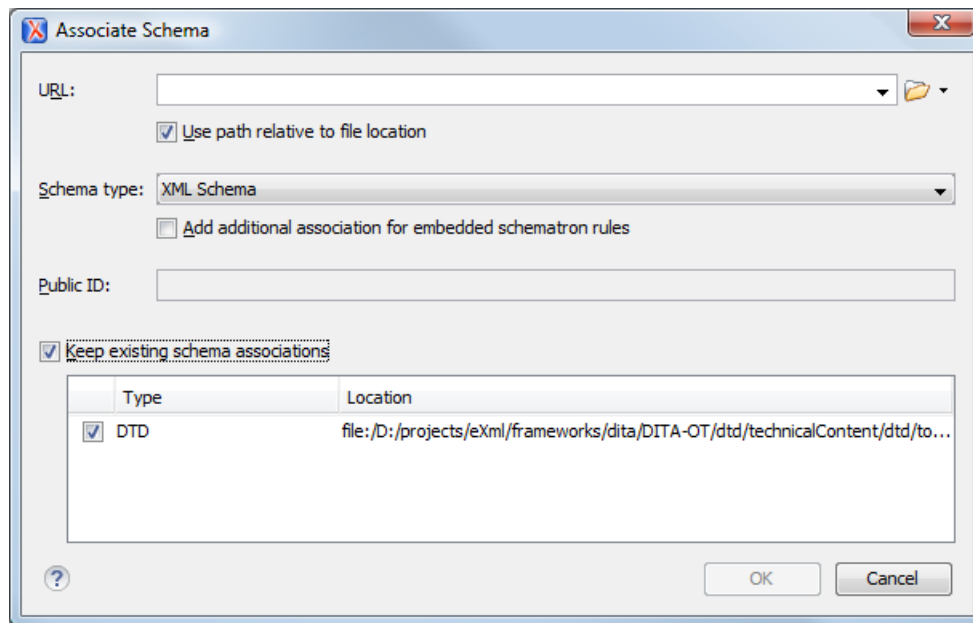
Figure 70: The Associate Schema Dialog

The available options are:

- **URL** - Contains a predefined set of schemas that are used more often and it also keeps a history of the last used schemas. The URL must point to the schema file which can be loaded from the local disk or from a remote server through HTTP(S), FTP(S);

- **Schema type** - Selected automatically from the list of possible types in the **Schema type** combo box (XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, NVDL) based on the extension of the schema file that was entered in the **URL** field;
- **Public ID** - Specify a public ID if you have selected a DTD;
- **Add additional association for embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules, enable this option;
- **Use path relative to file location** - Enable this option if the XML instance document and the associated schema contain relative paths. The location of the schema file is inserted in the XML instance document as a relative file path. This practice allows you, for example, to share these documents with other users, without running into problems caused by different project locations on physical disk;
- **Keep existing schema associations** - Enable this option to keep the associations of the currently edited document with a Schema when you associate a new one.

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

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

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

Associating a Schema With the Namespace of the Root Element

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

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

The `xml-model` Processing Instruction

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

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

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

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

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

Streamline with Content Completion

The intelligent **Content Completion Assistant** available in Oxygen XML Author enables rapid, in-line identification and insertion of structured language elements, attributes and, in some cases, their parameter options.
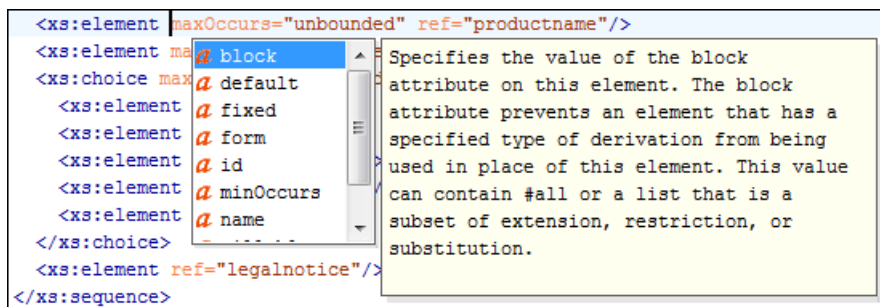
Figure 71: Content Completion Assistant

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

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

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

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

> **Note:** You can also start the **Content Completion Assistant** from **Document** > **Content Completion** > **Start Content Completion**.

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

To insert the selected content:

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

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

After inserting an element, the cursor is positioned:

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

The **Content Completion Assistant** is displayed:

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

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

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

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

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

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

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

Set Schema for Content Completion

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

- The schema specified explicitly in the document. In this case Oxygen XML Author 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.

Content Completion in Documents with Relax NG Schemas

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

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

In documents based on this schema, the **Content Completion Assistant** offers the following list of values:
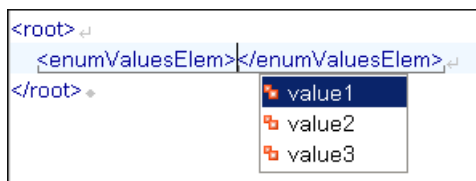
Figure 72: Content Completion assistant - element values in Relax NG documents

Schema Annotations

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

- the Content Completion Assistant.
- a small tooltip window shown when the mouse hovers over an element or attribute. The tooltip window can be invoked at any time using the F2 shortcut.

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

Styling Annotations with HTML

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

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

Collecting Annotations from XML Schemas

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

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

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

Collecting Annotations from Relax NG Schemas

For Relax NG schema element / attribute annotation are made using the `<documentation>` element from the `http://relaxng.org/ns/compatibility/annotations/1.0` namespace. However, 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. To activate this behaviour, enable the *Use all Relax NG annotations as documentation* option.

Collecting Annotation from DTDs

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

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

Content Completion Helper Views

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

The Model View

The **Model** view presents the structure of the currently edited tag and tag documentation defined as annotation in the schema of the current document.
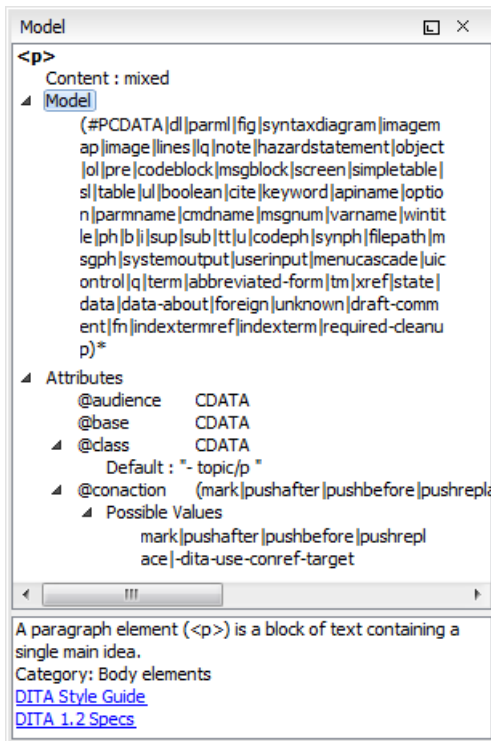
Figure 73: The Model View

The **Model** view is comprised of:

- *An element structure panel.*

- *An annotation panel.*

The Element Structure Panel

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

The information includes the name, model and attributes the currently edited tag may have. The allowed attributes are shown along with imposed restrictions, if any.
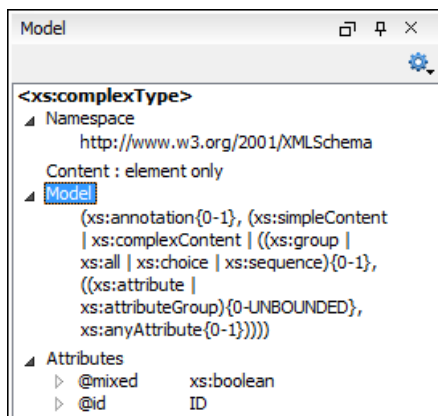
Figure 74: The Element Structure Panel

The Annotation Panel

The **Annotation** panel displays the annotation information for the currently selected element. This information is collected from the XML schema.
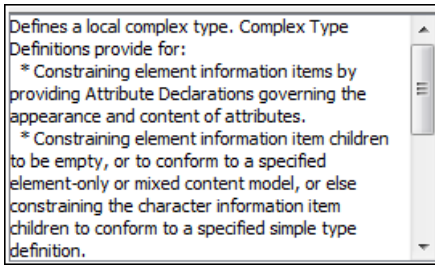
Figure 75: The Annotation panel

The Attributes View

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

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

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

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

The **Attributes** table is sortable, three sorting modes being available by clicking the **Attribute** column name: alphabetically ascending, alphabetically descending, or custom order. The custom order places the already used attributes at the beginning of the table, as they appear in the element, followed by the rest of the allowed elements, as they are declared in the associated schema.
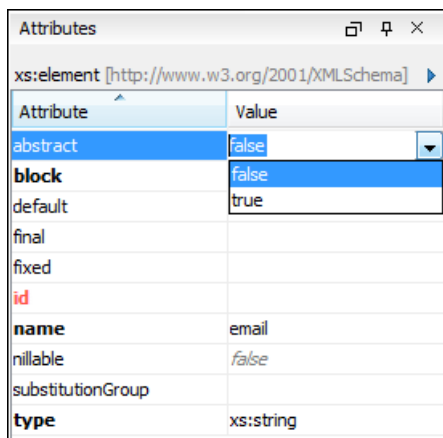
Figure 76: The Attributes View

The Elements View

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

Figure 77: The Elements View

The Entities View

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

Figure 78: The Entities View

Code Templates

Code templates are code fragments that can be inserted quickly at the editing position. Oxygen XML Author comes with a set of ready-to use templates for XSL, XQuery, and XML Schema. You can also *define you own code templates and share them with others*.

To get a complete list of available code templates, press **Ctrl+Shift+Space (Command+Shift+Space on OS X)**. To enter the code template, select it from the list or type its shortcut code and press **Enter**.

If you know the template shortcut, type it and press **Ctrl+Shift+Space (Command+Shift+Space on OS X)**.

Code templates are also displayed when you press the content completion keys,**Ctrl+Space (Command+Space on OS X)** or **Enter** in *Author mode*.

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

Validating XML Documents

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

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

Checking XML Well-formedness

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

The XML Syntax rules for Well-Formed XML are:

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

The namespace-wellformed rules are:

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

The namespace-valid rules are:

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

To check if a document is *Namespace Well-Formed XML*, go to **Document** > **Validate** > **Check Well-Formedness** (**Ctrl+Shift+W (Command+Shift+W on OS X)**). You can also open the drop-down menu of the ✅ validate button on the toolbar and select ✅ **Check Well-Formedness**. If any error is found the result is returned to the message panel. Each error is one record in the result list and is accompanied by an error message. Clicking the record will open the document containing the error and highlight its approximate location.

A not Well-Formed XML Document

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

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

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

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

A not namespace-wellformed document

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

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

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

A not namespace-valid document

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

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

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

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

Validating XML Documents Against a Schema

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

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

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

Marking Validation Errors and Warnings

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

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

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

  A more detailed report of the errors is displayed in the tooltip of the validation indicator. In case there are errors, only the first three of them will be presented in the tooltip.

- Middle area where the error markers are depicted in red (with a darker color tone for the current selected one). To limit the number of markers shown *open the **Preferences** dialog* and go to **Editor** > **Document checking** > **Maximum number of problems reported per document**.

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

  The *Document checking user preferences* are easily accessible from the button displayed at the beginning of the error message on the bottom of the editor panel.

- Bottom area containing two navigation arrows that will go to the next or to the previous error and a button for clearing all the error markers from the ruler. The same actions can be triggered from menu **Document** > **Automatic**

validation > **Next Error** <u>**Ctrl+. (Command+. on OS X)**</u> and **Document** > **Automatic validation** > **Previous Error** <u>**Ctrl+, (Command+, on OS X)**</u>.

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

Status messages from every validation action are logged into the *Information 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 **Validate** toolbar and on the **Document** > **Validate** menu. This action collects all error messages in the **Errors** view.

Customising Assert Error Messages

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

Validation Example - A DocBook Validation Error

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

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

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

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

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

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

Automatic Validation

Oxygen XML Author *can be configured* to mark validation errors in the document as you are editing. If you *enable the Automatic validation option* any validation errors and warnings will be *highlighted automatically in the editor panel*. The automatic validation starts parsing the document and marking the errors after a *configurable delay* from the last key typed. Errors are highlighted with underline markers in the main editor panel and small rectangles on the right side ruler of the editor panel, *in the same way as for manual validation invoked by the user.*
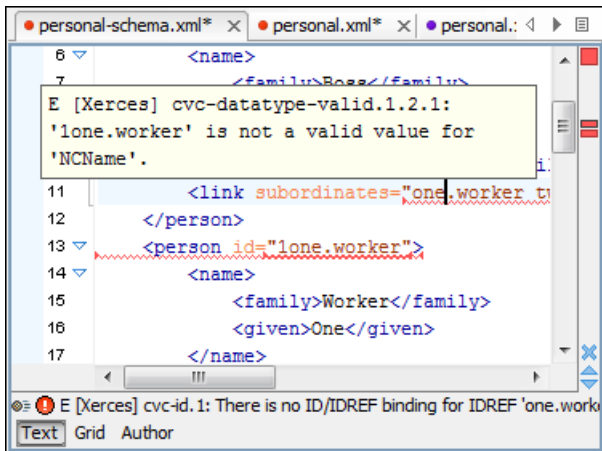
Figure 79: Automatic Validation on the Edited Document

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

Custom Validators

If you need to validate the edited document with other validation engine than the built-in one you have the possibility to configure external validators in the Oxygen XML Author 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 Author for linked messages:

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

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

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

  > ⚠ **Attention:**
  >
  > On OS X if the full path to the LIBXML executable file is not specified in the **Executable path** text field, some errors may occur during validation against a W3C XML Schema like:
  >
  > ```
  > Unimplemented block at ... xmlschema.c
  > ```
  >
  > To avoid these errors, specify the full path to the LIBXML executable file.

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

- **MSXML 4.0** - Included in Oxygen XML Author (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 Author (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 Author. 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 Author* for the [OXYGEN_DIR]/xsv installation folder. If it is installed in a different folder the predefined executable path must be *corrected in Preferences.* It is associated to XML Editor and XSD Editor. It is able to validate the edited document against XML Schema or a custom schema type.
- **SQC (Schema Quality Checker from IBM)** - Not included in Oxygen XML Author. It can be downloaded *from here* (it comes as a .zip file, at the time of this writing SQC2.2.1.zip is about 3 megabytes). The executable path and working directory are already configured for the SQC installation directory [OXYGEN_DIR]/sqc. If it is installed in a different folder the predefined executable path and working directory must be *corrected in the Preferences page.* It is associated to XSD Editor.

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

Linked Output Messages of an External Engine

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

Linked messages have the following format:

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

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

- *AdditionalInfoURL*: the URL string pointing to a remote location where additional information about the error can be found - this line is optional in a linked message.

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

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

```
Type: E
SystemID: file:///c:/path/to/validatedFile.xml
```

```
Line: 10
Column: 20
EndLine: 10
EndColumn: 35
AdditionalInfoURL: http://www.host.com/path/to/errors.html#errorID
Description: custom validator message
```

Validation Scenario

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

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

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

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

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

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

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

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

How to Create a Validation Scenario

Follow these steps for creating a validation scenario:

1. To open the **Configure Validation Scenario** dialog box, go to **Document** > **Validate** > **Configure Validation Scenario** . You can also open this dialog from the **Validate** toolbar.
   The following dialog is displayed. It contains the following types of scenarios:

   - **Predefined** scenarios are organized in categories depending on the type of file they apply to. You can identify **Predefined** scenarios by a yellow key icon that marks them as *read-only*. If the predefined scenario is the default scenario of the framework, its name is written in bold font. If you try to edit one of these scenarios, Oxygen XML Author creates a customizable duplicate;
   - **User defined** scenarios are organized under a single category, but you can use the drop-down option box to filter them by the type of file they validate;

     > **Note:** The default validation scenarios are not displayed in the scenarios list. If the current file has no associated scenarios, the preview area displays a message to let you know that you can apply the default validation.
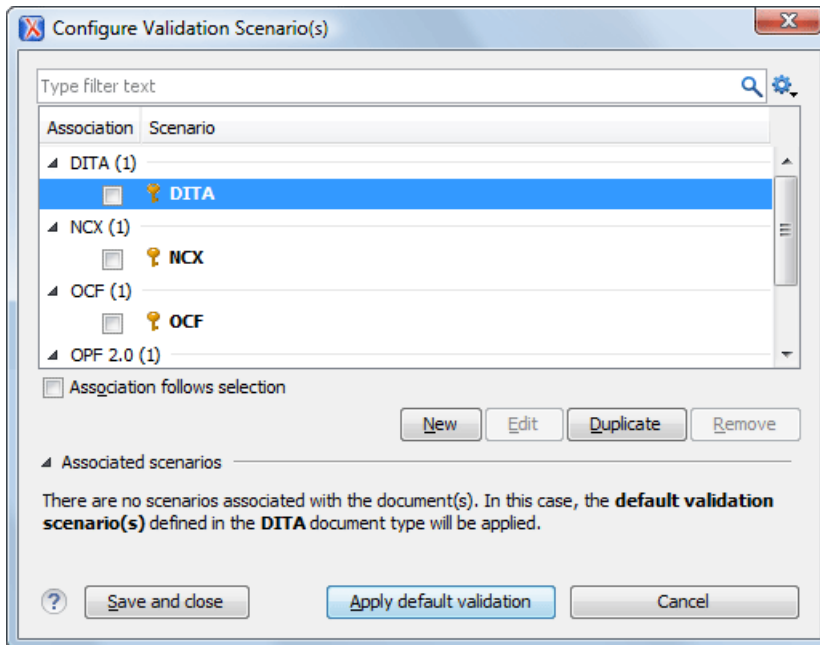
Figure 80: Configure Validation Scenario

2. Press the **New** button to add a new scenario.
3. Press the **Add** button to add a new validation unit with default settings.
   The dialog that lists all validation units of the scenario is opened.
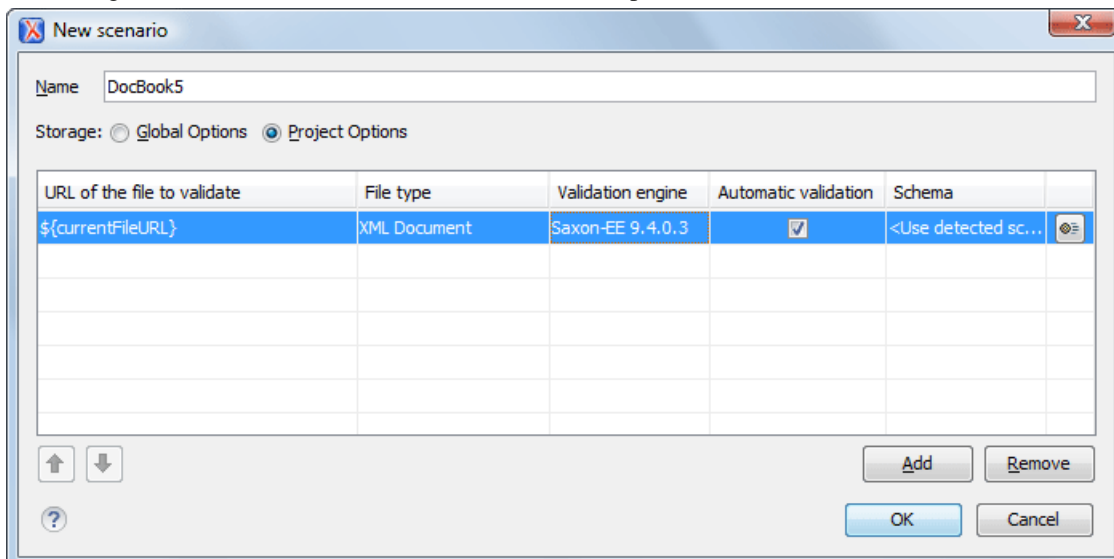
Figure 81: Add / Edit a Validation Unit

The table holds the following information:

- **Storage** - allows you to create a scenario at project level, or as global;
- **URL of the file to validate** - the URL of the main module which includes the current module. It is also the entry module of the validation process when the current one is validated;
- **File type** - the type of the document validated in the current validation unit. Oxygen XML Author 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 Author for validation of the type of document to which the current module belongs. **Default engine** is the default setting and means that the default engine

executes the validation. This engine is set in **Preferences** pages for the type of the current document (XML document, XML Schema, XSLT stylesheet, XQuery file, and others) instead of a validation scenario;

- **Automatic validation** - if this option is checked, then the validation operation defined by this row of the table is applied also by *the automatic validation feature.* If the **Automatic validation** feature is *disabled in Preferences* then this option does not take effect as the Preference setting has higher priority;
- **Schema** - the this option becomes active when you set the **File type** to **XML Document**;
- **Settings** - opens the **Specify Schema** dialog box, allowing you to set a schema for validating XML documents, or a list of extensions for validating XSL or XQuery documents. You can also set a default phase for validation with a Schematron schema.

4. Edit the URL of the main validation module.

Specify the URL of the main module:

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

```
${Desktop} - My Desktop
${start-dir} - Start directory of custom validator
${standard-params} - List of standard params for command line
${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 82: 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.

Sharing Validation Scenarios

Sometimes a group of users want to apply the same validation settings, like the main module where the validation starts, the validation engine, the schema, extensions of the engine. In order to apply the same settings consistently it is preferable to share the validation scenario with the settings by storing it at project level and sharing the project file using a source version control system (like CVS, SVN, Source Safe).
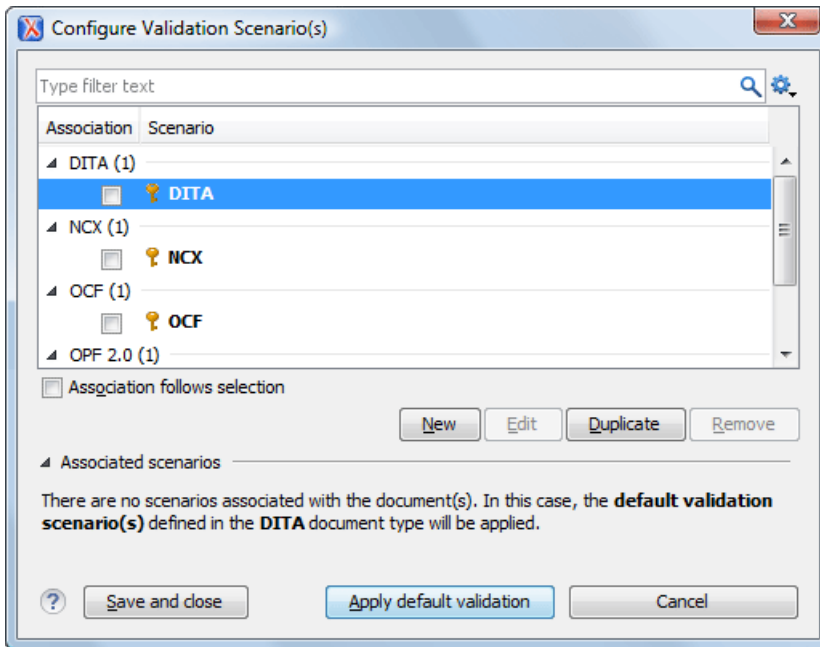
Figure 83: Configure Validation Scenario

You can specify that you want to store a scenario at project level by selecting the **Project Scenarios** option, or you can store them in the user home directory by selecting **Global Scenarios**. When you create a scenario at the project level, the URLs from the scenario become relative to the project URL.

Validation Actions in the User Interface

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

- Go to **Document** > **Validate** > **Validate Document (Ctrl+Shift+V)** or click the ✓ **Validate** button from the **Validate** toolbar. An error list is presented in the message panel. Mark-up of current document is checked to conform with the specified DTD, XML Schema or Relax NG schema rules. This action also re-parses the XML catalogs and resets the schema used for content completion.

- Go to **Document** > **Validate** > **Validate (cached)** or click the ✓ **Validate (cached)** button from the **Validate** toolbar. This action caches the schema, allowing it to be reused for the next validation. Mark-up of the current document is checked to conform with the specified DTD, XML Schema or Relax NG schema rules.

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

- Go to **Document** > **Validate** > **Validate with (Ctrl+Shift+H)** or click the ✓ **Validate with** button from the **Validate** toolbar. You can use this action to validate the current document using a schema of your choice (XML Schema, DTD, Relax NG, NVDL, Schematron schema), other than the associated one. An error list is presented in the message panel. Mark-up of current document is checked to conform with the specified schema rules. The **Validate with** action does not work for files loaded through an *Oxygen XML Author custom protocol plugin* developed independently and added to Oxygen XML Author after installation.

- Go to **Document** > **Schema** > **Open External Schema** or click the 📄 **Open External Schema** button from the **Document** toolbar to open the schema used for validating the current document in a new editor.

- Select submenu **Validate Selection** > **Validate** in the contextual menu of **Project** panel, to validate all selected files with their declared schemas.

- Select **Validate Selection with Schema ...** > **Validate With ...** from the contextual menu of the **Project** panel, to choose a schema and validate all selected files with it.

- Select the submenu **Validate** > **Configure Validation Scenario ...** of the contextual menu of **Project** panel, to configure and apply a validation scenario in one action to all the selected files in the **Project** panel, .

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

Also you can select several files in the **Project** panel and validate them with one click by selecting the action **Validate selection**, the action **Validate selection with Schema ...** or the action **Configure Validation Scenario ...** available from the contextual menu of the **Project** view.

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

References to XML Schema Specification

If validation is done against XML Schema Oxygen XML Author indicates a specification reference relevant for each validation error. The error messages contain an **Info** field that when clicked will open the browser on the *XML Schema Part 1:Structures* specification at exactly the point where the error is described. This allows you to understand the reason for that error.
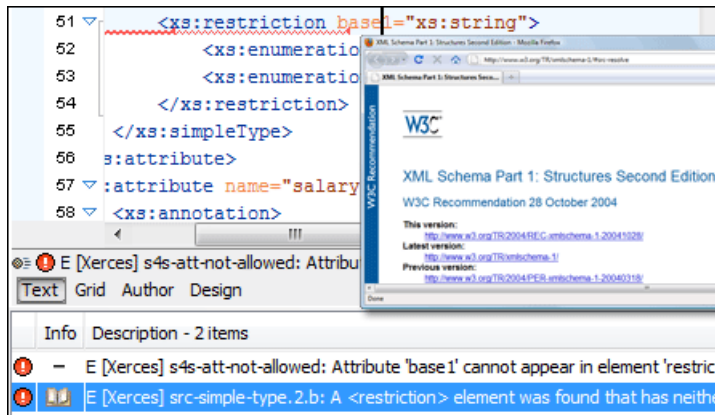
Figure 84: Link to Specification for XML Schema Errors

Resolving References to Remote Schemas with an XML Catalog

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

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

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

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

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

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

the URN can be resolved to a local schema file with a catalog entry like:
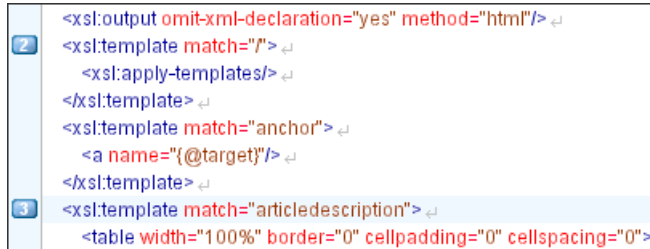
```
<uri name="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1"
    uri="topic.xsd"/>
```

Document Navigation

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

Quick Document Browsing Using Bookmarks

Using bookmarks, you are able to mark a position in an edited document so that you can return to it after further editing and browsing through one or more documents opened at the same time. You can place up to nine distinct bookmarks in any opened document. Configurable shortcut key strokes are available to place bookmarks and to return to any of the marked positions. To configure these keys, go to *Options > Menu shortcut keys*.

Figure 85: Editor Bookmarks

To place a bookmark, do one of the following:

- go to **Edit** > **Bookmarks** > **Create**;
- press **(F9)**;
- click the ⬤ **Bookmarks Quick Creation** toolbar button;
- click the margin of the editing area, to the left of the line number area, reserved for bookmarks.

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

Two contextual menu actions are available:

Remove

Removes the current bookmark.

Remove all

Removes all bookmarks set in the document.

Folding of the XML Elements

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

Figure 86: Folding of the XML Elements

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

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

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

- **Close Other FoldsCtrl+NumPad +/ (Command+NumPad+/ on OS X)** - Folds all the elements except the current element.

- **Collapse Child Folds (Ctrl+Decimal)** - Folds the elements indented with one level inside the current element.

- **Expand Child Folds (Ctrl+Equals)**- Unfolds all child elements of the currently selected element.

- **Expand All (Ctrl+NumPad+*)** - Unfolds all elements in the current document.

- **Toggle Fold** - Toggles the state of the current fold.

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

Outline View

The Outline view offers the following functionality:

- *XML Document Overview* on page 182
- *Outline Specific Actions* on page 183
- *Modification Follow-up* on page 183
- *Document Structure Change* on page 183
- *Document Tag Selection* on page 184
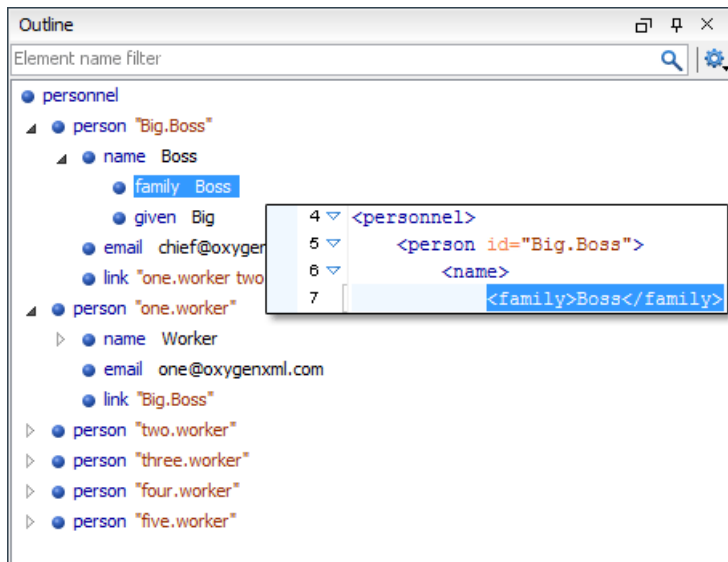
Figure 87: The Outline View

XML Document Overview

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

The **Outline** view allows you to:

- insert or delete nodes using pop-up menu actions;

- move elements by dragging them to a new position in the tree structure;
- highlight elements in the **Author** editor area.

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

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

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

Outline Specific Actions

The following actions are available in the **Settings** menu of the Outline view:

Filter returns exact matches

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

Flat presentation mode of the filtered results

When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions

Show/hide comments and processing instructions in the **Outline** view.

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

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

Configure displayed attributes

Displays the *XML Structured Outline preferences page*.

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

Modification Follow-up

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

Document Structure Change

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

- If you drag an XML element in the **Outline** view and drop it on another one in the same panel then the dragged element will be moved after the drop target element.
- If you hold the mouse pointer over the drop target for a short time before the drop then the drop target element will be expanded first and the dragged element will be moved inside the drop target element after its opening tag.

- You can also drop an element before or after another element if you hold the mouse pointer towards the upper or lower part of the targeted element. A marker will indicate whether the drop will be performed before or after the target element.
- If you hold down the **(Ctrl (Command on OS X))** key after dragging, there will be performed a copy operation instead of a move one.

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

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

The Popup Menu of the Outline Tree

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

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

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

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

Document Tag Selection

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

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

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

Navigation Buttons

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

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

Using the Go To Dialog

To navigate precisely to a part of the document you are editing in the **Text** mode, use the **Go to** dialog. To open the **Go to** dialog, go to **Find** > **Go to ... (Ctrl+L (Cmd+L on Mac))**.
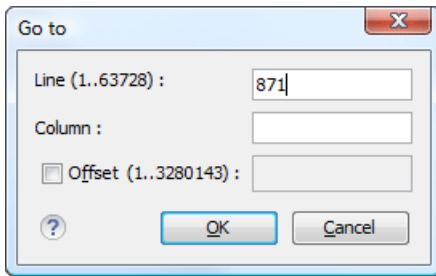
Figure 88: Go to Dialog

Complete the dialog as follows:

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

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). This means that each file is a valid XML instance and can be independently validated. It also means that the main document to which smaller instances are included can be validated without having to remove or comment out the DOCTYPE. as is the case with External Entities. This makes XInclude a more convenient and effective method for managing XML instances that need to be stand-alone documents and part of a much larger project.

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

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

Include a chapter in an article using XInclude

Create a chapter file and an article file in the `samples` folder of the Oxygen XML Author 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 declaration defines an entity that references a file containing the information to add the *xi* namespace to certain elements defined by the DocBook DTD.

- The href attribute of the xi:include element specifies that the `introduction.xml` file will replace the *xi:include* element when the document is parsed.

- If the `introduction.xml` file cannot be found, the parser will use the value of the *xi:fallback* element - a FIXME message.

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

```
<?xml 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 Author is turned on by default. To *toggle it*, *Open the **Preferences** dialog* and go to **XML** > **XML Parser** > **Enable XInclude processing**. When enabled, Oxygen XML Author will be able to validate and transform documents comprised of parts added using XInclude.

Working with XML Catalogs

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

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

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

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

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

Table 6: Catalog Mappings

Document	Referred Resource	Mappings
XML	DTD	*system* or *public*
		The ***Prefer*** option controls which one of the mappings should be used.
	XML Schema	The following strategy is used (if one step fails to provide a resource, the next is applied):
	Relax NG	1. resolve the schema using *URI* catalog mappings.
		2. resolve the schema using *system* catalog mappings.
	Schematron	This happens only if the ***Resolve schema locations also through system mappings*** option is enabled (it is by default);
	NVDL	
		3. resolve the root *namespace* using *URI* catalog mappings.

Document	Referred Resource	Mappings
XSL	XSL/ANY	*URI*
CSS	CSS	*URI*
XML Schema	XML Schema	The following strategy is used (if one step fails to provide a resource, the next is applied):
Relax NG	Relax NG	1. resolve schema reference using *URI* catalog mappings.
		2. resolve schema reference using *system* catalog mappings. This happens only if the ***Resolve schema locations also through system mappings*** option is enabled (it is by default);
		3. resolve schema *namespace* using *uri* catalog mappings. This happens only if the ***Process namespaces through URI mappings for XML Schema*** option is enabled (it is not by default);

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog
     PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN"
     "http://www.oasis-open.org/committees/entity/release/1.1/catalog.dtd">

<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">

    <!-- Use "system" and "public" mappings when resolving DTDs -->
    <system
          systemId="http://www.docbook.org/xml/4.4/docbookx.dtd"
          uri="frameworks/docbook/4.4/dtd/docbookx.dtd"/>
    <!-- The "systemSuffix" matches any system ID ending in a specified string -->
    <systemSuffix
          systemIdSuffix="docbookx.dtd"
          uri="frameworks/docbook/dtd/docbookx.dtd"/>

    <!-- Use "uri" for resolving XML Schema and XSLT stylesheets -->
    <uri
          name="http://www.oasis-open.org/docbook/xml/5.0/rng/docbook.rng"
          uri="frameworks/docbook/5.0/rng/docbookxi.rng"/>

    <!-- The "uriSuffix" matches any URI ending in a specified string -->
    <uriSuffix
          uriSuffix="docbook.xsl"
          uri="frameworks/docbook/xsl/fo/docbook.xsl"/>

</catalog>
```

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

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

> An XML catalog can be used to map a W3C XML Schema specified with an URN in the `xsi:noNamespaceSchemaLocation` attribute of an XML document to a local copy of the schema.
>
> Considering the following XML document code snippet:
>
> ```xml
> <topic xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>                   xsi:noNamespaceSchemaLocation="urn:oasis:names:tc:dita:xsd:topic.xsd:1.1">
> ```

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

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

Resolve Schemas Through XML Catalogs

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

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

XML Resource Hierarchy/Dependencies View

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

To open this view, go to **Window** > **Show View** > **Resource Hierarchy / Dependencies** . As an alternative, right click the current document and either select **Resource Hierarchy** or **Resource Dependencies**.
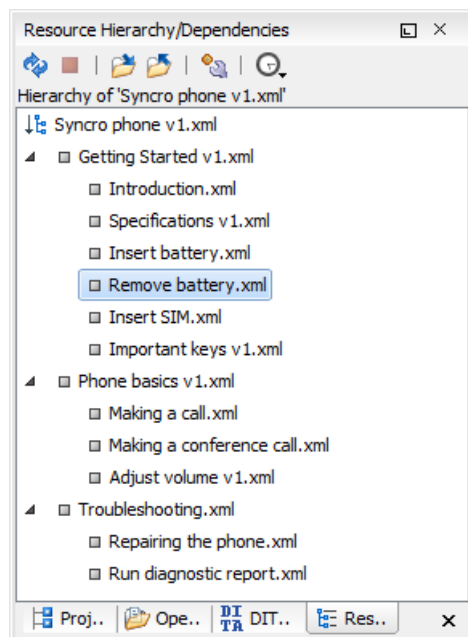
Figure 89: Resource Hierarchy/Dependencies View - Hierarchy for `Syncro phone v1.xml`

The build process for the dependencies view is started with the **Resource Dependencies** action available on the contextual menu.
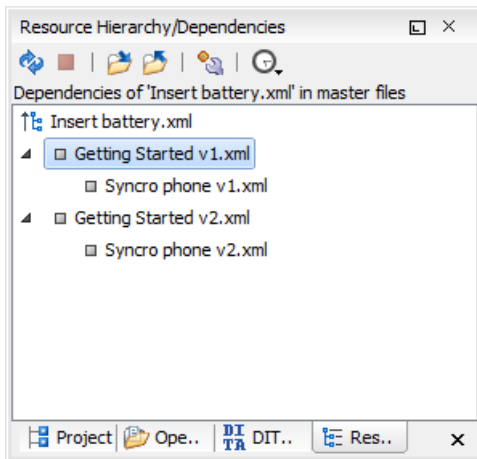
Figure 90: Resource Hierarchy/Dependencies View - Dependencies for `Insert battery.xml`

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

Refreshes the Hierarchy/Dependencies structure.

Stops the hierarchy/dependencies computing.

Allows you to choose a resource to compute the hierarchy structure.

Allows you to choose a resource to compute the dependencies structure.

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

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

The contextual menu contains the following actions:

Open

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

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the Master Files directory*.

Expand All

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

Collapse All

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

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

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

Moving/Renaming XML Resources

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

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

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

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

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

Formatting and Indenting XML Documents

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

- **Document** > **Source** > **Format and Indent** - formats and indents the whole document.

- **Document** > **Source** > **Indent Selection** - indents the current selection (but does not add line breaks)

- **Document** > **Source** > **Format and Indent Element**. - formats and indents the current element (the inmost nested element which contains the current caret) and its child-elements.

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

Significant and insignificant whitespace in XML

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

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

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

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

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

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

Here every whitespace character between the `codeblock` tags should be treated as significant.

How Oxygen XML Author determines when whitespace is significant

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

Ignore space

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

Normalize space

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

Mixed content

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

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

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

For example:

```
<p>The file is located in <i>HOME</i>/<i>USER</i>/hello. This is s <strong>big</strong>
```

```
<emphasis>deal</emphasis>.
</p>
```

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

Preserve space

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

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

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

To assign elements to these categories, Oxygen XML Author consults information from the following sources:

`xml:space`

If the XML element contains the `xml:space` attribute, the element is promoted to the appropriate category based on the value of the attribute.

CSS whitespace property

If the CSS stylesheet controlling the Author mode editor applies the `whitespace: pre` setting to an element, it is promoted to the preserve space category.

CSS display property

If a text node contains only white-spaces:

- If the node has a parent element with the CSS `display` property set to `inline` then the node is promoted to the mixed content category.
- If the left or right sibling is an element with the CSS `display` property set to `inline` then the node is promoted to the mixed content category.
- If one of its ancestors is an element with the CSS `display` property set to `table` then the node is assigned to the ignore space category.

Schema aware formatting

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

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

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

Schema aware formatting can be turned on and off.

- To turn it on or off for Author mode, *open the **Preferences** dialog* and go to **Editor** > **Edit modes** > **Author** > **Schema aware** > **Schema aware normalization, format and indent**.

- To turn it on or off for all other editor modes, *open the **Preferences** dialog* and go to **Editor** > **Format** > **XML** > **Schema aware format and indent**.

Preserve space elements list

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

Default space elements list

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

Mixed content elements list

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

Element content

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

Text node content

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

An exception to the rule

In general, a element can only be promoted to a more restrictive category (one that treats more whitespace as significant). However, there is one exception. In author mode, if an element is marked as mixed content in the schema, but the actual element contains no text content, it can be demoted to the space ignore category if all of its child elements are displayed as blocks by the associated CSS (that is, they have a CSS property of `display: block`). For example, in some schemas, a section or a table entry can be defined as having mixed content but in many cases they contain only block elements. In these cases, any whitespace they contain cannot be significant and they can be treated as space ignore elements. This exception can be turned on or off using the option *Editor / Edit modes / Author / Schema aware*.

How Oxygen XML Author formats and indents XML

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

- *Format Preferences* on page 753
- *XML Formatting Preferences* on page 754
- *Whitespaces Preferences* on page 755

When Oxygen XML Author formats and indents XML

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

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

Setting an indent size of zero

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

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

There are two cases to consider.

Maintaining zero indent in documents with zero indent

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

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

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

Enforcing zero indent for all documents

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

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

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

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

Editing Modular XML Files in the Master Files Context

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

You cat set a main XML document either using the *master files support from the **Project** view*, or using a validation scenario.

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

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

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

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

Managing ID/IDREFS.

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

Highlight IDs Occurrences in Text Mode

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

> **Note:** Highlighted ID declarations are rendered with a different color than highlighted ID references.

Search and Refactor Actions for ID/IDREFS

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

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

Rename in

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

Rename in File

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

> **Note:** Available in the **Text** mode only.

Search References in

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

Search References

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

Search Declarations in

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

Search Declarations

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

Search Occurrences in file

Searches for the declaration an references of the ID in the current document.

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

Selecting an ID for which you execute search or refactor operations differs from the **Text** mode to the **Author** mode. In the **Text** mode you position the caret inside the declaration or reference of an ID. In the **Author** mode Oxygen XML Author collects all the IDs by analyzing each element from the path to the root. In case more IDs are available, you are prompted to choose one of them.
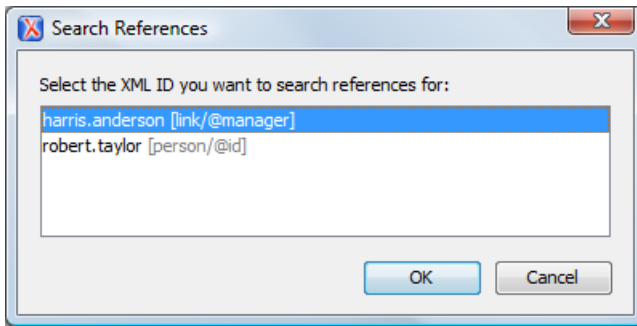
Figure 91: Selecting an ID in the Author Mode

Quick Assist Support for ID/IDREFS in Text Mode

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

The following actions are available:

Rename in

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

Search Declarations

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

Search References

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

Change scope

Opens the ***Select the scope for the Search and Refactor operations*** dialog;

Rename in File

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

> **Note:** Available in the **Text** mode only.

Search Occurrences

Searches for the declaration an references of the ID located at the caret position in the current document.

Search and Refactor Operations Scope

The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the **Change scope** operation, available in the Quick Assist action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project or to one or multiple working sets. The **Use only Master Files, if enabled** check-box allows you to restrict the scope of the search and refactor operations to the resources from the **Master Files** directory. Click **read more** for details about the *Master Files support*.
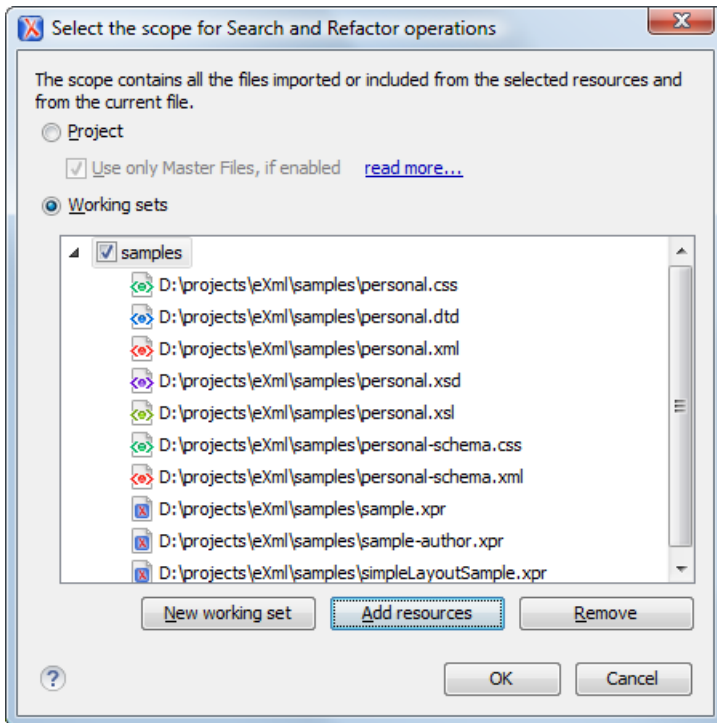
Figure 92: Change Scope Dialog

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

Viewing Status Information

Status information generated by the **Schema Detection**, **Validation**, **Automatic validation**, and **Transformation** threads are fed into the **Information** view allowing you to monitor how the operation is being executed.
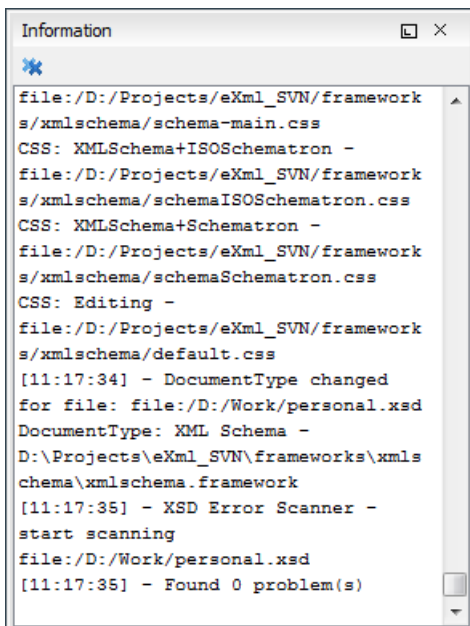
Figure 93: Information view messages

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

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

Image Preview

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

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

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

> **Note:** You can drag an image from the **Image Preview** view and drop it in a DITA, DocBook, or TEI document.

Making a Persistent Copy of Results

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

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

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

File > Save Results

Displays the **Save Results** dialog, used to save the result list of the current message tab. The action is also available on the right click menu of the **Results** panel.

File > Print Results

Displays the **Page Setup** dialog used to define the page size and orientation properties for printing the result list of the current **Results panel**. The action is also available on the right click menu of the **Results** panel.

Save Results as XML from the contextual menu

Saves the content of the **Results** panel in an XML file with the format:

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

Locking and Unlocking XML Markup

For documents with fixed markup such as forms in which the XML tags are not allowed to be modified but only their text content, editing of the XML tag names can be disabled and re-enabled with the action available from:

- **Document** main menu, **Source** > **Lock / Unlock the XML Tags** action;
- contextual menu, **Source** > **Lock / Unlock the XML Tags** action;
- 🔒 **Lock / Unlock the XML tags** toolbar action.

You can set the default lock state for all opened editors in the *Preferences XML Editor Format* preferences page.

Adjusting the Transparency of XML Markup

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

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

XML Editor Specific Actions

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

Split Actions

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

◀▮▶ **Split Editor Horizontally**

The currently edited file is displayed in two side-by-side editors. Useful when working with documents that require frequent scrolling between two area of interest.

▤ **Split Editor Vertically**

The currently edited file is displayed in two stacked editors. Useful when working with documents that require frequent scrolling between two area of interest.

▣ **Unsplit Editor**

Reverts the split editing mode to the usual, single editor, mode.

Edit Actions

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

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

Select Actions

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

- **Document** > **Select** > **Element** - Selects the entire current element;
- **Document** > **Select** > **Content** - Selects the content of the current element, excluding the start tag and end tag. If it is applied repeatedly, starts with selecting the XML element from the cursor position and extends the selection to the ancestor XML elements. Each execution of the action extends the current selection to the surrounding element;

- **Document** > **Select** > **Attributes** - Selects all the attributes of the current element;
- **Document** > **Select** > **Parent** - Selects the parent element of the current element;
- Triple click an element or processing instruction - If the triple click is done before the start tag of an element or after the end tag of an element then all the element is selected by the triple click action. If it is done after the start tag or before the end tag then only the element content without the start tag and end tag is selected;
- Triple click an attribute in **Text** mode - If the triple click is performed before the start tag of an attribute or after its end tag, the entire attribute is selected by the triple click action. If it is performed after the start tag or before the end tag, only the attribute content (without the start tag and end tag) is selected;
- Double click after the opening quote or before the closing quote of an attribute value - Select the whole attribute value.

Source Actions

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

Document > Source > 🔒 Lock / Unlock the XML Tags

Disables / Enables editing of XML tags.

Document > Source > To Lower Case

Converts the selection content to lower case characters.

Document > Source > To Upper Case

Converts the selection content to upper case characters.

Document > Source > Capitalize lines

Converts to upper case the first character of every selected line.

Document > Source > 📧 Shift Right <u>Tab</u>

Shifts the currently selected block to the right.

Document > Source > 📧 Shift Left <u>Shift+Tab</u>

Shifts the selected block to the left.

Document > Source > ⁺& Escape Selection ...

Escapes a range of characters by replacing them with the corresponding character entities.

Document > Source > &⁺ Unescape Selection ...

Replaces the character entities with the corresponding characters.

Document > Source > ≣ Indent selection <u>Ctrl+I (Command+I on OS X)</u>

Corrects the indentation of the selected block of lines if it does not follow the current *indenting preferences of the user*.

Document > Source > ≣ Format and Indent Element <u>Ctrl+Shift+I</u>

Pretty prints the element that surrounds the caret position.

Document > Source > 🗎 Insert XInclude

Shows a dialog that allows you to browse and select the content to be included and generates automatically the corresponding XInclude instruction.

> **Note:** In the **Author** mode, this dialog presents a preview of the inserted document as an author page in the **preview** tab and as a text page in the **source** tab. In the **Text** mode only the **source** tab is presented.

Document > Source > Import entities list

Shows a dialog that allows you to select a list of files as sources for external DTD entities. The internal subset of the DOCTYPE declaration of your document will be updated with the chosen entities. For instance, if choosing the file `chapter1.xml` and `chapter2.xml`, the following section is inserted in the DOCTYPE:

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

Document > Source > Capitalize lines

It capitalizes the first letter found on every new line that is selected. Only the first letter is affected, the rest of the line remains the same. If the first character on the new line is not a letter then no changes are made.

Document > Source > Join and Normalize Lines

The action works on the selection. It joins the lines by replacing the *line separator* with a single space character. It also normalizes the whitespaces by replacing a sequence of such characters with a single space.

Document > Source > Insert new line after

This useful action has the same result with moving the caret to the end of the current line and pressing the *ENTER* key.

XML Document Actions

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

- **Document** > **Schema** > **Show Definition** (also available on the contextual menu of the editor panel) - Moves the cursor to the definition of the current element or attribute in the schema (DTD, XML Schema, Relax NG schema) associated with the edited XML document. In case the current attribute is "type" belonging to the "*http://www.w3.org/2001/XMLSchema-instance*" namespace, the cursor is moved in the XML schema, to the definition of the type referred in the value of the attribute.

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

  - **Ctrl+Shift+ENTER (Command+Shift+ENTER on OS X)** on your keyboard;
  - **Ctrl+Click (Command+Click on OS X)** an element or attribute name.

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

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

  Oxygen XML Author fully implements the W3C recommendation regarding *Associating Style Sheets with XML documents*. See also *Specifying external style sheets* in HTML documents.

You can use the **Ctrl+Click (Command+Click on OS X)** shortcut to open:

- any absolute URLs (URLs that have a protocol) regardless of their location in the document;

- URI attributes such as: *schemaLocation*, *noNamespaceSchemaLocation*, *href* and others;
- processing instructions used for associating resources, xml-models, xml-stylesheets.

XML Refactoring Actions

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

- **Document** > **XML Refactoring** > ⚄ **Surround with tag...** <u>**Ctrl+E (Command + E on OS X)**</u> - 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

- **Document** > **XML Refactoring** > ⚄ **Surround with <tag>** <u>**(Ctrl+/)**</u> - Similar in behavior with the **Surround with tag...** action, except that it inserts the last tag used by the **Surround with tag...** action.

- **Document** > **XML Refactoring** > ⬆ **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.

  Document > **XML Refactoring** > ⬆ **Rename prefix** > <u>**Alt+Shift+P**</u> > - 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 Author will introduce a new declaration `xmlns:p5="ns1"` in the current element and will change the prefix from `p1` to `p5`. If `p5` is already associated in the current element with another namespace, let's say `ns5`, then a dialog showing the conflict will be displayed. Pressing the **OK** button, the prefix will be modified from `p1` to `p5` without inserting a new declaration `xmlns:p5="ns1"`. On **Cancel** no modification is made.

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

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

- **Document** > **XML Refactoring** > ⧖ **Split element** <u>**Ctrl+Alt+D**</u> - Split the element from the caret position in two identical elements. The caret must be inside the element.

- **Document** > **XML Refactoring** > ⧗ **Join elements** <u>**Ctrl+Alt+J**</u> - Joins the left and right elements relative to the current caret position. The elements must have the same name, attributes and attributes values.

- **Document** > **XML Refactoring** > ⬰ **Delete element tags** <u>**Ctrl+Alt+X (Command+Alt+X on Mac OS)**</u> - 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:
- *Auto-rename matching tag* - When you edit the name of the start tag, Oxygen XML Author will mirror-edit the name of the matching end tag. This feature can be controlled from the *Content Completion option page*.
- *Auto-breaking the edited line* - The *Hard line wrap option* breaks the edited line automatically when its length exceeds the maximum line length *defined* for *the format and indent operation*.
- *Indent on Enter* - The *Indent on Enter option* indents the new line inserted when Enter is pressed.

- *Smart Enter* - The *Smart Enter option* inserts an empty line between the start and end tags. If Enter is pressed between a start and an end tag the action places the cursor in an indented position on the empty line between the lines that contain the start and end tag.
- *Double click* - A double click selects a different region of text of the current document depending on the position of the click in the document:

  - if the click position is inside a start tag or an end tag then the entire element enclosed by that tag is selected;
  - if the click position is immediately after a start tag or immediately before an end tag then the entire content of the element enclosed by that tag is selected, including all the child elements but excluding the start tag and the end tag of the element;
  - otherwise, the double click selects the entire current line of text.

Syntax Highlight Depending on Namespace Prefix

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

Figure 94: Example of Coloring XML Tags by Prefix

Editor Highlights

An editor highlight is a text fragment emphasized by a colored background. Highlights are generated in both **Editor** and **Author** mode, when the following actions generate results: **XPath**, **Find All**, and **Find in Files**.

By default, Oxygen XML Author uses a different color for each type of highlight: *XPath*, *Find*, *Search References*, and *Search Declarations*. You can customize these colors and the maximum number of highlights displayed in a document on the *Editor preferences page*. The default maximum number of highlights is 10000.

You are able to navigate in the current document through the highlights using one of the following methods:

- clicking the markers from the range ruler, located at the right side of the document;
- clicking the **Next** and **Previous** buttons from the bottom of the range ruler;

  > **Note:** When there are multiple types of highlights in the document, the **Next** and **Previous** buttons navigate through highlights of the same type.

- clicking the messages displayed in the *Results view*.

To remove the highlights, you can:

- click the **Remove all** button from bottom of the range ruler;
- close the results tab which contains the output of the action that generated the highlights;
- click the ✱ **Remove all** button from the results panel.

  > **Note:** Use the **Highlight all results in editor** button to either display all the highlights or hide them.

Batch Editing Actions on Highlights

Working with XML documents implies frequent changes to structure and content. You are often faced with a situation where you need to make a slight change in hundreds of places in the same document. Oxygen XML Author introduced a new feature, **Manage Highlighted Content**, designed to help you achieve that.

When you are in **Text** mode and you perform a search operation or apply an XPath that highlights more than one result, you can select the **Manage Highlighted Content** action from the contextual menu of any highlight in the document. In the sub-menu, the following options are available:

- **Modify All** - Use this option to modify in-place all the occurrences of the selected content. When you use this option, a thin rectangle replaces the highlights and lets you start editing;

  > **Note:** In case you select a very large number of highlights that you want to modify using this feature, when you select this option a dialog informs you that you may experience performance issues. You have the option to either use the **Find/Replace** dialog box, or continue the operation.

- **Surround All** - Use this option to surround the content with a specific tag. This option opens the **Tag** dialog box. The **Specify the tag** drop-down presents all the available elements that you can choose from.
- **Remove All** - Removes all the highlighted content.

In case you right click a different part of the document than a highlight, you only have the option to select **Modify All Matches**.

XML Quick Fixes

The Oxygen XML Author Quick Fix support helps you resolve errors that appear in an XML document by offering quick fixes to problems like missing required attributes or invalid elements. Quick fixes are available only for XML documents with XSD schemas and that are validated with the Xerces validation engine.

Quick fixes are available in **Text** mode:
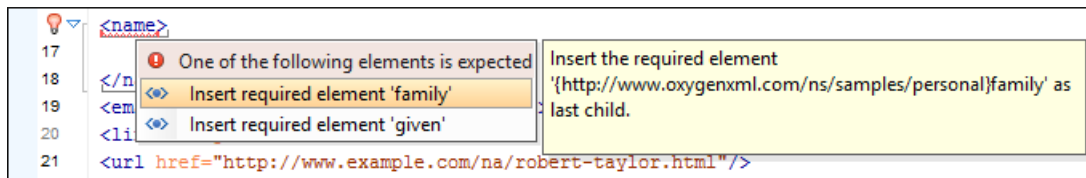
and **Author** mode:

To activate this feature, when Oxygen XML Author finds a validation error in an XML document, place the caret in the highlighted area of text. If Oxygen XML Author can provide a quick fix for that error, the 💡 icon is displayed in the left side stripe. When you click this icon, the list of available fixes is displayed. Also, you can invoke the quick fix menu if you press **Alt + 1 (Meta + Alt + 1 on OS X)** on your keyboard.

Oxygen XML Author provides quick fixes for the following cases:

Problem type	Available quick fixes
The content of the element should be empty	Remove the element content
The attribute has a fixed value	Set the correct attribute value

Problem type	Available quick fixes
A specific element is required in the current context	Insert the required element
An element is invalid in the current context	Remove the invalid element
A required attribute is missing	Insert the required attribute
An attribute is not allowed to be set for the current element	Remove the attribute
An element is not allowed to have child elements	Remove all child elements
ID value already defined	Generate unique ID value
References to an invalid ID	Change the reference to an already defined ID

> **Note:** A quick fix that adds an element inserts it together with required and optional elements, and required and fixed attributes, depending on how the *Content Completion options* are set.

Editing XHTML Documents

XHTML documents with embedded CSS, JS, PHP, and JSP scripts are rendered with dedicated coloring schemes. To customize them, *open the **Preferences** dialog* and go to **Editor** > **Colors**.

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

The CSS properties accepted by the validator are the ones included in the current CSS profile that is selected in *the CSS validation preferences*. The **CSS 3 with Oxygen extensions** profile includes all the CSS 3 standard properties and the *CSS extensions specific for Oxygen* that can be used in *Author mode*. That means all Oxygen specific extensions are accepted in a CSS stylesheet by *the built-in CSS validator* when this profile is selected.

Specify Custom CSS Properties

To specify custom CSS properties, follow these steps:

1. Create a file named CustomProperties.xml, that has the following structure:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<css_keywords
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.oxygenxml.com/ns/css http://www.oxygenxml.com/ns/css/CssProperties.xsd"

    xmlns="http://www.oxygenxml.com/ns/css">
    <property name="custom">
    <summary>Description for custom property.</summary>
      <value name="customValue"/>
      <value name="anotherCustomValue"/>
    </property>
</css_keywords>
```

2. Go to your desktop and create the builtin/css-validator/ folder structure.

3. Press and hold **Shift** and right click your desktop. From its contextual menu, select **Open Command Window Here**.

4. In the command line, run the jar cvf custom_props.jar builtin/ command.
The custom_props.jar file is created.

5. Go to [OXYGEN_DIR]/lib and create the endorsed folder. Copy the custom_props.jar file to [OXYGEN_DIR]/lib/endorsed.

Content Completion in CSS Stylesheets

A **Content Completion Assistant** 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 (Command+Space on OS X)** shortcut and it is context-sensitive when invoked for the value of a property.

The properties and the values available are dependent on the CSS Profile selected in the *CSS preferences* . The CSS 2.1 set of properties and property values is used for most of the profiles, excepting CSS 1 and CSS 3. For these two, specific proposal sets are used.

The profile **CSS 3 with Oxygen extensions** includes all the CSS 3 standard properties and the *CSS extensions specific for Oxygen* that can be used in *Author mode*.

CSS Outline View

The CSS **Outline** view presents the import declarations for other CSS stylesheet files and all the selectors defined in the current CSS document. The selector entries can be presented as follows:

- in the order they appear in the document;
- sorted by element name used in the selector;
- sorted by the entire selector string representation.

You can synchronize the selection in the **Outline** view with the caret moves or the changes you make in the stylesheet document. When you select an entry from the **Outline** view, Oxygen XML Author highlights the corresponding import or selector in the CSS editor.

The selectors presented in this view can be quickly found using the key search field. When you press a sequence of character keys while the focus is in the view, the first selector that starts with that sequence is selected automatically.

Folding in CSS Stylesheets

In a large CSS stylesheet document, some styles can be collapsed so that only the needed styles remain in focus. The same *folding features available for XML documents* are also available in CSS stylesheets.

> **Note:** To enhance your editing experience, you can select entire blocks (parts of text delimited by brackets) by double-clicking after an opening or in front of a closing bracket.

Formatting and Indenting CSS Stylesheets (Pretty Print)

If the edited CSS stylesheet becomes unreadable because of the bad alignment of the text lines, the *format and indent operation available for XML documents* is also available for CSS stylesheets. It works in the same way as for XML documents and is available as the same menu and toolbar action.

Minifying CSS Stylesheets

Minification (or *compression*) of a CSS document is the practice of removing unnecessary code without affecting the functionality of the stylesheet.

To minify a CSS, invoke the contextual menu anywhere in the edited document and choose the **Minify CSS...** action. Oxygen XML Author opens a dialog that allows you to:

- set the location of the resulting CSS
- place each style rule on a new line

After pressing **OK**, Oxygen XML Author performs the following actions:

- all spaces are normalized (all leading and trailing spaces are removed, while sequences of white spaces are replaced with single space characters)
- all comments are removed

> **Note:** The CSS minifier relies heavily upon the W3C CSS specification. If the content of the CSS file you are trying to minify does not conform with the specifications, an error dialog will pop-up, listing all errors encountered during the processing.

The resulting CSS stylesheet gains a lot in terms of execution performance, but loses in terms of readability. The source CSS document is left unaffected.

> **Note:** To restore the readability of a minified CSS, invoke the **Format and Indent** action. However, this action will not recover any of the deleted comments.

Other CSS Editing Actions

The CSS editor type offers a reduced version of *the popup menu available in the XML editor*. Only *the folding actions, the edit actions* and a part of *the source actions* (only the actions **To lower case**, **To upper case**, **Capitalize lines**) are available.

Editing StratML Documents

Strategy Markup Language (StratML) is an XML vocabulary and schema for strategic plans. Oxygen XML Author supports StratML Part 1 (Strategic Plan) and StratML Part 2 (Performance Plans and Reports) and provides templates for the following documents:

*   **Strategic Plan** (StratML Part 1)
*   **Performance Plan** (StratML Part 2)
*   **Performance Report** - (StratML Part 2)
*   **Strategic Plan** - (StratML Part 2)

You can view the components of a StratML document in the **Outline** view. Oxygen XML Author implements a default XML with XSLT transformation scenario for this document type, called StratML to HTML.

Editing JavaScript Documents

This section explains the features of the Oxygen XML Author JavaScript Editor and how you can use them.

JavaScript Editor Text Mode

Oxygen XML Author allows you to create and edit JavaScript files and assists you with useful features such as syntax highlight, content completion, and outline view. To enhance your editing experience, you can select entire blocks (parts of text delimited by brackets) by double-clicking after an opening or in front of a closing bracket.

```
12 ▽ function newPage(filename, overlay) {
13        divs = document.getElementsByTagName("div");
14
15 ▽     if (divs) {
16      var xdiv = divs[0];
17
18 ▽    if (xdiv) {
19          var xid = xdiv.getAttribute("id");
20
21          var mytoc = window.top.frames[0];
22 ▽        if (mytoc.lastUnderlined) {
23          mytoc.lastUnderlined.style.textDecoration = "none";
24          }
25
26          var tdiv = xbGetElementById(xid, mytoc);
27
28 ▽        if (tdiv) {
29          var ta = tdiv.getElementsByTagName("a").item(0);
30          ta.style.textDecoration = "underline";
31          mytoc.lastUnderlined = ta;
32          }
33      }
34      }
35
36 ▽     if (overlay != 0) {
37      overlaySetup('lc');
38      }
```

Figure 95: JavaScript Editor Text Mode

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

Cut

Allows you to cut fragments of text from the editing area.

Copy

Allows you to copy fragments of text from the editing area.

Paste

Allows you to paste fragments of text in the editing area.

Toggle comment

Allows you to comment a line or a fragment of the JavaScript document you are editing. This option inserts a single comment for the entire fragment you want to comment.

Toggle line comment

Allows you to comment a line or a fragment of the JavaScript document you are editing. This option inserts a comment for each line of the fragment you want to comment.

Go to matching bracket

Use this option to find the closing, or opening bracket, matching the bracket at the caret position. When you select this option, Oxygen XML Author moves the caret to the matching bracket, highlights its row, and decorates the initial bracket with a rectangle.

> **Note:** A rectangle decorates the opening, or closing bracket which matches the current one at all times.

Compare

Select this option to open the **Diff Files** dialog and compare the file you are editing with a file you choose in the dialog.

Open

Allows you to select one of the following actions:

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

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

Folding

Allows you to select one of the following actions:

Toggle Fold

Toggles the state of the current fold.

Collapse Other Folds <u>(Ctrl (Meta on Mac OS)+NumPad /)</u>

Folds all the elements except the current element.

Collapse Child Folds <u>(Ctrl (Meta on Mac OS)+NumPad .)</u>

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 <u>(Ctrl (Meta on Mac OS)+NumPad *)</u>

Unfolds all elements in the current document.

Source

Allows you to select one of the following actions:

To Lower Case

Converts the selection content to lower case characters.

To Upper Case

Converts the selection content to upper case characters.

Capitalize Lines

Converts to upper case the first character of every selected line.

Join and Normalize Lines

Joins all the rows you select to one row and normalizes the content.

Insert new line after

Inserts a new line after the line at the caret position.

Content Completion in JavaScript Files

When you edit a JavaScript document, the *Content Completion Assistant* presents you a list of the elements you can insert at the caret position. For an enhanced assistance, JQuery methods are also presented. The following icons decorate the elements in the content completion list of proposals depending on their type:

- $f_{()}$ - function;
- V - variable;
- $\square$ - object;
- - property;
- $f_{()}$ - method.

> **Note:** These icons decorate both the elements from the content completion list of proposals and from the **Outline** view.

```
12 ▽ function newPage(filename, overlay) {
13      divs = document.getElementsByTagName("div");
14
15 ▽    if (divs) {
16    var xdiv = divs[0];
17
18 ▽    if (xdiv) {
19        var xid =  □ TypeInfo - TypeInfo              ▲
20                   □ UIEvent - UIEvent
21        var mytoc □ UserDataHandler - UserDataHandler
22 ▽      if (mytoc f() alert(msg)
23    mytoc.lastU  f() blur()
24        }         f() clearInterval(id_setinterval)   ▭
25                  f() clearTimeout(id_settimeout)
26        var tdiv                                       ▼
27
28 ▽      if (tdiv) {
29    var ta = tdiv.getElementsByTagName("a").item(0);
30    ta.style.textDecoration = "underline";
31    mytoc.lastUnderlined = ta;
32        }
33    }
34      }
35
36 ▽    if (overlay != 0) {
37    overlaySetup('lc');
38      }
```

Figure 96: JavaScript Content Completion Assistant

The **Content Completion Assistant** collects:

*   method names from the current file and from the library files;
*   functions and variables defined in the current file.

In case you edit the content of a function, the content completion list of proposals contains all the local variables defined in the current function, or in the functions that contain the current one.

JavaScript Outline View

Oxygen XML Author present a list of all the components of the JavaScript document you are editing in the **Outline** view. To open the **Outline** view, go to **Window** > **Show View** > **Outline**.
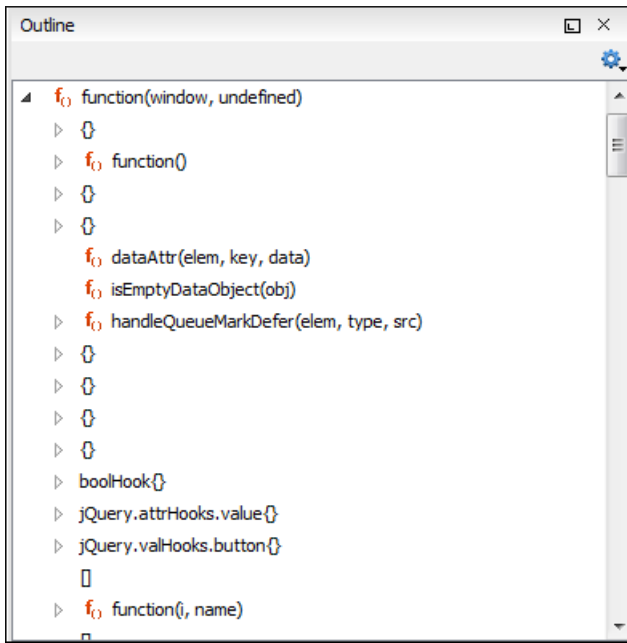
Figure 97: The JavaScript Outline View

The following icons decorate the elements in the **Outline** view depending on their type:

- $f_{()}$ - function;
- V - variable;
- $\square$ - object;
- ● - property;
- $f_{()}$ - method.

> **Note:** These icons decorate both the elements from the content completion list of proposals and from the **Outline** view.

The contextual menu of the JavaScript **Outline** view contains the usual ✂ **Cut**, 📋 **Copy**, 📋 **Paste**, and ✖ **Delete** actions. From the settings menu, you can enable the Update selection on caret move option to synchronize the **Outline** view with the editing area.

Validating JavaScript Files

You have the possibility to validate the JavaScript document you are editing. Oxygen XML Author uses the Mozilla Rhino library for validation. For more information about this library, go to *http://www.mozilla.org/rhino/doc.html*. The JavaScript validation process checks for errors in the syntax. Calling a function that is not defined is not treated as an error by the validation process. The interpreter discovers this error when executing the faulted line. Oxygen XML Author can validate a JavaScript document both on-request and automatically.

Editing SVG Documents

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

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

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

> 📄 **Note:** Batik partially supports SVG 1.1. Here you can find a detailed list of supported elements, attributes and properties: *Batik implementation status*.

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

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

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

The Standalone SVG Viewer

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

Zoom in

Zooms in the image by a factor of 2. The action is also available on **Mouse Wheel Up**;

Zoom out

Zooms out the image by a factor of 2. The action is also available on **Mouse Wheel Down**;

Rotate

Rotates the image 90 degrees clockwise;

Refresh

Refreshes the image, by reloading the SVG file.

> 📄 **Note:** When you open the SVG viewer, the active SVG is displayed.
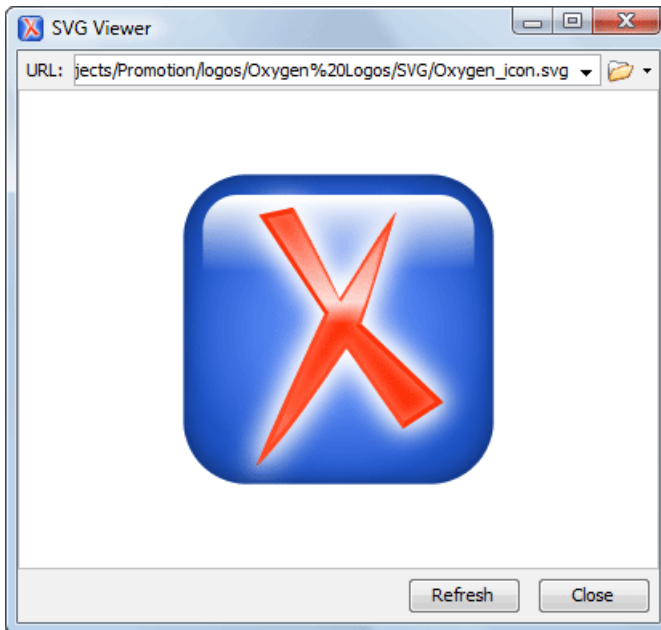
Figure 98: SVG Viewer

The Preview Result Panel

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

Figure 99: Integrated SVG Viewer

The basic use-case of Oxygen XML Author consists in the development of the XSL stylesheets capable of producing rich SVG graphics. For example, you have an XML document describing the evolution of a parameter over time and you create a graphic from it. You can start with a static SVG, written directly in Oxygen XML Author or exported from a graphics tool like the Adobe suite. Extract then the parts that are dependent of the data from the XML document and create the XSL templates. Select the option **Show as SVG** in *the dialog for configuring the XSLT transformation scenario*. When you run the transformation, the SVG result is displayed in the SVG result panel.

Spell Checking

The **Spelling** dialog allows you to check the spelling of the edited document. To open this dialog, click the **Check Spelling** toolbar button.
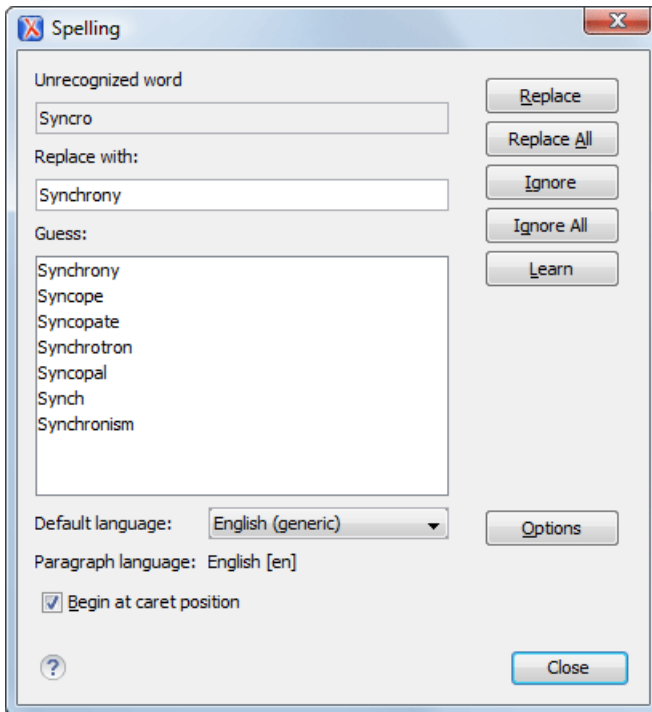
Figure 100: The Check Spelling Dialog

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 Author 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 Author 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 Author: **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 Author.

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 Author comes with the following built-in dictionaries for the Hunspell checker:

• English (US);
• English (UK);
• French;
• German;
• Spanish.

Each language-country variant combination has its specific dictionary. If you cannot find a Hunspell dictionary that is already built for your language, you can build the dictionary you need. To build a dictionary from this list follow *these instructions*.

Adding a Dictionary and Term Lists for the Hunspell Checker

To add a new spelling dictionary to Oxygen XML Author or to replace an existing one follow these steps:

1. *Download* the files you need for your language dictionary.
2. The downloaded `.oxt` file is a *zip* archive. Copy the `.aff` and `.dic` files from this archive in the `spell` subfolder of the Oxygen XML Author preferences folder, if you are creating a new dictionary.

   The Oxygen XML Author preferences folder is `[APPLICATION-DATA-FOLDER]/com.oxygenxml.author>`, where `[APPLICATION-DATA-FOLDER]` is:

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

3. Copy the `.aff` and `.dic` files into the folder `[OXYGEN_DIR]/dicts` if you are updating an existing dictionary.
4. Restart the application after copying the dictionary files.

   📄 **Note:** You can setup Oxygen XML Author to use dictionaries and term lists from a custom location configured in *the **Dictionaries** preferences page*.

Dictionaries for the Java Checker

A Java spell checker dictionary has the form of a `.dar` file located in the directory `[OXYGEN_DIR]/dicts`. Oxygen XML Author comes with the following built-in dictionaries for the Java checker:

• English (US)
• English (UK)
• English (Canada)
• French (France)
• French (Belgium)
• French (Canada)
• French (Switzerland)
• German (old orthography)
• German (new orthography)
• Spanish

A pre-built dictionary can be added by copying the corresponding `.dar` file to the folder `[OXYGEN_DIR]/dicts` and restarting Oxygen XML Author. There is one dictionary for each language-country variant combination.

Learned Words

Spell checker engines rely on dictionary to decide that a word is correctly spelled. To tell the spell checker engine that an unknown word is actually correctly spelled, you need to add that word to its dictionary. There are two ways to do so:

- press the **Learn** button from the **Spelling** dialog;
- invoke the contextual menu on an unknown word, then press **Learn word**.

Learned words are stored into a persistent dictionary file. Its name is composed of the currently checked language code and the `.tdi` extension, for example `en_US.tdi`. It is located in the:

- `[HOME_DIR]/Application Data/com.oxygenxml.author/spell` folder on Windows XP;
- `[HOME_DIR]/AppData/Roaming/com.oxygenxml.author/spell` folder on Windows Vista;
- `[HOME_DIR]/Library/Preferences/com.oxygenxml.author/spell` folder on Mac OS X;
- `[HOME_DIR]/com.oxygenxml.author/spell` folder on Linux.

> **Note:** To change this folder go to the *Editor > Spell Check > Dictionaries preferences page*.

> **Note:** To delete items from the list of learned words, press **Delete learned words** in the *Editor > Spell Check > Dictionaries preferences page*.

Ignored Words

The content of some XML elements like `programlisting`, `codeblock` or `screen` should always be skipped by the spell checking process. The skipping can be done manually word by word by the user using the **Ignore** button of *the Spelling dialog* or, more conveniently, automatically by maintaining a set of known element names that should never be checked. You maintain this set of element names *in the user preferences* as a list of XPath expressions that match the elements.

Only a small subset of XPath expressions is supported, that is only the '/' and '//' separators and the '*' wildcard. Two examples of supported expressions are `/a/*/b` and `//c/d/*`.

Automatic Spell Check

To allow Oxygen XML Author to automatically check the spelling as you write, you need to enable the **Automatic spell check** option from the *Spell Check* preferences page. Unknown words are highlighted and feature a contextual menu which offers the following actions:

Delete Repeated Word

Allows you to delete repeated words.

Learn Word

Allows you to add the current unknown word to the persistent dictionary.

Spell check options

Opens the *Spell Check preferences page*.

Also, a list of words suggested by the spell checking engine as possible replacements of the unknown word is offered in the contextual menu.

Spell Checking in Multiple Files

The **Check Spelling in Files** action allows you to check the spelling on multiple local or remote documents. This action is available in:

- the **Edit** menu;
- the contextual menu of the **Project** view;
- the contextual menu of the **DITA Maps Manager** view.

The spelling corrections are displayed in *the Results view*, that allows you to group the reported errors as a tree with two levels.
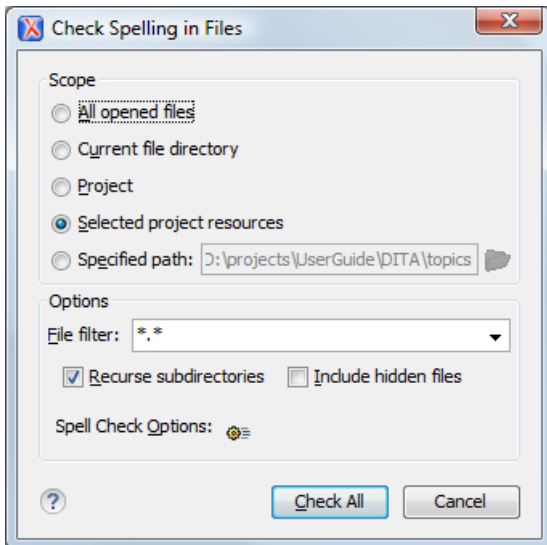
Figure 101: Check Spelling in Files Dialog

The following scopes are available:

- **All opened files** - spell check in all opened files;
- **Directory of the current file** - all the files from the folder of the current edited file;
- **Project files** - all files from the current project;
- **Selected project files** - the selected files from the current project;
- **Specified path** - checks the spelling in the files located at a path that you specify.

When you invoke the **Check Spelling in Files** action in the **DITA Maps Manager** view, a different dialog is displayed:
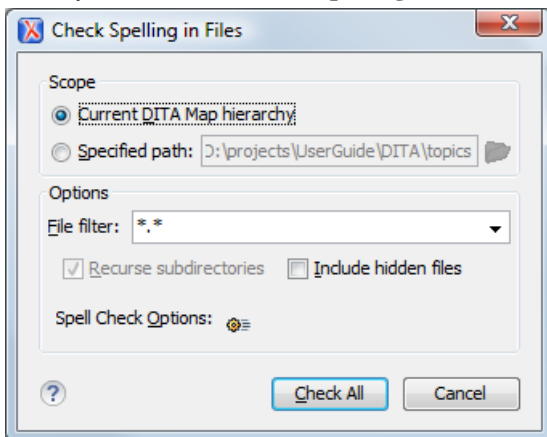
Figure 102: Check Spelling in Files Dialog in The DITA Maps Manager View

The following scopes are available:

- **Current DITA Map hierarchy** - All the files referred in the current DITA map opened in the **DITA Maps Manager** view;
- **Specified path** - checks the spelling in the files located at a path that you specify;

You can also choose a file filter, decide whether to recurse subdirectories or process hidden files.

The spell checker processor uses the options available in the *Spell Check preferences panel* .

Editing Large Documents

When you open a document with a file size larger than the limit configured in *Open/Save preferences*, Oxygen XML Author prompts you to choose whether you want to optimize the loading of the document for large files or for huge files.

If your file has a size smaller than 300 MB, the recommended approach is *Optimize loading for large files*. For documents that exceed 300 MB the recommended approach is *Optimize loading for huge files*.

File sizes smaller than 300 Megabytes

For editing large documents (file size up to 300 Megabytes), a special memory optimization is implemented on loading such a file so that the total memory allocated for the application is not exceeded.

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

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

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

- Less precise localizations for the results of an *XPath expression*.

XML file sizes greater than 300 MB

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

The loaded file is split in multiple pages (each having about 1MB in size). Each page is individually loaded (and edited) in the **Text** mode by using the special horizontal slider located at the top of the editing area. The loading operation is very fast and has no upper limit for the size of the loaded file.

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

- only XML files UTF-8, UTF-16, and ASCII encoded are loaded;
- a file larger than the value of the above option is edited only in Text mode;
- the *automatic validation* is not available when editing a very large file;
- the XPath filter is disabled in *the Find/Replace dialog*;
- the bidirectional Unicode support (right-to-left writing) is disabled;
- *the option **Format and indent the document on open*** is disabled for non-XML documents. For XML documents, it is done optimizing the memory usage but without respecting the options set in *the **Format** preferences page*;

- the **Outline** view is not supported;

- the editor is wrapped by default;

- the **Find/Replace** dialog only supports the **Find** action;

- saving changes is possible if **Safe save** is activated;

- the **undo** operation is not available if you go to other pages and come back to the modified page.

Scratch Buffer

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

Handling Read-Only Files

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

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

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

Editing Documents with Long Lines

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

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

The last two restrictions are valid only for XML documents.

Associating a File Extension with Oxygen XML Author

To associate a file extension with Oxygen XML Author on Windows:

- go to the **Start** menu and click **Control Panel**;
- go to **Default Programs**;
- click **Associate a file type or protocol with a program**;
- click the file extension you want to associate with Oxygen XML Author, then click **Change program**;
- in the **Open With** dialog click **Browse** and navigate to Oxygen XML Author.

To associate a file extension with Oxygen XML Author on Mac:

- In **Finder**, right click a file and from the contextual menu select **Get Info**;
- In the **Open With** subsection, select **Other** from the application combo and browse to Oxygen XML Author;
- With Oxygen XML Author selected, click **Change All**.

Chapter

7

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 a number of different topic types, the most common of which are:

- *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*.

Editing DITA Maps

Oxygen XML Author provides a view for managing and editing *DITA Maps*. The **DITA Maps Manager** view presents a DITA Map as a table-of-contents. It allows you to navigate to the topics and maps referenced in the maps, make changes, and apply transformation scenarios to obtain various output formats.
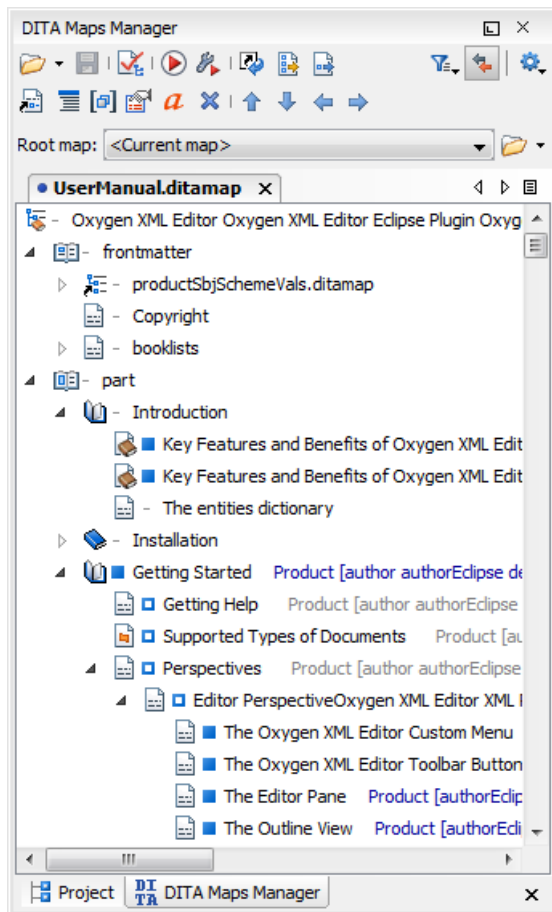
Figure 103: The DITA Maps Manager View

The **DITA Maps Manager** view supports multiple open maps at the same time.

When your open a file with the extension `.ditamap` or `.bookmap`, Oxygen XML Author offers you the choice of opening it the XML editor or in the **DITA Maps Manager**. In addition, you can right click a map file in the **Project** view and select **Open with**. If you have a ditamap file open in the XML editor, you can open it in the **DITA Maps Manager** by right clicking on the title tab and selecting the **Open in DITA Maps Manager**.

You can move topics in the same map or between different maps by dragging and dropping them into the desired position. Also, you can copy topics by dragging them while pressing the **Ctrl (Command on OS X)** key.

The toolbar includes the actions which are also available on menu **DITA Maps**:

Reopen

You can use this drop-down menu to reopen recently viewed DITA Maps. Apart from the history of the recently viewed DITA Maps, the drop-down menu also contains the **Clear history** , **Open**, and **Open URL** actions.

Clear history

Removes the list with the recently viewed DITA Maps.

Open

Allows opening the map in the **DITA Maps Manager** view. You can also open a map by dragging it in the **DITA Maps Manager** view from the file system explorer.

Open URL

Allows opening remote maps in the **DITA Maps Manager** view. See *Open URL* for details.

Open from Data Source Explorer

Allows you to open a DITA Map from a data source explorer.

Save (Ctrl (Meta on Mac OS)+S)

Saves the current DITA Map.

Validate and Check for Completeness

Checks the validity and integrity of the map.

Apply Transformation Scenario(s)

Applies the DITA Map transformation scenario that is associated with the current map from the view.

Configure Transformation Scenario(s)...

Allows *associating a DITA Map transformation* scenario with the current map.

Refresh References

You can use this action to manually trigger a refresh and update of all titles of referred topics. This action is useful when the referred topics are modified externally. When they are modified and saved from the Oxygen XML Author Author, the DITA Map is updated automatically.

Open Map in Editor with Resolved Topics

Opens the DITA Map in the main editor area with content from all topic references expanded in-place. Content from the referenced topics is presented as read-only and you have to use the contextual menu action **Edit Reference** to open the topic for editing.

> **Note:** If you want to print the expanded content (for example to PDF), you should consider changing the **Styles** drop down to **Map With Resolved Topics - Print ready**.

Open Map in Editor

For complex operations which cannot be performed in the simplified DITA Maps view (like editing a relationship table) you can open the map in the main editing area.

> **Note:** You can also use this action to open referenced DITA Maps in the **Editor**.

Link with Editor

Disables/Enables the synchronization between the file path of the current editor and the selected topic reference in the **DITA Maps Manager** view.

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

Profiling/Conditional Text

menu with the following actions:

- **Show Profiling Attributes** - Enables/Disables displaying the values of the profiling attributes at the end of the titles of topic references. When enabled, the values of the profiling attributes are displayed both in the **DITA Maps Manager** view and in the **Author** editor.

- **Configure Profiling Condition Sets ...** - Opens the preferences page for adding and editing the profiling conditions that you can apply in the **DITA Maps Manager** view and the **Author** editor. When a profiling condition set is applied, the keys defined in the DITA map are gathered by filtering out the excluded content.

Root map

Specifies a master DITA Map that Oxygen XML Author uses to establish a key space which you can use with any other DITA Map that the master one imports.

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*.

The following edit actions can be performed on an opened DITA Map:

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 86 topic.

Edit Profiling Attributes

Allows you to change the *profiling attributes* defined on all selected elements.

Edit Properties

Edit the properties of a selected node. You can find more details about this action in the *Edit Properties* on page 231 topic.

Append Child and Insert After

Container sub-menus for a number of actions that create a map node as a child of the currently selected node or as a sibling of the currently selected node.

- **New topic** - Inserts a new topic.

- **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.

- **Reference to the currently edited file** - Inserts a reference to the currently edited file.

- A set of actions that allow you to insert various *topic reference specializations*.

- **Topic Heading** - Inserts a topic heading. You can find more details about this action in the *Inserting a Topic Heading* topic.

- **Topic Group** - Inserts a topic group. You can find more details about this action in the *Inserting a Topic Group* on page 231 topic.

Search References

Searches all references to the current topic in the entire ditamap.

Refactoring > Rename resource...

Allows you to *change the name of a resource linked in the edited DITA Map*.

Refactoring > Move resource...

Allows you to *change the location on disk of a resource linked in the edited DITA Map*.

Find/Replace in Files

Allows you to find and replace content across multiple files.

Check Spelling in Files

Allows you to *spell check multiple files*.

Cut, Copy, Paste

Actions that allow you to cut, copy and paste parts of the DITA map.

Paste Before

Pastes the content of the clipboard (only if it is a part of the DITA map) before the currently selected DITA map node.

Paste After

Pastes the content of the clipboard (only if it is a part of the DITA map) after the currently selected DITA map node.

✖ Delete

Deletes the selected node.

Organize > ⬆ Move Up (Alt+Up Arrow)

Moves the selected nodes in front of their respective previous siblings.

Organize > ⬇ Move Down (Alt+Down Arrow)

Moves the selected nodes after their next respective siblings.

Organize > ⬅ Promote (Alt+Left Arrow)

Moves the selected nodes after their respective parents as siblings.

Organize > ➡ Demote (Alt+Right Arrow)

Moves the selected nodes as children to their respective previous siblings.

> **Note:** As an alternative to these actions, you can select one or multiple topics, then drag and drop them to the desired position inside the map.

Expand All, Collapse All

Actions that allow you to expand and collapse respectively, the entire DITA Map structure.

To watch our video demonstrations about DITA editing and DITA Maps Manager view, go to *http://oxygenxml.com/demo/DITA_Editing.html* and *http://oxygenxml.com/demo/DITA_Maps_Manager.html* respectively.

Editing Actions

> **Important:** References can be made either by using the `href` attribute or by using the new `keyref` attribute to point to a key defined in the map. Oxygen XML Author tries to resolve both cases. `keyrefs` are solved relative to the current map.

In additional to being available on the toolbar and on the contextual menu, more navigation actions and all edit actions appear in the **DITA Maps** menu. The menu is only available when the view is active on screen.

In addition to the edit actions described above, the contextual menu contains the following actions:

Open

Opens in the editor the resources referred by the nodes you select.

Append Child/Insert After

Sub-menus containing the following actions:

Reference

Appends/Inserts a topic reference as a child/sibling of the selected node.

Reference to the current edited file

Appends/Inserts a topic reference to the current edited file as a child/sibling of the selected node.

New topic

Create a new topic from templates, saves it on disk and adds it into the DITA map.

Anchor Reference

, **Key Definition**, **Map Reference**, **Topic Reference**, **Topic Set**, **Topic Set Reference**Allows you to insert a reference to a topic file, a map file, a topic set, or a key definition.

Topic heading

Appends/Inserts a topic heading as a child/sibling of the selected node.

Topic group

Appends/Inserts a topic group as a child/sibling of the selected node.

Find/Replace in Files

Find and replace in files using the scope of the current edited map.

Check Spelling in Files

Checks the spelling of the files in the scope of the current edited map.

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`. A shortcut can be set for this action in *the Menu Shortcut Keys preferences*.

Cut, Copy, Paste, Undo, Redo

Common edit actions with the same functionality as those found in the text editor.

Paste Before, Paste After

Pastes the content of the clipboard before, respectively after, the selected node.

You can also arrange the nodes by dragging and dropping one or more nodes at a time. Drop operations can be performed before, after or as child of the targeted node. The relative location of the drop is indicated while hovering the mouse over a node before releasing the mouse button for the drop.

Drag and drop operations allow you to:

Copy

Select the nodes you want to copy and start dragging them. Before dropping them in the appropriate place, press and hold the **Ctrl** key (**Meta** key on Mac). The mouse pointer changes to indicate that a copy operation is performed.

Move

Select the nodes you want to move and drag and drop them in the appropriate place.

Promote (Alt+Left Arrow)/Demote (Alt+Right Arrow)

You can move nodes between child and parent nodes which ensures both **Promote** (**Alt+Left Arrow**) and **Demote** (**Alt+Right Arrow**) operations.

Apart from the above actions, the contextual menu of the root of the map contains the following two options:

Open Map in Editor

Opens the ditamap in the **Editor** view.

> **Note:** The bubble displayed on the **Editor** tab of a DITA Map opened in the **Editor** view is rendered blue.

Open Map in Editor with resolved topics

Opens the DITA Map in the editor view and displays the content referred by the links in the ditamap.

Export DITA Maps

All the resources referred in a DITA Map are located and exported in a directory. You can also choose to export the resources as an archive.

> **Tip:**
>
> You can open and edit linked topics easily by double clicking the references or by right-clicking and choosing **Open in editor**. If the referenced file does not exist you are allowed to create it.
>
> By right clicking the map root element you can open and edit it in the main editor area for more complex operations.
>
> You can decide to open the reference directly in the Author mode and keep this setting as a default.

Creating a Map

To create a DITA map, Subject scheme, bookmap, or other type of DITA map:

1. Go to **File** > **New**.
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. Save the map using the 💾 **Save** button on the toolbar of the **DITA Maps Manager** view.

Validating DITA Maps

To validate a DITA map, go to the *the DITA Maps Manager view* and click ☑ **Validate and Check for Completeness**. You can also find the ☑ **Validate and Check for Completeness** action in the **DITA Maps** menu. Invoking this action opens the **DITA Map completeness Check** dialog box, which allows you to configure the DITA Map validation.

The validation process of a DITA MAP covers the following steps:

- verifies whether the file paths of the topic references are valid. In case an `href` attribute points to an invalid file path it is reported as a separate error in the **Errors** view.
- validates each referred topic and map. Each topic file is opened and validated against the appropriate DITA DTD. In case another DITA map is referred in the main one, the referred DITA Map is verified recursively, applying the same algorithm as for the main map.

The following options are available in the **DITA Map Completeness Check** dialog box:

- **Batch validate referenced DITA resources** - this option decides the level of validation that applies to referred DITA files:

  - if the check box is left unchecked (which is the default setting), the DITA files will be validated using the rules defined in the DTD or XML Schema declared in the document.
  - if the check box is checked, the DITA files will be validated using rules defined in their associated *validation scenario*.

- **Check the existence of non-DITA references resources** - extends the validation of referred resources to non-DITA files. Enable the **Include remote resources** options if you want to check that remote referenced binary resources (like images, movie clips, ZIP archives) exist at the specified location.
- **Use DITAVAL filters** - the content of the map is filtered by applying a *profiling condition set* before validation:

  - **From the current condition set** - the map is filtered using the condition set applied currently in the DITA Maps Manager view.
  - **From all available condition sets** - for each available condition set, the map content is filtered using the condition set before validation.
  - **From the associated transformation scenario** - the filtering condition set is specified explicitly as a DITAVAL file in the current transformation scenario associated with the DITA map.
  - **Other DITAVAL files** - for each DITAVAL file from this list, the map content is filtered using the DITAVAL file before validation.

    📄 **Note:** A link invalid in the content that resulted from the filtering process is reported as an error.

- **Check for duplicate element IDs within a topic** - if an ID is duplicated after assembling all topics referred in the map, it is reported as error.
- **Report links to topics not referenced in DITA Maps** - checks that all referred topics are linked in the DITA map.
- **Identify possible conflicts in profile attribute values** - when a topic's profiling attributes contain values that are not found in parent topics profiling attributes, the content of the topic is overshadowed when generating profiled output. This option reports such possible conflicts.

- **Report attributes and values that conflict with profiling preferences** - looks for profiling attributes and values not defined in the *Profiling / Conditional Text* preferences page. It also checks if profiling attributes defined as *single-value* have multiple values set in the searched topics.
- **Additional schematron checks** - allows you to select a Schematron schema that Oxygen XML Author uses for the validation of DITA resources.

Using a Root Map

You can set a DITA Map as a key space for all the other DITA Maps and topics that you edit in Oxygen XML Author. This DITA Map is called a *root map*. Specifying the correct root map assures that you encounter no validation problems when you work with keyrefs. All the keys that are defined in a root map are available in the maps that the root map imports.

You can specify the root map from:

- *the DITA Maps Manager view*
- *the Insert Key Reference Dialog dialog*
- *the Insert Content Key Reference dialog*

You can also click one of the key reference errors to select the root map.

To watch our video demonstration about the DITA Root Map support, go to *http://oxygenxml.com/demo/DITA_Root_Map.html*.

Create a Topic in a Map

To add a topic to a DITA map:

1. Select a node of a map open in the **DITA Maps Manager View**.
2. To insert the topic as a child of the selected node, right click that node and choose **Insert** > **Append Child**. To insert the topic as a sibling to the current node, choose **Insert** > **Insert After**. Then select the type of reference you want to create.
   The **Insert Reference** dialog box is displayed.
3. Select the topic to insert and press the **Insert** button or the **Insert and close** button.
   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.
5. Close the dialog using the **Close** button.

Organize Topics in a Map

To understand how to organize topics in a *DITA map* using the **DITA Maps Manager**, you can examine the sample map called `flowers.ditamap`, located in the `[OXYGEN_DIR]/samples/dita` folder.

1. Open the file `flowers.ditamap`.
2. Select the gear icon in the top right corner of the **DITA Maps Manager** and select **Show extended toolbar**.
3. 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*.
4. 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.
5. Close the map without saving.

Creating Relationship Tables

You can define relationships between topics in a relationship table. A relationship table it created inside a *DITA map*.

1. Open the DITA map file in the XML editor and switch to *Author mode*.
2. Go to **DITA** > **Relationship Table** > 🖳 **Insert Relationship Table**.

Th **Insert Relationship Table** dialog is displayed.

3. Set the number of rows, the number of columns, a table title (optional), and select whether you want a table header. Click **Insert**.

4. Enter 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.

5. To insert a topic reference in a cell, place the cursor in a table cell and click 📄 *Insert Reference*.

6. To add a new row to the table or remove an existing row use 📇 **Insert Relationship Row**/📇 **Delete Relationship Row.**

7. To add a new column to the table or remove an existing column, use 📇 **Insert Relationship Column**/📇 **Delete Relationship Column**.

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 a `topicref`, its possible child nodes are:

- anchor reference (the `anchorref` element),
- topic reference (the `topicref` element),
- map reference (the `mapref` element),
- navigation reference (the `navref` element),
- topic set reference (the `topicsetref` element),
- topic set (the `topicset` element),
- key definition (the `keydef` element),
- topic head (the `topichead` element),
- topic group (the `topicgroup` element)

The same dialog box can be used to insert a non-DITA file like a PDF document.
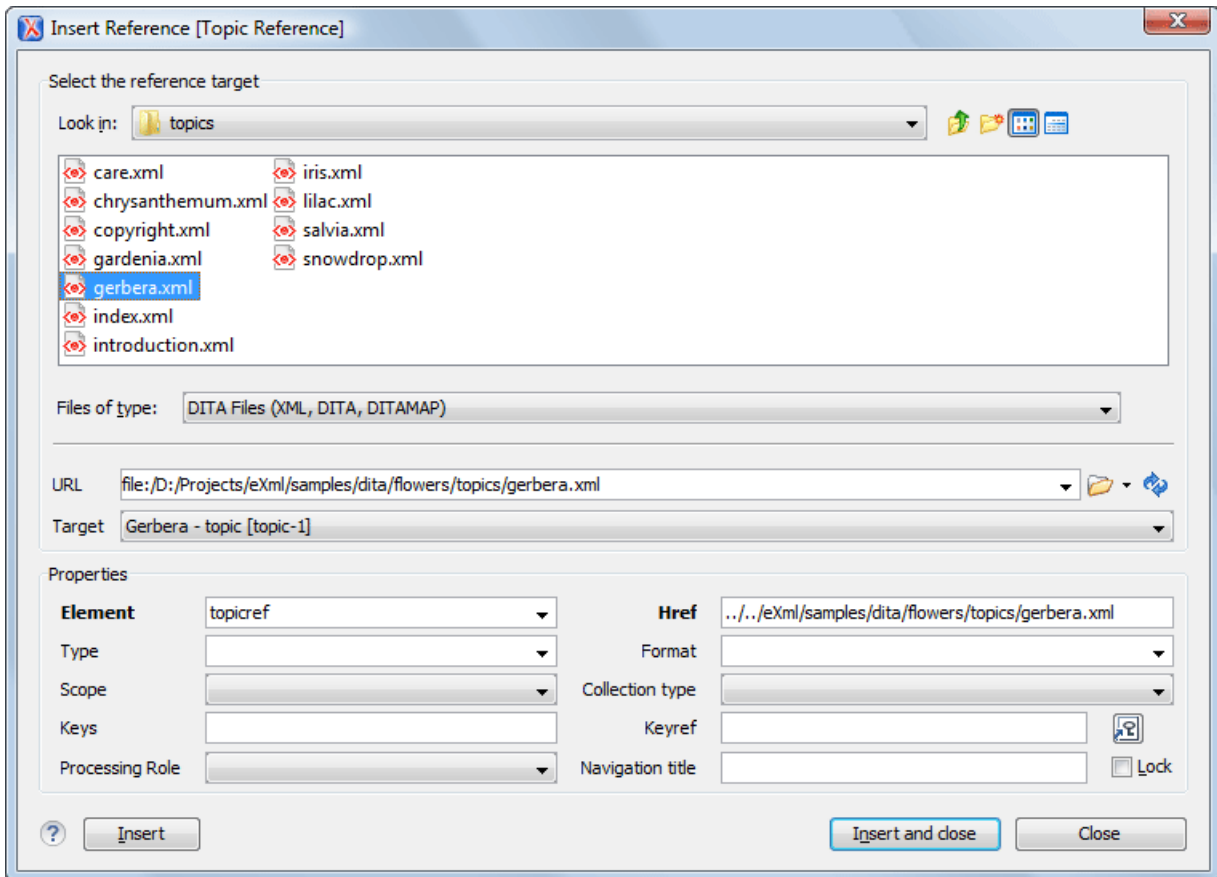
Figure 104: Insert Topic Reference Dialog

By using the **Insert Topic Reference** dialog you can easily browse for and select the source topic file. The **Target** combo box shows all available topics that can be targeted in the file. Selecting a target modifies the **Href** value to point to it which corresponds to the `href` attribute of the inserted `topicref` element. The **Format** and **Scope** combos are automatically filled based on the selected file and correspond to the `format` and `scope` attributes of the inserted `topicref` element. You can specify and enforce a custom navigation title by checking the **Navigation title** checkbox and entering the desired title.

The file chooser located in the dialog allows you to easily select the desired topic. The selected topic file will be added as a child or sibling of the current selected topic reference, depending on the insert action selected from the contextual menu of the **DITA Maps** view, that is an insert action from the **Append child** submenu or from the **Insert after** one. You can easily insert multiple topic references by keeping the dialog opened and changing the selection in the **DITA Maps Manager** tree. You can also select multiple resources in the file explorer and then insert them all as topic references.

Another easy way to insert a topic reference is to drag files from the **Project** view, file system explorer or **Data Source Explorer** view and drop them into the map tree.

You can also define keys using the **Keys** text field on the inserted `topicref` or `keydef` element. Instead of using the **Href** combo box to point to a location you can reference a key definition using the **Keyref** text field. Use the 🔧 **Choose key reference** to access the list of keys defined in the currently open DITA map.

The **Processing Role** combo box allows setting the `processing-role` attribute to one of the allowed values for DITA reference elements: `resource-only`, `normal`, `-dita-use-conref-target`.

Inserting a Topic Heading

The `topichead` element provides a title-only entry in a navigation map, as an alternative to the fully-linked title provided by the `topicref` element.

A topic heading can be inserted both from the toolbar action and the contextual node actions.
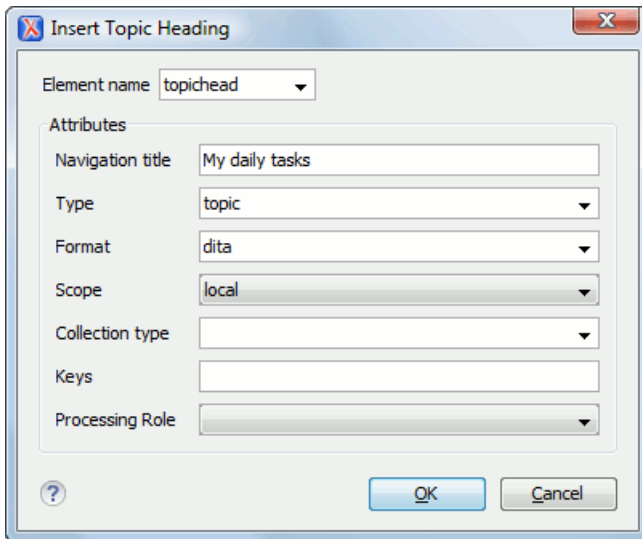
Figure 105: Insert Topic Heading Dialog

By using the **Insert Topic Heading** dialog you can easily insert a `topichead` element. The **Navigation title** is required but other attributes can be specified as well from the dialog.

Inserting a Topic Group

The `topicgroup` element identifies a group of topics (such as a concepts, tasks, or references) or other resources. A `topicgroup` can contain other `topicgroup` elements, allowing you to express navigation or table-of-contents hierarchies, as well as implying relationships between the containing `topicgroup` and its children. You can set the collection-type of a container `topicgroup` to determine how its children are related to each other. Relationships end up expressed as links in the output (with each participant in a relationship having links to the other participants by default).

A topic group may be inserted both from the toolbar action and the contextual node actions.
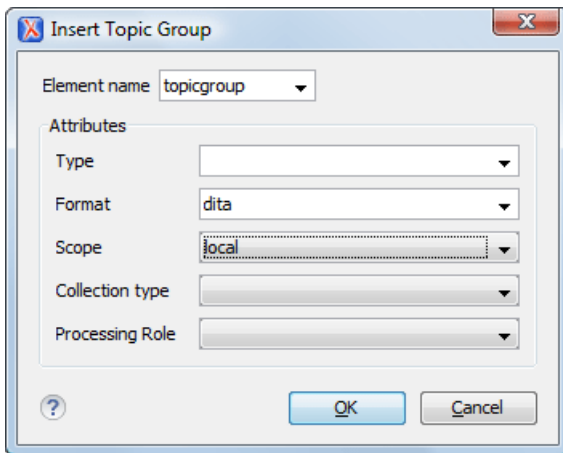
Figure 106: Insert Topic Group Dialog

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 Author uses the DITA Open Toolkit (DITA-OT) to transform DITA maps and topics into an output format. For this purpose both the DITA Open Toolkit 1.6.1 and ANT 1.7 come bundled in Oxygen XML Author.

More information about the DITA Open Toolkit are available at *http://dita-ot.sourceforge.net/*.

Creating a DITA Transformation Scenario

To create a **DITA OT Transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **DITA OT Transformation**;
- Click the **Configure Transformation Scenario(s)**(**Ctrl+Shift+T (Command+Shift+T on OS X)**) button on the **Transformation** toolbar, then click the **New** button and select **DITA OT Transformation**;
- Select **Ctrl+Shift+T (Command+Shift+T on OS X)** on your keyboard or click the **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **DITA OT Transformation**.

    Note: In case a scenario is already associated with the edited document, selecting **Ctrl+Shift+T (Command+Shift+T on OS X)** or **Apply Transformation Scenario** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **DITA transformation type** dialog box. This dialog presents the list of possible outputs that the **DITA OT Transformation** is able to produce. Select the transformation type, click **OK** and move on to configuring the options in the **New Scenario** dialog. This dialog allows you to configure the options that control the transformation.

All three methods open the **DITA transformation type** dialog box. This dialog presents the list of possible outputs that the **DITA OT Transformation** is able to produce:

- PDF
- WebHelp
- WebHelp - Mobile
- WebHelp with Feedback
- XHTML
- Electronic Publication (EPUB)
- *Compiled HTML Help (CHM)*
- JavaHelp
- Eclipse Help
- Eclipse Content
- *Kindle (DITA 4 Publishers)*
- HTML2 (DITA 4 Publishers)
- InDesign (DITA 4 Publishers)

- Graphical Map Visualizer (DITA 4 Publishers) - experimental
- *TocJS*
- Open Document Format
- DocBook
- RTF
- troff
- Legacy PDF
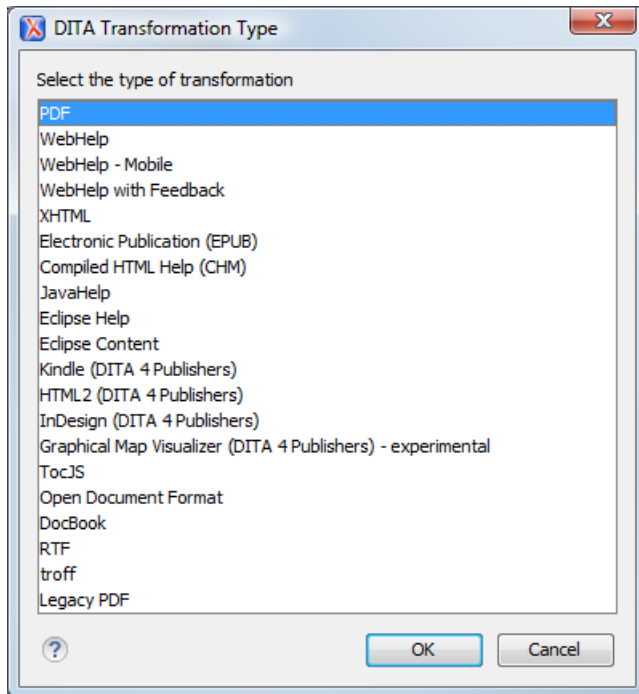
Figure 107: Select DITA Transformation type

Select the transformation type, click **OK** and move on to configuring the options in the **New Scenario** dialog. This dialog allows you to configure the options that control the transformation.

> **Note:** Depending on the type of output you choose, Oxygen XML Author generates values for the default ANT parameters, so you can execute the scenario right away without further customization.

The upper part of the dialog box contains the **Name** field and the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box allows you to *further customize the DITA OT Transformation scenario* and contains the following tabs:

- *Parameters*
- *Filters*
- *Advanced*
- *Output*
- *FO Processor*

Compiled HTML Help (CHM) Output Format

To perform a **Compiled HTML Help (CHM)** transformation Oxygen XML Author needs `HTML Help Workshop` to be installed on your computer. Oxygen XML Author automatically detects `HTML Help Workshop` and uses it. 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.

Changing the Output Encoding

Oxygen XML Author uses the `html.locale` parameter to correctly display specific characters of different languages in the output of the **Compiled HTML Help (CHM)** transformation. The **Compiled HTML Help (CHM)** default scenario that comes bundled with Oxygen XML Author has the `html.locale` parameter set to `en-US`. The format of the `html.locale` parameter is `LL-CC`, where `LL` represents the language code (`en` for example) and `CC` represents the country code (`US` for example). The language codes are contained in the `ISO 639-1` standard and the country codes are contained in the `ISO 3166-1` standard. For further details about language tags, go to *http://www.rfc-editor.org/rfc/rfc5646.txt*.

The default value of the `html.locale` is `en-US`. To customize this parameter, go to   **Configure Transformation Scenarios** and click the   **Edit** button. In the parameter tab search for the `html.locale` parameter and change its value to the desired language tag.

Viewing CHM on a Network Drive

When viewing a CHM on a network drive, if you only see the TOC and an empty page displaying "Navigation to the web page was canceled" note that this is normal behaviour. The Microsoft viewer for CHM does not display the topics for a CHM opened on a network drive.

As a workaround, copy the CHM file on your local system and view it there.

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 `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.sophos.tocjs/basefiles` folder in the transformation's output folder.
3. Copy the `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.sophos.tocjs/sample/basefiles/frameset.html` file in the transformation's output folder.
4. Edit `frameset.html` file.
5. Locate element `<frame name="contentwin" src="concepts/about.html">`.
6. Replace `"concepts/about.html"` with `"index.html"`.

Kindle Output Format

Enabling Oxygen XML Author to obtain Kindle output from DITA Maps, requires KindleGen. To download KindleGen, follow these steps:

- Go to *www.amazon.com/kindleformat/kindlegen* and download the zip file that matches your operating system.
- Unzip the file on your local disk.
- Start Oxygen XML Author and open a DITA Map in the *DITA Maps Manager View*.
- In the **DITA Maps Manager View** open the   **Configure Transformation Scenario(s)** dialog.
- Select the **DITA Map Kindle** transformation and press the **Edit** button to edit it. Go to **Parameters** tab and set the **kindlegen.executable** parameter as the path to the KindleGen directory.
- Accept the changes. Now the transformation is set-up and ready to be applied to a DITA map to obtain the Kindle output.

Migrating OOXML Documents to DITA

Oxygen XML Author integrates the entire DITA for Publishers plugins suite, enabling you to migrate content from Open Office XML documents to DITA:

- Open an OOXML document in Oxygen XML Author. The document is opened in the **Archive Browser** view;
- From the **Archive Browser**, open `document.xml`;

    📓   **Note:** `document.xml` holds the content of the document.

- Click 🔧 **Configure Transformation Scenario(s)** on the toolbar and apply the **DOCX DITA** scenario. In case you encounter any issues with the transformation, click the link below for further details about the Word to DITA Transformation Framework.

Customizing a DITA Scenario

This section explains how to customize the parameters of a DITA transformation scenario.

The Parameters Tab

This dialog allows you to configure the parameters sent to the DITA-OT build file.

All the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (for example: 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. Use the **Filter** text box to search for a specific term in the entire parameters collection.

Using the toolbar buttons you can add, edit or remove a parameter.

Depending on the type of a parameter, its value can be one of the following:

- a simple text field for simple parameter values
- a combo box with some predefined values
- a file chooser and an editor variables selector to simplify setting a file path as value to a parameter.

The value of a parameter can be entered at runtime if a value *ask('user-message', param-type, 'default-value' ?)* is used as value of parameter in the Configure parameters dialog.

Examples:

- `${ask('message')}` - Only the message displayed for the user is specified.
- `${ask('message', generic, 'default')}` - `'message'` is displayed, the type is not specified (the default is string), the default value is `'default'`.
- `${ask('message', password)}` - `'message'` is displayed, the characters typed are masked with a circle symbol.
- `${ask('message', password, 'default')}` - same as before, the default value is `'default'`.
- `${ask('message', url)}` - `'message'` is displayed, the parameter type is URL.
- `${ask('message', url, 'default')}` - same as before, the default value is `'default'`.

The *Filters* Tab

In the scenario **Filters** tab you can add filters to remove certain content elements from the generated output.

Figure 108: Edit Filters tab

There are three ways to define filters:

- **Use DITAVAL file** - If you already have a DITAVAL file associated with the transformed map you can specify the path to it and it will be used when filtering content. You can find out more about constructing a DITAVAL file in the *DITA OT Documentation* topic.
- **Use profiling condition set** - Sets the *profiling condition set* that applies to the document you transform.
- **Exclude from output all elements with any of the following attributes** - You can configure a simple list of attribute (name, value) pairs which when present on an element in the input will remove it from output.

The *Advanced* Tab

In the **Advanced** tab, you can specify advanced options for the transformation.

Figure 109: 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* parameter 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 path configured in the *Ant preferences page*. A custom value can also be set;
- **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 Author.
- **JVM Arguments** - This parameter allows you to set specific parameters to the Java Virtual Machine used by ANT. By default it is set to -Xmx384m which means the transformation process is allowed to use 384 megabytes of memory. Sometimes, when performing a large DITA map transformation you may want to increase the memory allocated to the Java Virtual Machine from the default value (384 MB) to a higher value, like 512 MB. This way, you can avoid the Out of Memory error messages (**OutOfMemoryError**) received from the ANT process.

  > **Note:** If you are publishing DITA to PDF and still experience problems, you should also increase the amount of memory allocated to the FO transformer. To do this, go to the **Advanced** tab and increase the value of the **Java Arguments** parameter.

- **Libraries** - Oxygen XML Author adds by default as high priority libraries which are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can specify all the additional libraries (jar files or additional class paths) which are used by the ANT transformer. You can also decide to control all libraries added to the classpath.

The Output Tab

In the **Output** tab, you can configure options related to the place where the output is generated.
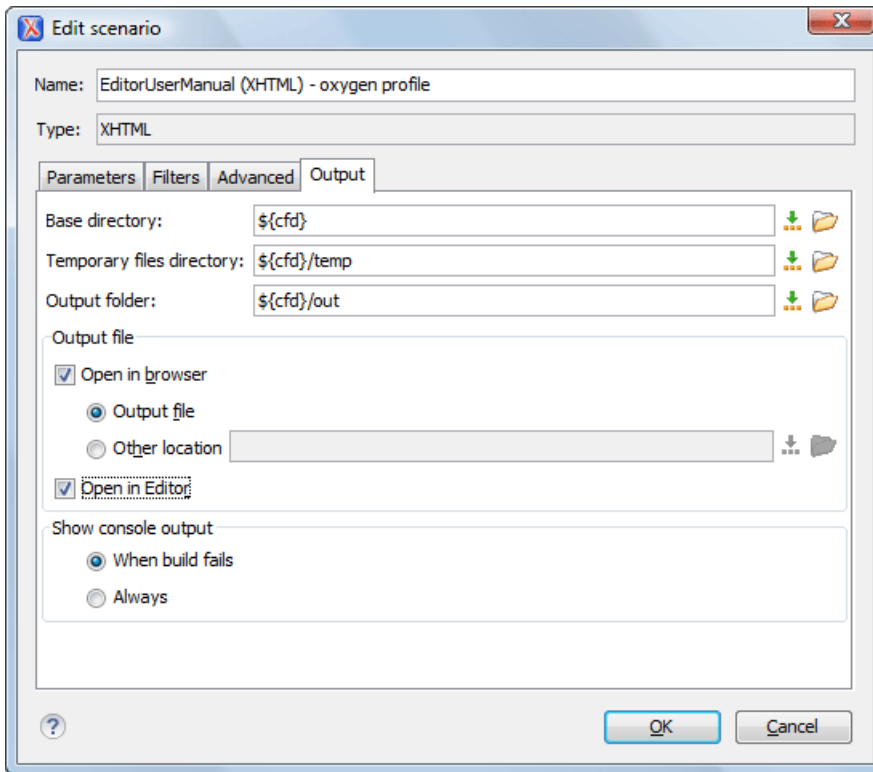
Figure 110: Output settings tab

You have several parameters that you can specify here:

- **Base directory** - All the relative paths which appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located.
- **Temporary files directory** - This directory is used to store pre-processed temporary files until the final output is obtained.
- **Output folder** - The folder where the final output content is copied.
- **Output file options** - The transformation output can then be opened in a browser or even in the editor, if specified.
- **Show console output** - Specifies whether the console is always displayed or only when the build fails.

> **Note:** If the DITA Map or topic is opened from a remote location or from a ZIP file, the scenario must specify absolute output, temporary and base file paths.

The *FO Processor* Tab

This tab allows you to set an FO Processor, when you choose to generate PDF output.

Figure 111: FO Processor configuration tab

You can choose between:

- **Apache FOP** - Default setting. This processor comes bundled with Oxygen XML Author.
- **XEP** - The *RenderX* XEP processor.

    If you select **XEP** in the combo and XEP was already installed in Oxygen XML Author you can see the detected installation path appear under the combo box.

    XEP is considered as installed if it was detected in one of the following sources:

    - XEP was configured as an external FO Processor in the *FO Processors option page*;
    - The system property *com.oxygenxml.xep.location* was set to point to the XEP executable file for the platform (for example: `xep.bat` on Windows);
    - XEP was installed in the `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/lib` directory of the Oxygen XML Author installation directory.

- **Antenna House** - The *Antenna House* AH (v5) or XSL (v4) Formatter processor.

    If Antenna House was already installed on your computer and you select **Antenna House** in the combo box, in Oxygen XML Author you can see the detected installation path appear under the combo.

    Antenna House is considered as installed if it was detected in one of the following sources:

    - Environment variable set by Antenna House installation (the newest installation version will be used, *v5* being preferred over *v4*).
    - Antenna House was added as an external FO Processor in the Oxygen XML Author preferences pages.

    To further customize the PDF output obtained from the Antenna House processor:

    - edit the transformation scenario
    - open the *Parameters tab*
    - add the `env.AXF_OPT` parameter and point to Antenna House configuration file

Running a DITA Map ANT Transformation

The transformation is run as an external ANT process so you can continue using the application as the transformation unfolds. All output from the process appears in the **DITA Transformation** tab.

> **Tip:** The HTTP proxy settings are also used for the ANT transformation so if the transformation fails because it cannot connect to an external location you can check the *the **Proxy** preferences page*

Set a Font for PDF Output Generated with Apache FOP

When a DITA map is transformed to PDF using the Apache FOP processor and it contains some Unicode characters that cannot be rendered by the default PDF fonts then a font that is capable to render these characters must be configured and embedded in the PDF result.

The settings that must be modified for configuring a font for the Apache FOP processor are detailed in *this section*.

How does the DITA Open Toolkit PDF font mapping work?

The DITA OT contains a file
[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml
which maps logical fonts used in the XSLT stylesheets to physical fonts which will be used by the FO processor to generate the PDF output.

The XSLT stylesheets used to generate the XSL-FO output contain code like:

```
<xsl:attribute name="font-family">monospace</xsl:attribute>
```

The font-family is defined to be *monospace*, but *monospace* is just an alias, it is not a physical font name. So another stage in the PDF generation takes this *monospace* alias and looks in the font-mappings.xml.

If it finds a mapping like this:

```
<aliases>
      <alias name="monospace">Monospaced</alias>
 </aliases>
```

then it looks to see if the *Monospaced* has a *logical-font* definition and if so it will use the *physical-font* specified there:

```
<logical-font name="Monospaced">
      <physical-font char-set="default">
        <font-face>Courier New, Courier</font-face>
      </physical-font>
............
</logical-font>
```

> **Important:**
>
> If no alias mapping is found for a font-family specified in the XSLT stylesheets, the processing defaults to **Helvetica**.

Tips and Tricks

This section contains solutions for common problems encountered when working with the DITA Open Toolkit.

Debugging PDF Transformations

To debug a DITA PDF transformation scenario using the XSLT Debugger follow these steps:

1. *Open the **Preferences** dialog* and go to **XML** > **XML Catalog**, click **Add** and select the file located at
   [OXYGEN_DIR]\frameworks\dita\DITA-OT\plugins\org.dita.pdf2\cfg\catalog.xml;
2. Open the map in the **DITA Maps Manager** and create a **DITA Map PDF** transformation scenario;
3. Edit the scenario, go to the **Parameters** tab and change the value of the **clean.temp** parameter to **no**;
4. Run the transformation scenario;

5. Open the **stage1.xml** file located in the temporary directory and format and indent it;

6. Create a transformation scenario for this XML file by associating the `topic2fo_shell_fop.xsl` stylesheet located at
`[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/topic2fo_shell_fop.xsl`.
If you are specifically using the RenderX XEP or Antenna House FO processors to build the PDF output, you should use the XSL stylesheets `topic2fo_shell_xep.xsl` or `topic2fo_shell_axf.xsl` located in the same folder.

7. In the transformation scenario edit the XSLT Processor combo box choose the Saxon EE XSLT processor (the same processor used when the DITA OT transformation is executed).

8. In the transformation scenario edit the **Parameters** list and set the parameter *locale* with the value *en_GB* and the parameter *customizationDir.url* to point either to your customization directory or to the default DITA OT customization directory. It's value should have an URL syntax
like:`file://c:/path/to/[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg`.

9. Debug the transformation scenario.

The PDF Processing Fails to Use the DITA OT and Apache FOP

There are cases when publishing DITA content which you consider perfectly valid fails when creating the PDF file. This topic tries to list some of the more usual situations and has fix-up hints for each of these cases.

• The FO processor cannot save the PDF at the specified target. The console output contains messages like:

```
[fop] [ERROR] Anttask - Error rendering fo file: C:\samples\dita\temp\pdf\oxygen_dita_temp\topic.fo <Failed
to open C:\samples\dita\out\pdf\test.pdf>
Failed to open samples\dita\out\pdf\test.pdf
[fop]  at org.apache.fop.tools.anttasks.FOPTaskStarter.renderInputHandler(Fop.java:647)
[fop]  at org.apache.fop.tools.anttasks.FOPTaskStarter.render(Fop.java:676)
............
[fop]  at org.apache.tools.ant.launch.Launcher.run(Launcher.java:280)
[fop]  at org.apache.tools.ant.launch.Launcher.main(Launcher.java:109)
[fop] Caused by: java.io.FileNotFoundException: C:\Users\radu_coravu\Desktop\bev\out\pdf\test.pdf
(The process cannot access the file because it is being used by another process)
[fop]  at java.io.FileOutputStream.open(Native Method)
[fop]  at java.io.FileOutputStream.<init>(Unknown Source)
[fop]  at java.io.FileOutputStream.<init>(Unknown Source)
[fop]  at org.apache.fop.tools.anttasks.FOPTaskStarter.renderInputHandler(Fop.java:644)
[fop]  ... 45 more
[fop] C:\samples\dita\temp\pdf\oxygen_dita_temp\topic.fo -> C:\samples\dita\out\pdf\test.pdf
```

> 🛈 **Tip:** Such an error message usually means that the PDF file is already opened in a PDF Reader application. The workaround is simple, always close the external PDF reader application before running the transformation.

• One of the DITA tables contains more cells in a table row than the defined number of *colspec* elements. The console output contains messages like:

```
[fop] [ERROR] Anttask - Error rendering fo file:
D:\projects\eXml\samples\dita\flowers\temp\pdf\oxygen_dita_temp\topic.fo
<net.sf.saxon.trans.XPathException: org.apache.fop.fo.ValidationException:
The column-number or number of cells in the row overflows the number of fo:table-columns specified for the
table. (See position 179:-1)>net.sf.saxon.trans.XPathException: org.apache.fop.fo.ValidationException: The
column-number or number of cells in the row overflows the number of fo:table-columns specified for the table.
 (See position 179:-1)
[fop]  at org.apache.fop.tools.anttasks.FOPTaskStarter.renderInputHandler(Fop.java:657)
[fop]  at net.sf.saxon.event.ContentHandlerProxy.startContent(ContentHandlerProxy.java:375)
............
[fop] D:\projects\samples\dita\flowers\temp\pdf\oxygen_dita_temp\topic.fo ->
D:\projects\samples\dita\flowers\out\pdf\flowers.pdf
```

> 🛈 **Tip:**
>
> • Find the DITA topic that contains a DITA CALS table for which one of the table rows has more cells than the number of *colspecs* defined on the table.
> • Edit the transformation scenario you and set the parameter *clean.temp* to *no*.
> • Run the transformation, open the `topic.fo` file in Oxygen XML Author, and look in it at the line specified in the error message (`See position 179:-1`).

- Look around that line in the `XSL-FO` file to find relevant text content which you can use, for example, with the **Find/Replace in Files** action in the **DITA Maps Manager** view to find the original DITA topic for which the table was generated.

- There is a broken link in the generated `XSL-FO` file. The PDF is generated but contains a link that is not working. The console output contains messages like:

```
[fop] 1248 WARN [ main ] org.apache.fop.apps.FOUserAgent - Page 6: Unresolved ID reference
"unique_4_Connect_42_wrongID" found.
```

  Tip:

  - Use the **Validate and Check for Completeness** action available in the **DITA Maps Manager** view to find such problems.
  - In case you publish to PDF using a certain `DITAVAL` filter, set the same DITAVAL file in the **DITA Map Completeness Check** dialog.
  - In case the **Validate and Check for Completeness** action does not discover any issues, edit the transformation scenario and set the *clean.temp* parameter to *no*.
  - Run the transformation, open the `topic.fo` file in Oxygen XML Author, and search in it for the `unique_4_Connect_42_wrongID` id.
  - Look around that line in the `XSL-FO` file to find relevant text content which you can use, for example, with the **Find/Replace in Files** action in the **DITA Maps Manager** view to find the original DITA topic for which the table was generated.

Topic References outside the main DITA Map folder

Referencing to a DITA topic, map or to a binary resource (for example: image) which is located outside of the folder where the main DITA Map is located usually leads to problems when publishing the content using the DITA Open Toolkit. The DITA OT does not handle well links to topics which are outside the directory where the published DITA Map is found. By default it does not even copy the referenced topics to the output directory.

You have the following options:

1. Create another DITA Map which is located in a folder path above all referenced folders and reference from it the original DITA Map. Then transform this DITA Map instead.
2. Edit the transformation scenario and in the **Parameters** tab edit the **fix.external.refs.com.oxygenxml** parameter. This parameter is used to specify whether the application tries to fix up such references in a temporary files folder before the DITA Open Toolkit is invoked on the fixed references. The fix up has no impact on your edited DITA content. Only "false" and "true" are valid values. The default value is false.

Adding Videos in the Output

Videos can be referred and played in all HTML5-based output formats (like *WebHelp*). For example, to add a YouTube video in the WebHelp output generated from a DITA topic, follow these steps:

- edit the DITA topic and refer the video using an `object` element like in the following example:

```
<object outputclass="video">
  <param name="src" value="http://www.youtube.com/watch/v/VideoName"/>
</object>
```

- execute a *WebHelp* or *WebHelp with Feedback* transformation scenario to obtain the output

Syntax Highlight Inside Codeblock Sections

`Codeblock` elements can be used in the DITA content to give examples from different programming or scripting languages like XML, CSS, Java and so on.

You can add to the XHTML-based and PDF published outputs syntax highlight inside these codeblock sections, by setting the *outputclass* attribute on the *codeblock* element to a specific value depending on the content type:

- `language-xml`
- `language-java`
- `language-css`
- `language-javascript`
- `language-sql`
- `language-c`
- `language-cpp`
- `language-csharp`
- `language-ini`
- `language-php`
- `language-python`
- `language-ruby`
- `language-perl`
- `language-bourne`

To customize different colors for the highlighted tokens you can edit the XSLT stylesheets:

- `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.highlight/pdfHighlight.xsl`
- `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.highlight/xhtmlHighlight.xsl`

DITA-OT Customization

Oxygen XML Author includes a bundled copy of the DITA-OT as an Oxygen XML Author *framework*. That framework includes a number of transformation scenarios for common output formats. This section explains how to customize specific parameters of a DITA transformation scenario like setting a custom DITA Open Toolkit, a custom build file or a separate installation of the Ant tool.

Support for Transformation Customizations

You can change all DITA transformation parameters to customize your needs. In addition, you can specify a custom build file, parameters to the JVM and many more for the transformation.

Using Your Custom Build File

You can specify a custom build file to be used in DITA-OT transformations by editing the transformation scenario that you are using. In the *Advanced* tab you should change the **Custom build file** path to point to the custom build file.

As an example, if you want to call a custom script before running the DITA OT, your custom build file would have the following content:

```
<project basedir="." default="dist">
<!--The DITA OT default build file-->
 <import file="build.xml"/>
 <target name="dist">
  <!-- You could run your script here -->
  <!--<exec></exec>-->
   <!--Call the DITA OT default target-->
   <antcall target="init"/>
 </target>
</project>
```

Customizing the Oxygen XML Author Ant Tool

The Ant 1.7 tool which comes with Oxygen XML Author is located in the `[OXYGEN_DIR]/tools/ant` directory. Any additional libraries for Ant must be copied to the Oxygen XML Author Ant `lib` directory.

If you are using Java 1.6 to run Oxygen XML Author the Ant tool should need no additional libraries to process JavaScript in build files.

Increasing the Memory for the Ant Process

For details about setting custom JVM arguments to the ANT process please see *this section*.

Resolving Topic References Through an XML Catalog

There are situations where you want to resolve URIs with an XML catalog:

- you customized your DITA map to refer topics using URI's instead of local paths
- you have URI content references in your DITA topic files and you want to map them to local files when the map is transformed

In such situations you have to *add the catalog to Oxygen XML Author*. The **DITA Maps Manager** view will solve the displayed topic refs through the added XML catalog and also the DITA map transformations (for PDF output, for XHTML output, etc) will solve the URI references through the added XML catalog.

DITA to PDF Output Customization

In this topic you will see how to do a basic customization of the PDF output by setting up a customization directory.

DITA Open Toolkit PDF output customizations can be made in two major ways:

1. Creating a DITA Open Toolkit plugin which adds extensions to the PDF plugin. More details can be found in the *DITA Open Toolkit user manual*.
2. Creating a customization directory and using it from the PDF transformation scenario. A small example of this procedure can be found below.

Let us take for example the common case of embedding a company logo image in the front matter of the book. You can later extend this example to create more complex customizations.

1. Copy the entire directory:
   `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/Customization` to some other place, for instance: C:\Customization.
2. Copy your logo image to: `C:\Customization\common\artwork\logo.png`.
3. Rename `C:\Customization\catalog.xml.orig` to: `C:\Customization\catalog.xml`
4. Open the `catalog.xml` in Oxygen XML Author and uncomment this line:

   ```
   <!--uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"/-->
   ```

   So now it looks like this:

   ```
   <uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"/>
   ```

5. Rename the file: `C:\Customization\fo\xsl\custom.xsl.orig` to: `C:\Customization\fo\xsl\custom.xsl`
6. Open the `custom.xsl` file in Oxygen XML Author and create the template called createFrontMatter_1.0. This will override the same template from the `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/front-matter.xsl`. Now, `custom.xsl` has the content:

   ```xml
   <?xml version='1.0'?>
   <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:fo="http://www.w3.org/1999/XSL/Format"
       version="1.1">

   <xsl:template name="createFrontMatter_1.0">
           <fo:page-sequence master-reference="front-matter" xsl:use-attribute-sets="__force__page__count">
               <xsl:call-template name="insertFrontMatterStaticContents"/>
               <fo:flow flow-name="xsl-region-body">
                   <fo:block xsl:use-attribute-sets="__frontmatter">
                       <!-- set the title -->
                       <fo:block xsl:use-attribute-sets="__frontmatter__title">
                           <xsl:choose>
                               <xsl:when test="$map/*[contains(@class,' topic/title ')][1]">
                                   <xsl:apply-templates select="$map/*[contains(@class,' topic/title ')][1]"/>
                               </xsl:when>
                               <xsl:when test="$map//*[contains(@class,' bookmap/mainbooktitle ')][1]">
   ```

```
                                        <xsl:apply-templates select="$map//*[contains(@class,' bookmap/mainbooktitle
')][1]"/>
                                    </xsl:when>
                                    <xsl:when test="//*[contains(@class, ' map/map ')]/@title">
                                        <xsl:value-of select="//*[contains(@class, ' map/map ')]/@title"/>
                                    </xsl:when>
                                    <xsl:otherwise>
                                        <xsl:value-of select="/descendant::*[contains(@class, ' topic/topic
')][1]/*[contains(@class, ' topic/title ')]"/>
                                    </xsl:otherwise>
                                </xsl:choose>
                            </fo:block>

                            <!-- set the subtitle -->
                            <xsl:apply-templates select="$map//*[contains(@class,' bookmap/booktitlealt ')]"/>

                            <fo:block xsl:use-attribute-sets="__frontmatter__owner">
                                <xsl:apply-templates select="$map//*[contains(@class,' bookmap/bookmeta ')]"/>
                            </fo:block>

                            <fo:block text-align="center" width="100%">
                                <fo:external-graphic src="url({concat($artworkPrefix,
'/Customization/OpenTopic/common/artwork/logo.png')})"/>
                            </fo:block>

                        </fo:block>

                        <!--<xsl:call-template name="createPreface"/>-->

                    </fo:flow>
                </fo:page-sequence>
            <xsl:call-template name="createNotices"/>
        </xsl:template>
</xsl:stylesheet>
```

7. Edit (or duplicate, then edit) the DITA Map to PDF transformation scenario. In the Parameters tab, set the customization.dir parameter to `C:\Customization`.

There are other ways in which you could directly modify the XSL stylesheets from the DITA OT but this customization gives you flexibility to future DITA OT upgrades in Oxygen XML Author.

Header and Footer Customization

The XSLT stylesheet
`[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/static-content.xsl`
contains templates which output the static header and footers for various parts of the PDF like the prolog, table of contents, front matter or body.

> The templates for generating a footer for pages in the body are called `insertBodyOddFooter` or
> `insertBodyEvenFooter`.

These templates get the static content from resource files which depend on the language used for generating the PDF. The default resource file is
`[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/common/vars/en.xml`.
These resource files contain variables like *Body odd footer* which can be set to specific user values.

Instead of modifying these resource files directly, they can be overwritten with modified versions of the resources in a PDF customization directory as explained in *DITA to PDF Output Customization* on page 244.

Customizing `<note>` Images in PDF

Here are some steps to customize the images which appear next to each type of note in the PDF output using a *PDF customization folder*:

1. Copy the
`[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/common/vars/en.xml`
file to the `[CUSTOMIZATION_DIR]\common\vars` folder.

2. Edit the copied en.xml file and modify, for example, the path to the image for <note> element with the type attribute set to notice from:

```
<variable id="notice Note Image Path">Configuration/OpenTopic/cfg/common/artwork/important.png</variable>
```

to:

```
<variable id="notice Note Image Path">Customization/OpenTopic/common/artwork/notice.gif</variable>
```

3. Add your custom **notice** image to Customization_DIR_NAME\common\artwork\notice.gif.
4. Edit the **DITA to PDF** transformation scenario and in the **Parameters** tab set the path for the customization.dir property to point to the customization folder.

Installing a plugin in the DITA Open Toolkit

The architecture of the DITA Open Toolkit allows additional plugins to be installed.

1. The additional plugin(s) should be copied to the plugins directory from the used DITA Open Toolkit installation (by default [OXYGEN_DIR]\frameworks\dita\DITA-OT\plugins.
2. The DITA OT ANT integrator build file needs to be run. In the **Transformation Scenarios** view there is a predefined transformation scenario called **Run DITA OT Integrator** which can be used for this.

> ⚠ **Important:** The folder where the DITA OT is located needs to have full write access permissions set to it.

3. If the plugin contributed with a new **transtype** to the publishing stage, the application will not detect it by default. You have to create a new **DITA OT** transformation scenario with a predefined type which is close to the newly added **transtype**, then edit the transformation scenario and in the **Parameters** tab add a parameter called **transtype** with the value of the newly added transformation type.

Creating a Simple DITA OT HTML and PDF Customization Plugin

This example describes a **DITA Open Toolkit** plugin which you can use to provide syntax highlight when publishing DITA **codeblock** elements to **PDF** or **HTML**-based outputs. The plugin is available in the DITA Open Toolkit distribution which comes with the application.

> 📄 **Note:** Starting with version 15.2, this plugin is bundled by default with the application:*Syntax Highlight Inside Codeblock Sections* on page 242.

Here are some steps to help anyone wanting to create an **XSLT** customization plugin for the **DITA Open Toolkit** for HTML and PDF based outputs.

1. Create a folder for your plugin in the DITA OT **plugins** folder. The **DITA OT** bundled with Oxygen can be found here:

   [OXYGEN_DIR]/frameworks/dita/DITA-OT

   In my case I created the following folder:

   [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.highlight

2. Create a **plugin.xml** file in that folder containing the extension points of the plugin. In my case, the plugin descriptor file contains:

```
<plugin id="com.oxygenxml.highlight">
  <feature extension="package.support.name" value="Oxygen XML Editor Support"/>
  <feature extension="package.support.email" value="support@oxygenxml.com"/>
  <feature extension="package.version" value="1.0.0"/>
  <feature extension="dita.xsl.xhtml" value="xhtmlHighlight.xsl" type="file"/>
  <feature extension="dita.xsl.xslfo" value="pdfHighlight.xsl" type="file"/>
</plugin>
```

The important extensions in it are the references to the XSLT stylesheets which will be used to style the HTML and the PDF outputs.

You can find a bunch of other DITA OT plugin extension points here:

3. Create an XSLT stylesheet called **xhtmlHighlight.xsl** located in the same plugin folder.

As I want to overwrite the creation of the HTML content from a DITA **codeblock** element I will first need to find the XSLT template that I need to overwrite. A DITA **codeblock** element has the **class** attribute value "+ **topic/pre pr-d/codeblock** ". Usually in such cases I take part of the class attribute value and search using the "**Find/Replace in Files**" Oxygen action in all of the DITA OT XSLT resources.

In this case I searched for **topic/pre** and found this XSLT stylesheet:

`[OXYGEN_DIR]/frameworks/dita/DITA-OT/xsl/xslhtml/dita2htmlImpl.xsl`

containing this XSLT template:

```
<xsl:template match="*[contains(@class,' topic/pre ')]" name="topic.pre">
  <xsl:apply-templates select="."  mode="pre-fmt" />
</xsl:template>
```

thus my **xhtmlHighlight.xsl** will overwrite the content of the template like:

```
<xsl:template match="*[contains(@class,' topic/pre ')]" name="topic.pre">
    <!-- This template is deprecated in DITA-OT 1.7. Processing will moved into the main element rule. -->
    <xsl:if test="contains(@frame,'top')"><hr /></xsl:if>
    <xsl:apply-templates select="*[contains(@class,' ditaot-d/ditaval-startprop ')]" mode="out-of-line"/>
    <xsl:call-template name="spec-title-nospace"/>
    <pre>
      <xsl:attribute name="class"><xsl:value-of select="name()"/></xsl:attribute>
      <xsl:call-template name="commonattributes"/>
      <xsl:call-template name="setscale"/>
      <xsl:call-template name="setidaname"/>
     <!--Here I'm calling the styler of the content inside the codeblock.-->
      <xsl:call-template name="outputStyling"/>
    </pre>
    <xsl:apply-templates select="*[contains(@class,' ditaot-d/ditaval-endprop ')]" mode="out-of-line"/>
    <xsl:if test="contains(@frame,'bot')"><hr /></xsl:if><xsl:value-of select="$newline"/>
  </xsl:template>
```

and call another XSLT template which applies as a Java extension the XSLTHL library to style the content.

4. Create an XSLT stylesheet called **pdfHighlight.xsl** located in the same plugin folder which will contain the PDF XSLT customization. In this case I will overwrite the XSLT template from:

`[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/legacypdf/xslfo/dita2fo-elems.xsl`

which has the content:

```
<xsl:template match="*[contains(@class,' topic/pre ')]">
  <xsl:call-template name="gen-att-label"/>
  <fo:block xsl:use-attribute-sets="pre">
    <!-- setclass -->
    <!-- set id -->
    <xsl:call-template name="setscale"/>
    <xsl:call-template name="setframe"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

5. To install your plugin in the DITA OT, run the integrator. In the application's **Transformation Scenarios** view, enable the **Show all scenarios** action, available in the drop down settings button. Just check that and execute the transformation scenario called **Run DITA OT Integrator**.

And that's it, your XSLT content will be applied with priority when publishing both to XHTML-based (WebHelp, CHM, EPUB, JavaHelp, Eclipse Help) and to PDF-based outputs.

Now, let's take a look at what the step (5) - running the integrator to install the plugin - really did:

1. In the XSLT stylesheet:

`[OXYGEN_DIR]/frameworks/dita/DITA-OT/xsl/dita2html-base.xsl`

a new import automatically appeared:

```
<xsl:import href="../plugins/com.oxygenxml.highlight/xhtmlHighlight.xsl"/>
```

This import is placed after all base imports and because of this it has a higher priority. More about imported template precedence can be found in the XSLT specs:

http://www.w3.org/TR/xslt#import

2. Likewise, in the top-level stylesheets related to PDF publishing like:

`[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/xsl/fo/topic2fo_shell_fop.xsl`

a new import statement has appeared:

```
<xsl:import href="../../../com.oxygenxml.highlight/pdfHighlight.xsl"/>
```

Now, you can distribute your plugin folder to anyone having a DITA OT installation along with some simple installation notes. Your customization will work as long as the templates you are overwriting have not changed from one DITA OT distribution to the other.

DITA Specialization Support

This section explains how you can integrate and edit a DITA specialization in Oxygen XML Author.

Integration of a DITA Specialization

A DITA specialization usually includes:

- DTD definitions for new elements as extensions of existing DITA elements
- optionally specialized processing, that is new XSLT template rules that match the extension part of the `class` attribute values of the new elements and thus extend the default processing available in DITA Open Toolkit

A specialization can be integrated in the application with minimum effort:

1. If the DITA specialization is available as a DITA Open Toolkit plugin, copy the plugin to the location of the DITA OT you are using (by default `[OXYGEN_DIR]\frameworks\dita\DITA-OT\plugins`). Then run the DITA OT integrator to integrate the plugin. In the ***Transformation Scenarios view*** there is a predefined scenario called **Run DITA OT Integrator** which can be used for this.

⚠ **Important:** The folder where the DITA OT is located needs to have full write access permissions set to it.

2. If the specialization is not available as a plugin, you have the following options:

- If the DTD's that define the extension elements are located in a folder outside the DITA Open Toolkit folder, 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/XML Schema 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 Author

Oxygen XML Author comes bundled with a DITA Open Toolkit, located in the `[OXYGEN_DIR]/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* parameter to point to the new directory.
2. To make changes in the libraries that come with the new DITA Open Toolkit and are used by the ANT process, go to the **Advanced** tab, click the **Libraries** button and uncheck **Allow Oxygen to add high priority libraries to classpath**.
3. If there are also changes in the DTD's and you want to use the new versions for content completion and validation, go to the Oxygen XML Author preferences in the **Document Type Association** page, edit the **DITA** and **DITA Map** document types and modify the catalog entry in the **Catalogs** tab to point to the custom catalog file `catalog-dita.xml`.

Reusing Content

The DITA framework allows reusing content from other DITA files with a content reference in the following ways:

*   You can select content in a topic, create a reusable component from it and reference the component in other locations using the actions **Create Reusable Component** and **Insert Reusable Component**. A reusable component is a file, usually shorter than a topic. You also have the option of replacing the selection with the component that you are in the process of creating. The created reusable component file is usually self-contained and it's automatically generated content can be fine tuned by modifying the resources located in the folder `[OXYGEN_DIR]\frameworks\dita\reuse`.
*   You can add, edit, and remove a content reference (`conref`) attribute to/from an existing element. The actions **Add/Edit Content Reference** and **Remove Content Reference** are available on the contextual menu of the Author editor and on the DITA menu. When a content reference is added or an existing content reference is edited, you can select any topic ID or interval of topic IDs (set also the `conrefend` field in the dialog for adding/editing the content reference) from a target DITA topic file.
*   You can insert an element with a content reference (`conref` or `conkeyref`) attribute using one of the actions **Insert Content Reference** and **Insert Content Key Reference** that are available on the DITA menu, the Author custom actions toolbar and the contextual menu of the Author editor.

DITA makes the distinction between local content, that is the text and graphics that are actually present in the element, and referenced content that is referred by the element but is located in a different file. To display referenced content, *Open the **Preferences** dialog* and go to **Editor** > **Edit modes** > **Author** > **Display referred content**.

Working with Content References

The `conref` DITA feature (short for *content reference*) lets you include a piece of source content as reference in other contexts. When you need to update that content, you do it in only one place. Typical uses of content references are for product names, warnings, definitions or process steps.

You can use either or both of the following strategies for managing content references:

- *Reusable components* - With this strategy, you create a new file for each piece of content that you want to reuse.
- *Arbitrary content references* - You may prefer to keep many pieces of reusable content in one file. For example, you might want one file to consist of a list of product names, with each product name in a `phrase` (`<ph>` element) within the file. Then, wherever you need to display a product name, you can insert a content reference that points to the appropriate `<ph>` element in this file.

    Note: A reference displays tracked changes and also comments of the source fragment. To edit these comments or accept/reject the changes, right click them and select **Edit Reference**.

    This strategy requires more setup than reusable components, but makes easier centrally managing the reused content.

Oxygen XML Author creates a reference to the external content by adding a `conref` attribute to an element in the local document. The `conref` attribute defines a link to the referenced content, made up of a path to the file and the topic ID within the file. The path may also reference a specific element ID within the topic. Referenced content is not physically copied to the referencing file, but Oxygen XML Author displays it as if it is there in the referencing file. You can also choose to view local content instead of referenced content, to edit the attributes or contents of the referencing element.

    Note: To search for references made through a direct content reference of a topic, paragraph, list item and so on, use the **Search References** action from the contextual menu.

How to Work with Reusable Components

When you need to reuse a part of a DITA topic in different places (in the same topic or in different topics) it is recommended to create a separate component and insert only a reference to the new component in all places. Below are the steps for extracting a reusable component, inserting a reference to the component and quickly editing the content inside the component.

1. Select with the mouse the content that you want to reuse in the DITA file opened in **Author** mode.
2. Start the action **Create Reusable Component** that is available on the DITA menu, the Author framework actions toolbar and the contextual menu of the Author editor.
3. In the combo box **Reuse Content** select the DITA element with the content that you want to extract in a separate component. The combo box contains the current DITA element where the cursor is located (for example a `p` element - a paragraph - or a `step` or a `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 can use the same content in multiple topics by inserting a DITA content reference to that content. The following steps describe the procedure of inserting a DITA content reference:

1. Position your caret inside the element that you want to refer and in the *Attributes view* enter a value in the **ID** field.

    In case you want to reuse just a part of the content of an element, select the content with your cursor, press **Enter** and in the proposals list select ph. This encapsulates your content inside a phrase(<ph>) element, allowing you to set an ID and then refer it.

2. Open the topic where you want to insert the reference to this element.

3. Click ▣ **Insert a DITA Content Reference** on the main toolbar.
    The **Insert Content Reference** dialog is displayed.

4. In the **Insert Content Reference** dialog, from the **URL** field, navigate to the topic that holds the element you want to refer.
    In the **Target ID** section of the **Insert Content Reference** dialog Oxygen XML Author presents the elements that you can refer.

5. Click the ID of the element you want to refer, then click **OK**.

    In case you select an interval of elements, the **Conrefend** field is filled with the id value of the element that ends the selected interval.

    A reference to the selected element is inserted at the caret position.

    > **Note:** To search for references made through a direct content reference of a topic, paragraph, list item and so on, use the **Search References** action from the contextual menu.

The Insert Content Reference Dialog

The **Insert Content Reference** dialog lets you reuse content by inserting references to the DITA elements that hold the content you want to reuse.

> **Note:** To refer the content inside a DITA element you first have to set an ID for that element.

The DITA conref attribute provides a mechanism for reuse of content fragments. The conref attribute stores a reference to another element and is processed to replace the referencing element with the referenced element. The element containing the content reference acts as a placeholder for the referenced element. The identifier for the referenced element must be either absolute or resolvable in the context of the referencing element. For more details, go to *http://docs.oasis-open.org/dita/v1.0/archspec/conref.html*.

Oxygen XML Author *displays the referred content* of a DITA conref if it can resolve it to a valid resource. If you have URI's instead of local paths in the XML documents and your DITA OT transformation needs an XML catalog to map the URI's to local paths you have to *add the catalog to Oxygen XML Author*. If the URI's can be resolved, the referred content is displayed in the **Author** mode and in the transformation output.

To open the **Insert Content Reference** dialog, do one of the following:

* go to **DITA** > ▣ **Insert a DITA Content Reference**
* click the ▣ **Insert a DITA Content Reference** action on the main toolbar
* in the contextual menu of the editing area, go to **Reuse** > ▣ **Insert a DITA Content Reference**
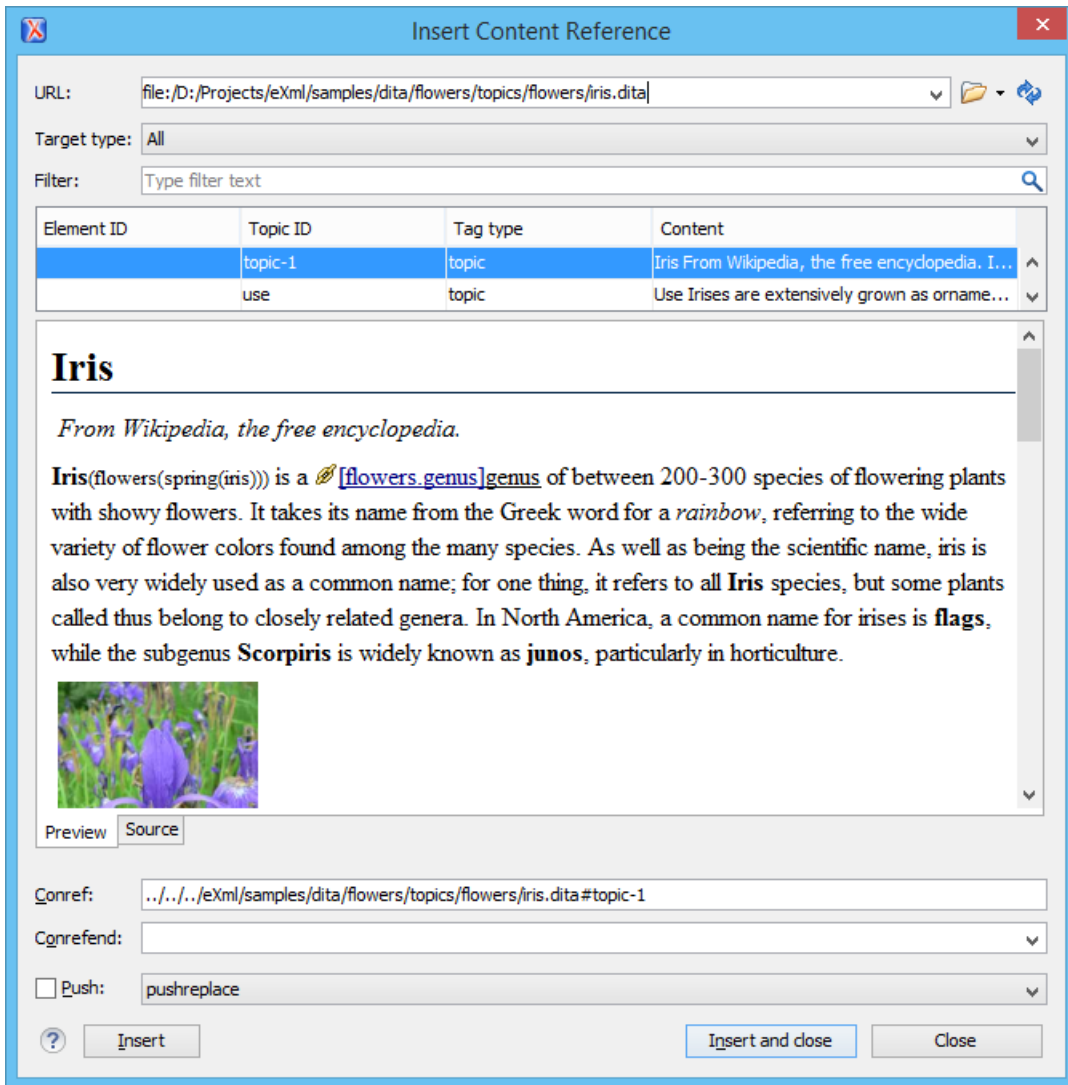
Figure 112: The Insert Content Reference Dialog

> **Note:** The **Insert Content Reference** dialog is not modal. The dialog is closed automatically in case you switch to a different editor.

The following fields are available in this dialog:

- **URL** - specifies the path to the topic that holds the content you want to refer
- **Target type** - specifies the type of the element to which you are targeting your `conref`
- **Target ID** - presents all the element IDs defined in the source topic
- **Preview** - displays a preview of the content in the element that you select in the **Target ID** list
- **Source** - displays the source of the element your want to refer
- **Conref** - displays the value of the attribute
- **Conrefend** - in case you select an interval of elements, this field displays the end value of the `conref` attribute
- **Push** - this option enables you to push content into DITA topics and maps, provided that the topics and maps contain elements with `id` attributes that identify the places where the content is to be pushed.

Moving and Renaming Resources

You can move or rename resources on disk directly from Oxygen XML Author. To do this, use one of the following actions available in the contextual menu of the **DITA Maps Manager** view:

Rename resource

This action allows you to change the name of a resource linked in the edited DITA Map, using the **Rename resource** dialog. This dialog contains the following options:

- **Update references** - enable this checkbox to update all references of the file in the edited DITA Map and in the files referred from the DITA Map. This way, the completeness of the DITA Map is preserved;
- **Preview** - select this button to display a preview of the changes Oxygen XML Author is about to make;
- **Rename** - executes the **Rename resource** operation;
- **Cancel** - cancels the **Rename resource** operation. No changes are applied.

Move resource

This action allows you to change the location of a resource linked in the edited DITA Map, using the **Move resource** dialog. This dialog contains the following options:

- **Destination** - specifies the target location on disk of the edited resource;
- **File name** - allows you to change the name of the edited resource;
- **Update references** - enable this checkbox to update all references of the file in the edited DITA Map and in the files referred from the DITA Map. This way, the completeness of the DITA Map is preserved;
- **Preview** - select this button to display a preview of the changes Oxygen XML Author is about to make;
- **Move** - moves the edited resource in the target location on disk;
- **Cancel** - cancels the **Move resource** operation. No changes are applied.

> **Note:** If a root DITA Map is not defined, the move and rename actions are executed in the context of the current DITA Map.

DITA Profiling / Conditional Text

Conditional text is a way to mark blocks of text meant to appear in some renditions of the document, but not in others. It differs from one variant of the document to another, while unconditional text appear in all document versions.

For instance you can mark a section of a document to be included in the manual designated for the *expert* users, other for the *novice* users manual while unmarked sections are included in any rendition.

You can use conditional text when you develop documentation for:

- a series of similar products
- different releases of a product
- various audiences

The benefits of using conditional text include reduced effort for updating and translating your content and an easy way to customize the output for various audiences.

Oxygen XML Author 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 Author you can define values for the DITA profiling attributes. The profiling configuration can be shared between content authors through the project file. There is no need for coding or editing configuration files.

Several profiling attributes can be aggregated into a profiling condition set that allow you to apply more complex filters on the document content. A Profiling Condition Set is a very powerful and convenient tool used to preview the content

that goes into the published output. For example, an installation manual available both in Windows and Linux variants can be profiled to highlight only the Linux procedures for more advanced users.

To watch our video demonstration about DITA profiling, go to *http://oxygenxml.com/demo/DITA_Profiling.html*.

Profiling / Conditional Text Markers

If **Show Profiling Attributes** option (available in the *Profiling / Conditional Text toolbar menu*) is set all profiling attributes set on the current element are listed at the end of the highlighted block. Profiled text is marked in the **Author** mode with a light green border.
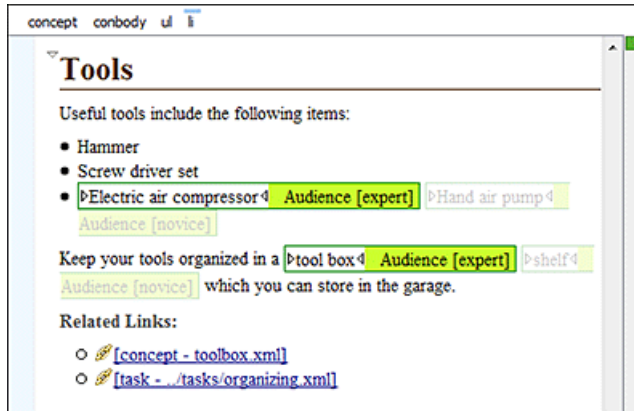
Figure 113: Profiling in Author

In the *DITA Maps Manager View* view different decorators are user to mark profiled and non-profiled topics:

* ■ - the topic contains profiling attributes;
* □ - the topic inherits profiling attribute from its ancestors;
* ◪ - the topic contains and inherits profiling attributes;
* ‐ (dash) - the topic neither contains, nor inherits profiling attributes.
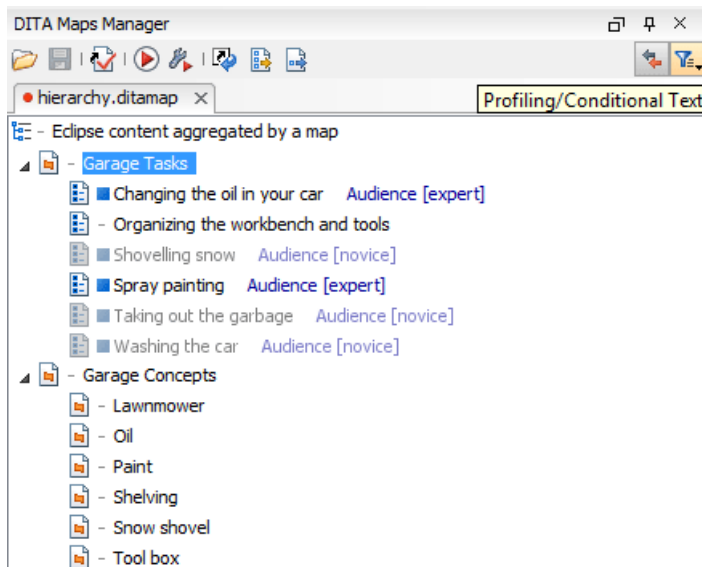
Figure 114: Profiling in DITA Maps Manager

The profiled content that does not match the rules imposed by the current condition sets is grayed-out, meaning that it will not be included in the published output.

Profiling with a Subject Scheme Map

A subject scheme map allows you to create custom profiling values and to manage the profiling attribute values used in the DITA topics without having to write a DITA specialization.

Subject scheme maps use key definitions to define a collection of profiling values instead of a collection of topics. The map that uses the set of profiling values must reference at its highest level the subject scheme map in which the profiling values are defined, for example:

```
<topicref href="test.ditamap" format="ditamap" type="subjectScheme"/>
```

A profiled value is a short and readable keyword that identifies a metadata attribute. For example, the `audience` metadata attribute may take a value that identifies the user group associated with a particular content unit. Typical user values for a medical-equipment product line might include `therapist`, `oncologist`, `physicist`, `radiologist`, `surgeon` and so on. A subject scheme map can define a list of these audience values.

Following is the sample content of the `test.ditamap` subject scheme:

```
<subjectScheme>
  <!-- Pull in a scheme that defines audience user values -->
  <subjectdef keys="users">
    <subjectdef keys="therapist"/>
    <subjectdef keys="oncologist"/>
    <subjectdef keys="physicist"/>
    <subjectdef keys="radiologist"/>
    <subjectdef keys="surgeon">
      <subjectdef keys="neuro-surgeon">
        <subjectdef keys="plastic-surgeon"/>
      </subjectdef>
    </subjectdef>
  </subjectdef>
  <!-- Define an enumeration of the audience attribute, equal to
       each value in the users subject. This makes the following values
       valid for the audience attribute: therapist, oncologist, physicist, radiologist -->
  <enumerationdef>
    <attributedef name="audience"/>
    <subjectdef keyref="users"/>
  </enumerationdef>
</subjectScheme>
```

When you edit a DITA topic in the **Text** mode or in the **Author** mode, Oxygen XML Author collects all the profiling values from the Subject Scheme Map that is referenced in the map currently opened in the *DITA Maps Manager*. The values of profiling attribute defined in a Subject Scheme Map are available in the **Edit Profiling Attribute** dialog regardless of their mapping the **Conditional Text** preferences page.

> In our example the values `therapist`, `oncologist` up to `plastic-surgeon` are displayed in *the content completion window* as values for the `audience` attribute.

Now let us consider we have the following fragment in a topic:

```
<p audience="neuro-surgeon">Some text.. </p>
```

When you define a DITAVAL filter you can exclude for instance anything which is `surgeon`:

```
<val>
    <prop action="exclude" att="audience" val="surgeon"/>
</val>
```

Then if you transform the main DITA Map specifying the DITAVAL filter file in the transformation scenario the `p` element should be excluded from the output. So excluding the `surgeon` audience excludes also the `neuro-surgeon` and `plastic-surgeon` from the output. More details about how hierarchical filtering and Subject Scheme Maps should work are found in the specification:*http://docs.oasis-open.org/dita/v1.2/os/spec/langref/subjectScheme.html#subjectScheme http://docs.oasis-open.org/dita/v1.2/os/spec/langref/subjectScheme.html?23subjectScheme*

Publish Profiled Text

Oxygen XML Author comes with preconfigured transformation scenarios for DITA. All these scenarios take into account the current profiling condition set.

| Name: | DITA Map PDF |
| Type: | PDF |

FO Processor | Parameters | **Filters** | Advanced | Output

○ Use DITAVAL file:

◉ Use current profiling condition set : "For experts"

○ Exclude from output all elements with any of the following attributes:
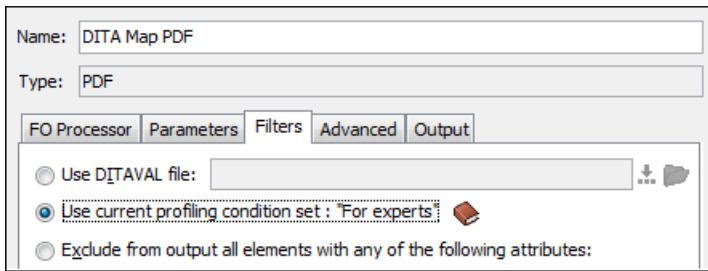
How to Profile DITA Content

1. *Open the Preferences dialog* and go to **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`, `otherprops` and `rev`. 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 **(Command+Click on OS X)** a topic reference in **DITA Maps Manager** and choose **Edit Profiling Attributes** from the contextual menu.

   b) In the **Author** editing mode, right-click **(Command+Click on OS X)** an XML element and choose **Edit Profiling Attributes** from the contextual menu.

   c) Use the **Attributes** view to set profiling attributes.

   Turn on the **Show Profiling Attributes** option to display the profiling markup in the **Author** editing mode.

Working with MathML

You can add MathML equations in a DITA document using one of the following methods:

- embed MathML directly into a DITA topic. You can start with the **Framework templates / DITA / topic / Composite with MathML** document template, available in the **New** file action wizard.

- reference an external MathML file as an image, using the **Insert Image Reference** toolbar action.

Note that MathML equations contained in DITA topics can only be published out-of-the-box in PDF using the **DITA PDF** transformation scenario. For other publishing formats users must employ additional customizations for handling MathML content.

MathML Equations in the HTML Output

MathJax is a solution to properly view MathML equations embedded in **HTML** content in a variety of browsers.

Without the help of the MathJax JavaScript code, right now only **Firefox** can render **MathML** equations embedded in the **HTML** code.

Let's say you have DocBook or DITA content which has embedded **MathML** equations and you want to properly view the equations in the published HTML output types (WebHelp, CHM, EPUB and so on).

You need to add a reference to the MathJax script in the **head** of all HTML files which have the equation embedded in them:

```
<script type="text/javascript" src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
  </script>
```

For DITA you can edit the **DITA Map WebHelp** transformation scenario and set the `args.hdf` parameter to point to the `footer.html` resource. Then transform to WebHelp and the equation should be properly rendered in the browsers like IE, Chrome and Firefox.

Chapter

8

Predefined Document Types

Topics:

The following document types come bundled with Oxygen XML Author. For each document type there are presented built-in transformation scenarios, document templates and Author extension actions.

Document Type

A *document type* or *framework* is associated to an XML file according to a set of rules. It includes also many settings that improve editing in the **Author** mode for the category of XML files it applies for. These settings include:

- a default grammar used for validation and content completion in both **Author** mode and Text mode
- CSS stylesheet(s) for rendering XML documents in **Author** mode
- user actions invoked from toolbar or menu in **Author** mode
- predefined scenarios used for transformation of the class of XML documents defined by the document type
- XML catalogs
- directories with file templates
- user-defined extensions for customizing the interaction with the content author in **Author** mode

Oxygen XML Author comes with built-in support for many common document types. Each document type is defined in a framework. You can make create new frameworks or make changes to existing frameworks to suit your individual requirements.

To see a video on configuring a framework in Oxygen XML Author, go to
http://oxygenxml.com/demo/FrameworkConfiguration.html.

The DocBook 4 Document Type

DocBook is a very popular set of tags for describing books, articles, and other prose documents, particularly technical documentation.

A file is considered to be a *DocBook 4* document when one of the following conditions are true:

- root element name is `book` or `article`;
- the PUBLIC ID of the document contains the string `-//OASIS//DTD DocBook XML`.

The schema of *DocBook 4* documents is `[OXYGEN_DIR]/frameworks/docbook/dtd/docbookx.dtd`.

The CSS file used for rendering DocBook content is located in `[OXYGEN_DIR]/frameworks/docbook/css/docbook.css`.

The XML catalog is stored in `[OXYGEN_DIR]/frameworks/docbook/catalog.xml`.

To watch our video demonstration about editing DocBook documents, go to
http://oxygenxml.com/demo/DocBook_Editing_in_Author.html.

DocBook 4 Author Extensions

Specific actions for DocBook documents are:

B **Bold emphasized text**

Emphasizes the selected text by surrounding it with `<emphasis role="bold"/>` tag;

I **Italic emphasized text**

Emphasizes the selected text by surrounding it with `<emphasis role="italic"/>` tag;

U̲ **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. When editing DITA documents, this action is available in the contextual menu of the editing area (under the **About Element** sub-menu) and in the documentation tip displayed by the **Content Completion Assistant**.

Cross reference (link)

Inserts a hypertext link;

Cross reference (xref)

Inserts a cross reference to another part of the document;

> **Note:** These actions are grouped under the **Link** toolbar actions group.

Web Link (ulink)

Inserts a link that addresses its target with an URL (Universal Resource Locator);

Insert olink

Inserts a link that addresses its target indirectly, using the `targetdoc` and `targetptr` values which are present in a *Targetset* file;
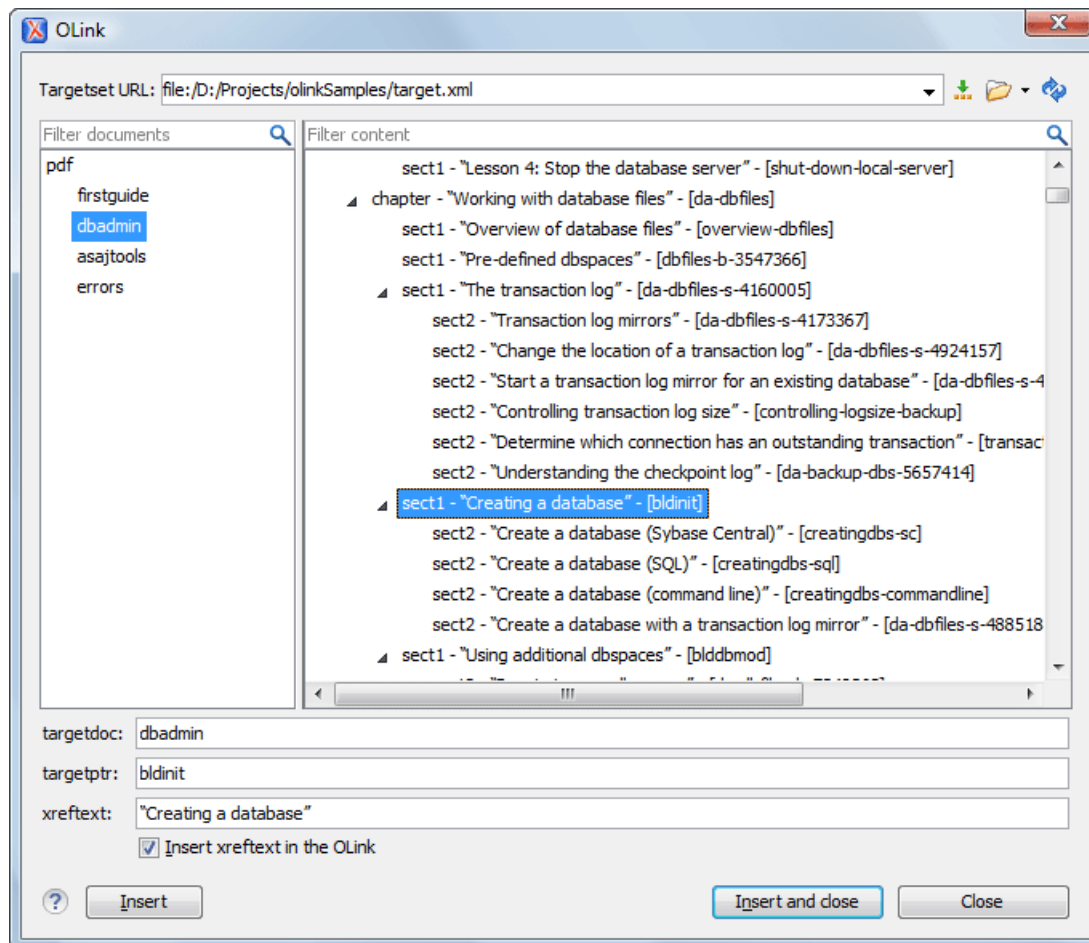
Figure 115: Insert OLink Dialog

After you choose the **Targetset** URL, the structure of the target documents is presented. For each target document (`targetdoc`), the content is displayed allowing for easy identification of the `targetptr` for the `olink` element which will be inserted. You can use the search fields to quickly identify a target. If you already know the values for the `targetdoc` and `targetptr`, you can insert them directly in the corresponding fields. You also have the possibility to edit an `olink` using the action **Edit OLink** available on the contextual menu. The last used **Targetset** URL will be used to identify the edited target.

Insert URI

Inserts an URI element. The URI identifies a Uniform Resource Identifier (URI) in content;

§ **Insert Section**

Inserts a new section / subsection in the document, depending on the current context. For example if the current context is `sect1` then a `sect2` is inserted, and so on;

← **Promote Section**

Inserts the current node as a brother of the parent node;

→ **Demote Section**

Inserts the current node a child of the previous node;

Insert image reference

Inserts a graphic object at the caret position. This is done by inserting either `<figure>` or `<inlinegraphic>` element depending on the current context. The following graphical formats are supported: GIF, JPG, JPEG, BMP, PNG, SVG;

Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also inserted automatically by default.

Insert an unordered list at the caret position

Inserts an itemized list. A child list item is also inserted automatically by default.

Insert a step or list Item

Inserts a new list item in any of the above list types.

Insert a variable list at the caret position

Inserts a DocBook variable list. A child list item is also inserted automatically by default;

Insert a procedure

Inserts a DocBook `procedure` element. A step `child` item is also inserted automatically;

Insert Table

Opens a dialog that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

Insert Row

Inserts a new table row with empty cells. This action is available when the caret is positioned inside a table.

Insert Column

Inserts a new table column with empty cells after the current column. This action is available when the caret is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, Oxygen XML Author a new cell at caret position. If the caret is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at caret position.

Delete Row

Deletes the table row located at caret position.

Insert Row Above

Inserts a row above the current one.

Insert Row Below

Inserts a row below the current one.

Insert Column Before

Inserts a column before the current one.

Insert Column After

Inserts a column after the current one.

Join Row Cells

Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current caret position with the content of the cell above it. This action works only if both cells have the same column span.

Join Cell Below

Joins the content of the cell from the current caret position with the content of the cell below it. This action works only if both cells have the same column span.

> **Note:** When you use **Join Cell Above** and **Join Cell Below**, Oxygen XML Author deletes the source row is case it remains empty. The cells that span over multiple rows are also updated.

Split Cell To The Left

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current caret position in two cells, inserting a new empty table cell above. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current caret position in two, inserting a new empty table cell below. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

> **Caution:** Column specifications are required for table actions to work properly.

Generate IDs

This action generates and sets unique IDs for:

- the element at caret position
- all top-level elements found in the current selection. Additionally, if the selection contains elements from the **Docbook** > **ID Options** list, they will all receive an unique ID

> **Note:** IDs already set are preserved.

The action is available both in the contextual menu and in the **Docbook** main menu.

ID Options

> Action available in the **Docbook** main menu, allows you to specify the elements for which Oxygen XML Author generates an unique ID if the **Auto generate IDs for elements** option is enabled. The configurable ID value pattern can accept most of the application supported *editor variables*.
>
> To keep an already set element ID when copying content in the same document, make sure the **Remove IDs 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 (Command+Shift+G on OS X)** and can be changed in the **DITA Topic** document type.

All actions described above are available in the contextual menu, the **DocBook4** submenu of the main menu or in the **Author custom actions** toolbar.

Dragging a file from *the Project view* or from *the DITA Maps Manager view* and dropping it into a DocBook 4 document that is edited in Author mode creates a link to the dragged file (the `ulink` DocBook element) at the drop location. Dragging an image file from the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example) and dropping it into a DocBook 4 document inserts an image element (the `inlinegraphic` DocBook element with the `fileref` attribute) with the location of the dragged file at the drop location (similar with the **Insert Graphic** toolbar action).

DocBook 4 Transformation Scenarios

Default transformation scenarios allow you to convert DocBook 4 to DocBook 5 documents and transform DocBook documents to HTML, HTML Chunk, PDF, XHTML, XHTML Chunk, WebHelp, EPUB and EPUB 3.

WebHelp Output Format

WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser.

Oxygen XML Author allows you to publish DocBook 4 documents into a WebHelp format which provides both table of contents and advanced search capabilities.

The layout is composed of two frames:

- the left frame, containing separate tabs for **Content**, **Search**, and **Index**;

    > **Note:** In case your documents contain no `indexterm` elements, the **Index** tab is not generated.

    > **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

- the right frame where help pages are displayed.

You can navigate through the content of your output using the arrows in the upper right part of the page. These arrows allow you to move to the parent, previous, and next topic. The parents of the currently opened topic are also presented at the top of the page.

You can use this button, displayed in the **Content** tab, to collapse all the topics presented the table of contents.

The top right corner of the page contains the following options:

- **With frames** - displays the output using HTML frames to render two separate sections: a section that presents the table o contents in the left side and a section that presents the content of a topic in the right side;

Figure 116: WebHelp Output

To publish DocBook 4 to WebHelp, use the **DocBook WebHelp** transformation. To further customize the out-of-the-box transformation, you can edit some of its parameters:

- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. Default setting is `false`;
- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `webhelp.copyright` - this parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output);
- `webhelp.indexer.language` - this parameter is used to identify the correct stemmer, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on;
- `xml.file` - this parameter specifies the path to the DocBook XML file;
- `webhelp.logo.image` - specifies a path to an image displayed as a logo in the left side of the output's header;
- `webhelp.logo.image.target.url` - specifies a target URL set on the logo image. When you click the logo image, you will be redirected to this address.

The **Search** tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on:

- number of keywords found in a single page. The higher the number, the better;

- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.

Rules applied during search:

- the space character separates keywords. An expression like *grow flowers* counts as two separate keywords: *grow* and *flowers*;
- do not use quotes to perform exact search for multiple-word expressions. An expression like *"grow flowers"*, returns no results in our case, because it searches for two separate words: *"grow* and *flowers"* (note the quote signs attached to each word);
- `indexterm` and `keywords` DITA elements are an effective way to increase the ranking of a page. For example, content inside *keywords* elements weighs twice as much as content inside a *H1* HTML element;
- words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters, count as a single word;
- search for words containing three or more characters. Shorter words, like *to*, or *of* are ignored. This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

> **Note:** This output format is compatible with the following browsers:
>
> - Internet Explorer 8 or newer;
> - Chrome;
> - Firefox;
> - Safari;
> - Opera.

WebHelp with Feedback Output Format

This section presents the Feedback-Enabled WebHelp systems support.

Oxygen XML Author has the ability to transform DocBook documents into feedback-enabled WebHelp systems. WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser. The Feedback system allows you to view discussion threads in a tree-like representation, reply to already posted comments and use stylized comments.

Oxygen XML Author allows you to publish DocBook 4 documents into a WebHelp with Feedback format which provides both Table of Contents and advanced search capabilities.

The layout is composed of two frames:

- the left frame, containing separate tabs for **Content**, **Search**, and **Index**;

> **Note:** In case your documents contain no `indexterm` elements, the **Index** tab is not generated.

> **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

- the right frame where help pages are displayed.

You can navigate through the content of your output using the arrows in the upper right part of the page. These arrows allow you to move to the parent, previous, and next topic. The parents of the currently opened topic are also presented at the top of the page.

To publish DocBook 4 to WebHelp with Feedback, use the **DocBook WebHelp with Feedback** transformation scenario. To further customize the out-of-the-box transformation, you can edit some of its parameters such as:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. Default setting is `false`;

- `webhelp.copyright` - this parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output);
- `webhelp.indexer.language` - this parameter is used to identify the correct stemmer, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on;
- `webhelp.product.id` - this parameter specifies a short name for the documentation target (product), for example `mobile-phone-user-guide`, `hvac-installation-guide`. You can deploy documentation for multiple products on the same server.

For further information about all the DocBook transformation parameters, go to *http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html*.

The **Search** tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on:

- number of keywords found in a single page. The higher the number, the better;
- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.

Rules applied during search:

- the space character separates keywords. An expression like *grow flowers* counts as two separate keywords: *grow* and *flowers*;
- do not use quotes to perform exact search for multiple-word expressions. An expression like *"grow flowers"*, returns no results in our case, because it searches for two separate words: *"grow* and *flowers"* (note the quote signs attached to each word);
- `indexterm` and `keywords` DITA elements are an effective way to increase the ranking of a page. For example, content inside *keywords* elements weighs twice as much as content inside a *H1* HTML element;
- words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters, count as a single word;
- search for words containing three or more characters. Shorter words, like *to*, or *of* are ignored. This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

To watch our video demonstration about the feedback-enabled WebHelp system, go to *http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html*.

> **Note:** This output format is compatible with the following browsers:
>
> - Internet Explorer 8 or newer;
> - Chrome;
> - Firefox;
> - Safari;
> - Opera.

> **Important:** Due to some security restrictions in Google Chrome, WebHelp pages loaded from the local system (through URLs of the `file:///...` format) may not work properly. We recommend you to load WebHelp pages in Google Chrome only from a web server.

> **Note:** In case you need to automate the transformation process and use it outside of Oxygen XML Author, you can use *the Oxygen XML WebHelp* transformation.

Introduction

Oxygen XML Author has the ability to transform DocBook 4 documents into feedback-enabled WebHelp systems.

WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser. It also provides table of contents and advanced search capabilities. The feedback system allows you to view discussion threads in a tree-like representation, post comments, reply to already posted comments, use stylized comments, and define administrators and moderators.

The DocBook 4 WebHelp with Feedback transformation

To publish DocBook 4 documents to WebHelp with Feedback, use the **DocBook WebHelp with Feedback** transformation. You can customize the out-of-the-box transformation by editing some of its parameters:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. Default setting is `false`;
- `webhelp.copyright` - this parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output);
- `webhelp.indexer.language` - this parameter is used to identify the correct stemmer, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on;
- `webhelp.product.id` - this parameter specifies a short name for the documentation target (product), for example `mobile-phone-user-guide`, `hvac-installation-guide`. You can deploy documentation for multiple products on the same server.

Before the transformation starts, enter the documentation product ID and the documentation version. After you run a **DocBook WebHelp with Feedback** transformation, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions and deployment of the output.

To watch our video demonstration about the feedback-enabled WebHelp system, go to *http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html*.

Installation

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Author requires the following system components:

- Apache Web Server running
- MySQL server running
- PHP Version 5.1.6 or later
- PHP MySQL Support
- Oxygen XML WebHelp system supports the following browsers:IE7+, Chrome 19+, Firefox 11+, Safari 5+, Opera 11+

Installation Instructions

📄 **Note:** These instructions were written for XAMPP 1.7.7 with PHP 5.3.8 and for *phpMyAdmin* 3.4.5. Later versions of these packages may change the location or name of some options, however the following installation steps should remain valid and basically the same.

In case you have a web server configured with PHP, MySQL, you can deploy the WebHelp output directly. Otherwise, install XAMPP. XAMPP is a free and open source cross-platform web server solution stack package. It consists mainly of the Apache HTTP Server, MySQL database, and interpreters for scripts written in PHP.

Deploying the WebHelp output

To deploy the WebHelp output, follow these steps:

1. Locate the directory of the HTML documents. Open `http://localhost/xampp/phpinfo.php` in your browser and see the value of the DOCUMENT_ROOT variable. In case you installed XAMPP in `C:\xampp`, the value of DOCUMENT_ROOT is `C:/xampp/htdocs`

2. Copy the transformation output folder in the `DOCUMENT_ROOT`
3. Rename it to a relevant name, for example, `webhelp_1`
4. Open `http://localhost/webhelp_1/`. You are redirected to
   `http://localhost/webhelp_1/oxygen-webhelp/install/`

   - Verify that the prerequisites are met
   - Press **Start Installation**
   - Configure the **Deployment Settings** section. Default values are provided, but you should adjust them as needed
   - Configure the **MySql Database Connection Settings** section. Use the details from the Create the WebHelp Feedback database section to fill-in the appropriate text boxes

     > ⚠️ **Warning:** Checking the **Create new database structure** option will overwrite any existing data in the selected database, if it already exists.

   - If the **Create new database structure** option is checked, the **Create WebHelp Administrator Account** section becomes available. Here you can set the administrator account data. The administrator is able to moderate new posts and manage WebHelp users.

     The same database can be used to store comments for different WebHelp deployments. If a topic is available in more than one WebHelp deployments and there are comments associated with it, you can choose to display the comments in all deployments that share the database. To do this, enable the **Display comments from other products** option. In the **Display comments from** section a list with the deployments sharing the same database is displayed. Select the deployments allowed to share common feedback.

     > 📝 **Note:** You can restrict the displayed comments of a product depending on its version. In case you have two products that use the same database and you restrict one of them to display comments starting from a certain version, the comments of the other product are also displayed from the specified version onwards.

   - Press **Next Step**
   - Remove the installation folder from your web server
   - Click the link pointing to the index of the documentation, or visit: `http://localhost/webhelp_1/`

To test your system, create a user and post a comment. Check if the notification emails are delivered to your inbox.

> 📝 **Note:** To read debug messages generated by the system:
>
> 1. Enable *JScript* logging:
>   - Either: open the `log.js` file, locate the `var log= new Log(Level.NONE);` line, and change the logging level to: `Level.INFO`, `Level.DEBUG`, `Level.WARN`, or `Level.ERROR`
>   - or append `?log=true` to the WebHelp URL
>
>   - Inspect the PHP and Apache server log files.

Layout of the Feedback-Enabled WebHelp System

The layout of the feedback-enabled WebHelp system resembles the layout of the basic WebHelp, the left frame remaining the same. However, the bottom of the right frame contains a **comments** bar. Select **Log in** from this bar to authenticate as a user of the WebHelp system. In case you do not have a user name, complete the fields in the dialog box that opens to create a user. Under the **comments** bar, you can click the **Add New Comment** button to add a comment whether you are logged in or not. The tabs in the left frame have the same functionality as the Content, Search, and Index tab of the basic WebHelp.

> 📝 **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

Figure 117: The layout of the Feedback-Enabled WebHelp System

After you log in, your name and user name are displayed in the **Comments** bar together with the **Log of** and **Edit** buttons. Click the **Edit** button to open the **User Profile** dialog. In this dialog you can customize the following options:

- **Your Name** - you can use this field to edit the initial name that you used to create your user profile;
- **Your e-mail address** - you can use this field to edit the initial e-mail address that you used to create your profile;
- When to receive an e-mail:

    - when a comment is left on a page that you commented on;
    - when a comment is left on any topic in the Help system ;
    - when a reply is left to one of my comments.

- **New Password** - allows you to enter a new password for your user account.

    > **Note:** The **Current Password** field from the top of the **User Profile** is mandatory in case you want to save the changes you make.

Advanced Customization and Management

Apart from the options available for a regular user, you can also use the administrative page for advanced customization and management. As an administrator, you have full access to all the features of the feedback-enabled WebHelp system. To access the administrative page, select **Admin Panel** from the **Comments** bar.

Figure 118: The Administrative Page

This page allows you to view all posts, export comments and set the version of the WebHelp system. You can also view the details of each user and search through these details using the **Search User Information** filter.

The upper part of the page contains the following actions:

* **Delete Orphan Comments** - deletes comments associated with topics that are no longer available
* **Delete Pending Users** - deletes all unconfirmed users that registered more than a week ago
* **View All Posts** - allows you to view all posts associated with a product and version
* **Export Comments** - allows you to export in XML format all posts associated with a product and version
* **Set Version** - use this action to display comments starting from a particular version

To edit the details of a user, click the corresponding row. Use the **Edit User** dialog to customize all the information associated with an user:

* **Name** - The user's full name
* **Level** - Use this field to modify the privilege level of the currently edited user. You can choose from:

  * **User** - regular user, able to post comments and receive e-mail notifications
  * **Moderator** - in addition to the regular **User** rights, this type of user has access to the **Admin Panel**. In the administrative page a moderator can view, delete, export comments and set the version of the feedback-enabled WebHelp system.
  * **Admin** - full administrative privileges. Can manage WebHelp-specific settings, users and their comments.

* **Company** - User's organization name
* **E-mail** - User's contact e-mail address. This is also the address where the WebHelp system sends notifications:

  * **WebHelp Notification** - when enabled, the user receives notifications when comments are posted anywhere in the feedback-enabled WebHelp system
  * **Reply Notification** - when enabled, the user receives notifications when comments are posted as a reply to one of his or hers comments
  * **Page Notification** - when enabled, the user receives notifications when comments are posted on a topic where he or she posted a comment

* **Date** - User registration date
* **Status** - Specifies the status of the currently edited user:

  * **Created** - the user is created but does not have any rights over the feedback-enabled WebHelp system
  * **Validated** - the user is able to use the feedback-enabled WebHelp system
  * **Suspended** - the user has no rights over the feedback-enabled WebHelp system

WebHelp Mobile Output Format

To further improve its ability to create online documentation, Oxygen XML Author offers support to transform DocBook documents into mobile WebHelp systems. This feature generates an output that works on multiple platforms (Android, iOS, BlackBerry, Windows Mobile) and is specially designed for mobile devices. All the specific touch screen gestures

are supported. The functionality of the desktop WebHelp layout is preserved, offering table of contents, search capabilities, and index navigation, organized in an intuitive layout.

Figure 119: Mobile WebHelp

To generate a mobile WebHelp system from your DocBook 4 document, go to the **DITA Maps Manager** view, click

 Configure Transformation Scenarios() and select the **DocBook WebHelp - Mobile** transformation scenario from the **DocBook 5** section. Click **Apply associated**. Once Oxygen XML Author finishes the transformation process, the output is opened in your default browser automatically.

Adding Videos in the Output

Videos can be referred and played in all HTML5-based output formats (like *WebHelp*). For example, to add a YouTube video in the WebHelp output generated from a DocBook document, follow these steps:

• edit the DocBook document and refer the video using an `mediaobject` element like in the following example:

```
<mediaobject>
  <videoobject>
    <videodata fileref="http://www.youtube.com/watch/v/VideoName"/>
  </videoobject>
</mediaobject>
```

• execute a *WebHelp* or *WebHelp with Feedback* transformation scenario to obtain the output

DocBook 4 Templates

Default templates are available in the *New File* wizard. You and can use them to create a skeletal form of a DocBook 4 book or article. These templates are stored in the `[OXYGEN_DIR]/frameworks/docbook/templates/DocBook 4` folder.

Here are some of the DocBook 4 templates available when creating *new documents from templates*.

• **Article**;
• **Article with MathML**;
• **Article with SVG**;
• **Article with XInclude**;
• **Book**;
• **Book with XInclude**;
• **Chapter**;
• **Section**;

- **Set of Books**.

Inserting olink Links in DocBook 5 Documents

An `olink` is a type of link between two DocBook XML documents.

The `olink` element is the equivalent for linking outside the current DocBook document. It has the attribute `targetdoc` for the document ID that contains the target element and the attribute `targetptr` for the ID (the value of an `id` or `xml:id` attribute) of the target element. The combination of those two attributes provides a unique identifier to locate cross references.

For example, the *Administrator Guide* is a book with the document ID `MailAdminGuide` and it contains a chapter about user accounts like the following:

```
<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 [
<!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:

```
<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 XML Author provides the **Insert OLink** action on the toolbar. This action allows selecting the target of an `olink` from the list of all possible targets from a specified target database document. In the following image the target database document is called `target.xml`.

Figure 120: 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 `[OXYGEN_DIR]/frameworks/docbook/5.0/rng/docbookxi.rng`, where *${frameworks}* is a subdirectory of the Oxygen XML Author install directory.

The CSS file used for rendering DocBook content is located in `[OXYGEN_DIR]/frameworks/docbook/css/docbook.css`.

The XML catalog is stored in `[OXYGEN_DIR]/frameworks/docbook/5.0/catalog.xml`.

To watch our video demonstration about editing DocBook documents, go to *http://oxygenxml.com/demo/DocBook_Editing_in_Author.html*.

DocBook 5 Author Extensions

The DocBook 5 extensions are the same as the *DocBook 4 extensions*.

A drag and drop with a file from *the **Project** view* or from *the **DITA Maps Manager** view* to a DocBook 5 document that is edited in **Author** mode will create a link to the dragged file (the `link` DocBook element) at the drop location. A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the `inlinemediaobject` DocBook element with an `imagedata` child element) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

DocBook 5 Transformation Scenarios

Default transformation scenarios allow you to transform DocBook 5 documents to HTML, HTML Chunk, PDF, XHTML, XHTML Chunk, WebHelp, EPUB, and EPUB 3.

DocBook to EPUB Transformation

The EPUB specification recommends the use of *OpenType* fonts (recognized by their `.otf` file extension) when possible. To use a specific font:

- first you need to declare it in your CSS file, like:

```
@font-face {
font-family: "MyFont";
font-weight: bold;
font-style: normal;
src: url(fonts/MyFont.otf);
}
```

- tell the CSS where this font is used. To set it as default for `h1` elements, use the `font-family` rule as in the following example:

```
h1 {
font-size:20pt;
margin-bottom:20px;
font-weight: bold;
font-family: "MyFont";
text-align: center;
}
```

- in your DocBook to EPUB transformation, set the `epub.embedded.fonts` parameter to `fonts/MyFont.otf`. If you need to provide more files, use comma to separate their file paths.

> **Note:** The `html.stylesheet` parameter allows you to include a custom CSS in the output EPUB.

WebHelp Output Format

WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser.

Oxygen XML Author allows you to publish DocBook5 documents into a WebHelp format which provides both Table of Contents and advanced search capabilities.

The layout is composed of two frames:

- the left frame, containing separate tabs for **Content**, **Search**, and **Index**;

  > **Note:** In case your documents contain no `indexterm` elements, the **Index** tab is not generated.

  > **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

- the right frame where help pages are displayed.

You can navigate through the content of your output using the arrows in the upper right part of the page. These arrows allow you to move to the parent, previous, and next topic. The parents of the currently opened topic are also presented at the top of the page.

You can use this button ⊟, displayed in the **Content** tab, to collapse all the topics presented the table of contents.

The top right corner of the page contains the following options:

- **With frames** - displays the output using HTML frames to render two separate sections: a section that presents the table o contents in the left side and a section that presents the content of a topic in the right side;

Figure 121: DocBook WebHelp

To publish DocBook 5 to WebHelp, use the **DocBook WebHelp** transformation. To further customize the out-of-the-box transformation, you can edit some of its parameters:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. Default setting is `false`;
- `webhelp.copyright` - this parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output);
- `webhelp.indexer.language` - this parameter is used to identify the correct stemmer, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on
- `xml.file` - this parameter specifies the path to the DocBook XML file.
- `webhelp.logo.image` - specifies a path to an image displayed as a logo in the left side of the output's header;
- `webhelp.logo.image.target.url` - specifies a target URL set on the logo image. When you click the logo image, you will be redirected to this address.

The **Search** tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on:

- number of keywords found in a single page. The higher the number, the better;

- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.

Figure 122: WebHelp Search with Stemming Enabled

📄 **Note:** This output format is compatible with the following browsers:

- Internet Explorer 8 or newer;
- Chrome;
- Firefox;
- Safari;
- Opera.

WebHelp with Feedback Output Format

WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser.

Oxygen XML Author allows you to publish DocBook5 with Feedback documents into a WebHelp format which provides both Table of Contents and advanced search capabilities.

The layout is composed of two frames:

- the left frame, containing separate tabs for **Content**, **Search**, and **Index**;

    📄 **Note:** In case your documents contain no indexterm elements, the **Index** tab is not generated.

Note: You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

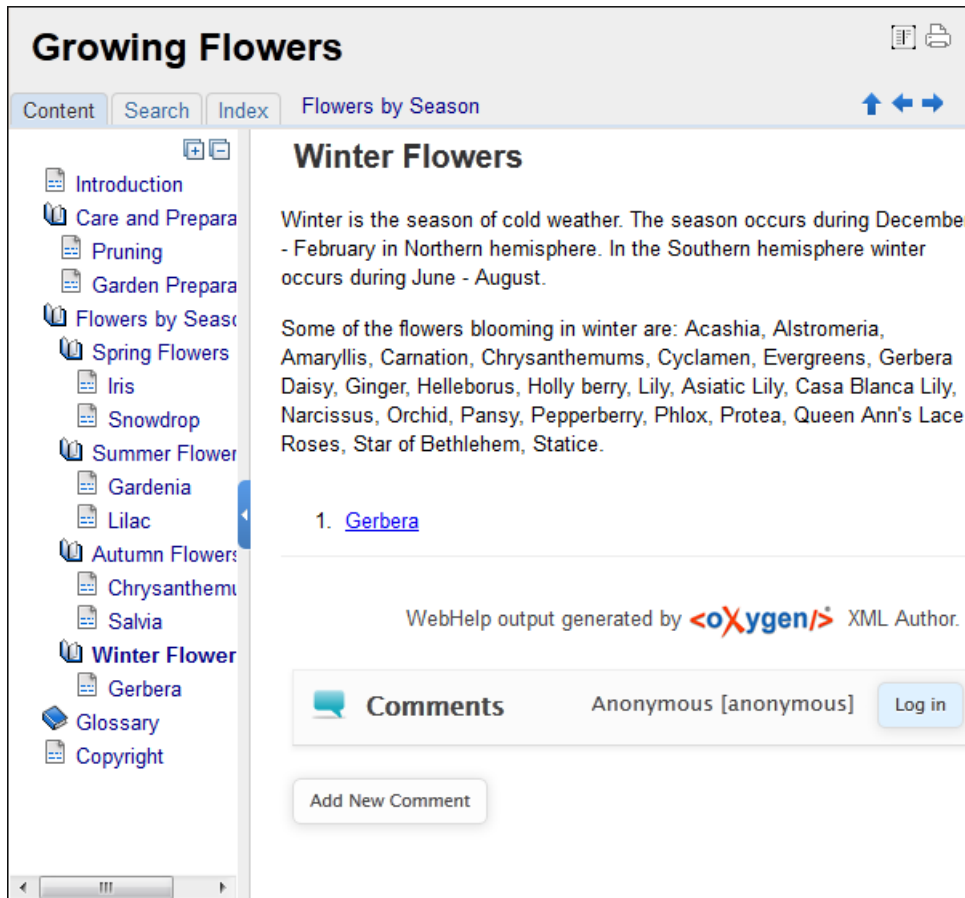- the right frame where help pages are displayed.

You can navigate through the content of your output using the arrows in the upper right part of the page. These arrows allow you to move to the parent, previous, and next topic. The parents of the currently opened topic are also presented at the top of the page.

To publish DocBook 5 to WebHelp with Feedback, use the **DocBook WebHelp with Feedback** transformation. To further customize the out-of-the-box transformation, you can edit some of its parameters:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. Default setting is `false`;
- `webhelp.copyright` - this parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output);
- `webhelp.indexer.language` - this parameter is used to identify the correct stemmer, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on
- `webhelp.product.id` - this parameter specifies a short name for the documentation target (product), for example `mobile-phone-user-guide`, `hvac-installation-guide`. You can deploy documentation for multiple products on the same server;
- `webhelp.product.version` - this parameter specifies the documentation version. New comments are bound to this version. Multiple documentation versions can be deployed on the same server;
- `xml.file` - this parameter specifies the path to the DocBook XML file.

The **Search** tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on:

- number of keywords found in a single page. The higher the number, the better;
- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.

Rules applied during search:

- the space character separates keywords. An expression like *grow flowers* counts as two separate keywords: *grow* and *flowers*;
- do not use quotes to perform exact search for multiple-word expressions. An expression like *"grow flowers"*, returns no results in our case, because it searches for two separate words: *"grow* and *flowers"* (note the quote signs attached to each word);
- `indexterm` and `keywords` DITA elements are an effective way to increase the ranking of a page. For example, content inside *keywords* elements weighs twice as much as content inside a *H1* HTML element;
- words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters, count as a single word;
- search for words containing three or more characters. Shorter words, like *to*, or *of* are ignored. This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

To watch our video demonstration about the feedback-enabled WebHelp system, go to *http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html*.

Note: This output format is compatible with the following browsers:

- Internet Explorer 8 or newer;
- Chrome;
- Firefox;
- Safari;

- Opera.

⚠ **Important:** Due to some security restrictions in Google Chrome, WebHelp pages loaded from the local system (through URLs of the `file:///...` format) may not work properly. We recommend you to load WebHelp pages in Google Chrome only from a web server.

📄 **Note:** In case you need to automate the transformation process and use it outside of Oxygen XML Author, you can use *the Oxygen XML WebHelp transformation*.

Introduction

Oxygen XML Author has the ability to transform DocBook 5 documents into feedback-enabled WebHelp systems.

WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser. It also provides table of contents and advanced search capabilities. The feedback system allows you to view discussion threads in a tree-like representation, post comments, reply to already posted comments, use stylized comments, and define administrators and moderators.

The DocBook 5 WebHelp with Feedback transformation

To publish DocBook 5 documents to WebHelp with Feedback, use the **DocBook WebHelp with Feedback** transformation scenario. You can customize the out-of-the-box transformation by editing some of its parameters:

- `root.filename` - identifies the root of the HTML file when chunking. The `root.filename` is the base filename for the chunk created for the root of each processed document;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. Default setting is `false`;
- `webhelp.copyright` - this parameter specifies the copyright note that is added in the footer of the Table of Contents frame (the left side frame of the WebHelp output);
- `webhelp.indexer.language` - this parameter is used to identify the correct stemmer, and punctuation that differs from language to language. For example, for English the value of this parameter is `en`, for French it is `fr`, and so on
- `webhelp.product.id` - this parameter specifies a short name for the documentation target (product), for example `mobile-phone-user-guide`, `hvac-installation-guide`. You can deploy documentation for multiple products on the same server;
- `webhelp.product.version` - this parameter specifies the documentation version. New comments are bound to this version. Multiple documentation versions can be deployed on the same server;
- `xml.file` - this parameter specifies the path to the DocBook XML file.

For further information about all the DocBook transformation parameters, go to *http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html*.

Before the transformation starts, enter the documentation product ID and the documentation version. After you run a **DocBook WebHelp with Feedback** transformation, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

To watch our video demonstration about the feedback-enabled WebHelp system, go to *http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html*.

Installation

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Author requires the following system components:

- Apache Web Server running
- MySQL server running

- PHP Version 5.1.6 or later

- PHP MySQL Support

- Oxygen XML WebHelp system supports the following browsers:IE7+, Chrome 19+, Firefox 11+, Safari 5+, Opera 11+

Installation Instructions

📝 **Note:** These instructions were written for XAMPP 1.7.7 with PHP 5.3.8 and for *phpMyAdmin* 3.4.5. Later versions of these packages may change the location or name of some options, however the following installation steps should remain valid and basically the same.

In case you have a web server configured with PHP, MySQL, you can deploy the WebHelp output directly. Otherwise, install XAMPP. XAMPP is a free and open source cross-platform web server solution stack package. It consists mainly of the Apache HTTP Server, MySQL database, and interpreters for scripts written in PHP.

Deploying the WebHelp output

To deploy the WebHelp output, follow these steps:

1. Locate the directory of the HTML documents. Open `http://localhost/xampp/phpinfo.php` in your browser and see the value of the DOCUMENT_ROOT variable. In case you installed XAMPP in `C:\xampp`, the value of DOCUMENT_ROOT is `C:/xampp/htdocs`
2. Copy the transformation output folder in the `DOCUMENT_ROOT`
3. Rename it to a relevant name, for example, `webhelp_1`
4. Open `http://localhost/webhelp_1/`. You are redirected to `http://localhost/webhelp_1/oxygen-webhelp/install/`

   - Verify that the prerequisites are met
   - Press **Start Installation**
   - Configure the **Deployment Settings** section. Default values are provided, but you should adjust them as needed
   - Configure the **MySql Database Connection Settings** section. Use the details from the Create the WebHelp Feedback database section to fill-in the appropriate text boxes

     ⚠️ **Warning:** Checking the **Create new database structure** option will overwrite any existing data in the selected database, if it already exists.

   - If the **Create new database structure** option is checked, the **Create WebHelp Administrator Account** section becomes available. Here you can set the administrator account data. The administrator is able to moderate new posts and manage WebHelp users.

     The same database can be used to store comments for different WebHelp deployments. If a topic is available in more than one WebHelp deployments and there are comments associated with it, you can choose to display the comments in all deployments that share the database. To do this, enable the **Display comments from other products** option. In the **Display comments from** section a list with the deployments sharing the same database is displayed. Select the deployments allowed to share common feedback.

     📝 **Note:** You can restrict the displayed comments of a product depending on its version. In case you have two products that use the same database and you restrict one of them to display comments starting from a certain version, the comments of the other product are also displayed from the specified version onwards.

   - Press **Next Step**
   - Remove the installation folder from your web server
   - Click the link pointing to the index of the documentation, or visit: `http://localhost/webhelp_1/`

To test your system, create a user and post a comment. Check if the notification emails are delivered to your inbox.

📝 **Note:** To read debug messages generated by the system:

1. Enable *JScript* logging:

   - Either: open the `log.js` file, locate the `var log= new Log(Level.NONE);` line, and change the logging level to: `Level.INFO`, `Level.DEBUG`, `Level.WARN`, or `Level.ERROR`
   - or append `?log=true` to the WebHelp URL

   - Inspect the PHP and Apache server log files.

Layout of the Feedback-Enabled WebHelp System

The layout of the feedback-enabled WebHelp system resembles the layout of the basic WebHelp, the left frame remaining the same. However, the bottom of the right frame contains a **comments** bar. Select **Log in** from this bar to authenticate as a user of the WebHelp system. In case you do not have a user name, complete the fields in the dialog box that opens to create a user. Under the **comments** bar, you can click the **Add New Comment** button to add a comment whether you are logged in or not. The tabs in the left frame have the same functionality as the Content, Search, and Index tab of the basic WebHelp.

📝 **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

Figure 123: The layout of the Feedback-Enabled WebHelp System

After you log in, your name and user name are displayed in the **Comments** bar together with the **Log of** and **Edit** buttons. Click the **Edit** button to open the **User Profile** dialog. In this dialog you can customize the following options:

- **Your Name** - you can use this field to edit the initial name that you used to create your user profile;
- **Your e-mail address** - you can use this field to edit the initial e-mail address that you used to create your profile;

- When to receive an e-mail:

  - when a comment is left on a page that you commented on;
  - when a comment is left on any topic in the Help system ;
  - when a reply is left to one of my comments.

- **New Password** - allows you to enter a new password for your user account.

  > **Note:** The **Current Password** field from the top of the **User Profile** is mandatory in case you want to save the changes you make.

Advanced Customization and Management

Apart from the options available for a regular user, you can also use the administrative page for advanced customization and management. As an administrator, you have full access to all the features of the feedback-enabled WebHelp system. To access the administrative page, select **Admin Panel** from the **Comments** bar.

Figure 124: The Administrative Page

This page allows you to view all posts, export comments and set the version of the WebHelp system. You can also view the details of each user and search through these details using the **Search User Information** filter.

The upper part of the page contains the following actions:

- **Delete Orphan Comments** - deletes comments associated with topics that are no longer available
- **Delete Pending Users** - deletes all unconfirmed users that registered more than a week ago
- **View All Posts** - allows you to view all posts associated with a product and version
- **Export Comments** - allows you to export in XML format all posts associated with a product and version
- **Set Version** - use this action to display comments starting from a particular version

To edit the details of a user, click the corresponding row. Use the **Edit User** dialog to customize all the information associated with an user:

- **Name** - The user's full name
- **Level** - Use this field to modify the privilege level of the currently edited user. You can choose from:

  - **User** - regular user, able to post comments and receive e-mail notifications
  - **Moderator** - in addition to the regular **User** rights, this type of user has access to the **Admin Panel**. In the administrative page a moderator can view, delete, export comments and set the version of the feedback-enabled WebHelp system.
  - **Admin** - full administrative privileges. Can manage WebHelp-specific settings, users and their comments.

- **Company** - User's organization name
- **E-mail** - User's contact e-mail address. This is also the address where the WebHelp system sends notifications:

  - **WebHelp Notification** - when enabled, the user receives notifications when comments are posted anywhere in the feedback-enabled WebHelp system
  - **Reply Notification** - when enabled, the user receives notifications when comments are posted as a reply to one of his or hers comments

- **Page Notification** - when enabled, the user receives notifications when comments are posted on a topic where he or she posted a comment

- **Date** - User registration date
- **Status** - Specifies the status of the currently edited user:

  - **Created** - the user is created but does not have any rights over the feedback-enabled WebHelp system
  - **Validated** - the user is able to use the feedback-enabled WebHelp system
  - **Suspended** - the user has no rights over the feedback-enabled WebHelp system

Localizing the Email Notifications for DocBook to WebHelp with Feedback Transformation Scenario

The WebHelp with Feedback system uses emails to notify users when comments are posted. These emails are based on templates stored in the WebHelp directory. The default messages are in English, French, German and Japanese and they are stored in the WebHelp directory. For example, the English messages are stored in this directory: `[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\en`

We'll suppose that you want to localize the emails into Dutch. Follow these steps:

- create the following directory:

  `[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl`

- copy all English template files from `[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\en` and paste them into the directory you just created
- edit the HTML files from the `[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl` directory and translate the content into Dutch
- start Oxygen XML Author and edit the *WebHelp with Feedback* transformation scenario
- in the **Parameters** tab look for the `l10n.gentext.default.language` parameter and set its value to the appropriate language code. In our example, use the value `nl` for Dutch

  > **Note:** If you set the parameter to a value such as `LanguageCode-CountryCode` (for example, `en-us`), the transformation scenario will only use the language code

- execute the transformation scenario to obtain the *WebHelp with Feedback* output

WebHelp Mobile Output Format

To further improve its ability to create online documentation, Oxygen XML Author offers support to transform DocBook documents into mobile WebHelp systems. This feature generates an output that works on multiple platforms (Android, iOS, BlackBerry, Windows Mobile) and is specially designed for mobile devices. All the specific touch screen gestures are supported. The functionality of the desktop WebHelp layout is preserved, offering table of contents, search capabilities, and index navigation, organized in an intuitive layout.

Figure 125: Mobile WebHelp

To generate a mobile WebHelp system from your DocBook 5 document, go to the **DITA Maps Manager** view, click

Configure Transformation Scenarios() and select the **DocBook WebHelp - Mobile** transformation scenario from the **DocBook 5** section. Click **Apply associated**. Once Oxygen XML Author finishes the transformation process, the output is opened in your default browser automatically.

To customize the TOC section of the output, you need to alter the `oxygen-webhelp\resources\skins\mobile\toc.css` stylesheet.

DocBook to PDF Output Customization

Main steps for customization of PDF output generated from DocBook XML documents.

When the default layout and output look of the DocBook to PDF transformation need to be customized, the following main steps should be followed. In this example a company logo image is added to the front matter of a book. Other types of customizations should follow some similar steps.

1. Create a custom version of the DocBook title spec file.

   You should start from a copy of the file `[OXYGEN_DIR]/frameworks/docbook/xsl/fo/titlepage.templates.xml` and customize it. The instructions for the spec file can be found *here.*

   An example of spec file:

   ```
   <t:titlepage-content t:side="recto">
       <mediaobject/>
       <title
           t:named-template="book.verso.title"
           font-size="&hsize2;"
           font-weight="bold"
           font-family="{$title.font.family}"/>
       <corpauthor/>
       ...
   </t:titlepage-content>
   ```

2. Generate a new XSLT stylesheet from the title spec file from the previous step.

   Apply `[OXYGEN_DIR]/frameworks/docbook/xsl/template/titlepage.xsl` to the title spec file. The result is an XSLT stylesheet, let's call it `mytitlepages.xsl`.

3. Import `mytitlepages.xsl` in a *DocBook customization layer*.

The customization layer is the stylesheet that will be applied to the XML document. The `mytitlepages.xsl` should be imported with an element like:

```
<xsl:import href="dir-name/mytitlepages.xsl"/>
```

4. Insert logo image in the XML document.

The path to the logo image must be inserted in the `book/info/mediaobject` element of the XML document.

5. Apply the customization layer to the XML document.

A quick way is duplicating the transformation scenario **DocBook PDF** that comes with Oxygen and set the customization layer in *the XSL URL property of the scenario*.

Adding Videos in the Output

Videos can be referred and played in all HTML5-based output formats (like *WebHelp*). For example, to add a YouTube video in the WebHelp output generated from a DocBook document, follow these steps:

• edit the DocBook document and refer the video using an `mediaobject` element like in the following example:

```
<mediaobject>
  <videoobject>
    <videodata fileref="http://www.youtube.com/watch/v/VideoName"/>
  </videoobject>
</mediaobject>
```

• execute a *WebHelp* or *WebHelp with Feedback* transformation scenario to obtain the output

DocBook 5 Templates

Default templates are available in the *New File wizard* and can be used for easily creating a skeletal form of a DocBook 5 book or article. These templates are stored in the `[OXYGEN_DIR]/frameworks/docbook/templates/DocBook 5` folder.

Here are some of the DocBook 5 templates available when creating *new documents from templates*.

• **Article**;
• **Article with MathML**;
• **Article with SVG**;
• **Article with XInclude**;
• **Book**;
• **Book with XInclude**;
• **Chapter**;
• **Section**;
• **Set of Books**.

Inserting olink Links in DocBook 5 Documents

An `olink` is a type of link between two DocBook XML documents.

The `olink` element is the equivalent for linking outside the current DocBook document. It has the attribute `targetdoc` for the document ID that contains the target element and the attribute `targetptr` for the ID (the value of an `id` or `xml:id` attribute) of the target element. The combination of those two attributes provides a unique identifier to locate cross references.

For example, the *Administrator Guide* is a book with the document ID `MailAdminGuide` and it contains a chapter about user accounts like the following:

```
<chapter id="user_accounts">
<title>Administering User Accounts</title>
<para>blah blah</para>
...
```

You can form a cross reference to that chapter by adding an `olink` in the *User Guide* like the following:

```
You may need to update your
<olink targetdoc="MailAdminGuide" targetptr="user_accounts">user accounts
</olink>
when you get a new machine.
```

1. Decide what documents are included in the domain for cross referencing.

   An ID should be assigned to each document that will be referenced with an `olink`. Usually it is added as an `id` or `xml:id` attribute to the root element of the document. A document ID is a string that is unique for each document in your collection. For example the documentation may include a user's guide, an administrator's guide, and a reference document. These could have simple IDs like `ug`, `ag`, and `ref` or more specific IDs like `MailUserGuide`, `MailAdminGuide`, and `MailReference`.

2. Decide the output hierarchy.

   For creating links between documents, the relative locations of the output documents must be known. Generally the HTML files for multiple documents are output to different directories if chunking is used. Before going further you must decide the names and locations of the HTML output directories for all the documents from the domain. Each directory will be represented by an element `<dir name="directory_name">` in the target database document. In the example from the next step the hierarchy is `documentation/guides/mailuser`, `documentation/guides/mailadmin`, `documentation/guides/reference`.

3. Create the target database document.

   Each collection of documents has a master target database document that is used to resolve all `olinks` from that collection. The target database document is an XML file that is created once. It provides a framework that pulls in the target data for each document. The database document is static and all the document data is pulled in dynamically. An example is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE targetset [
<!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:

```
<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 provides the **Insert OLink** action on the toolbar. This action allows selecting the target of an `olink` from the list of all possible targets from a specified target database document. In the following image the target database document is called `target.xml`.
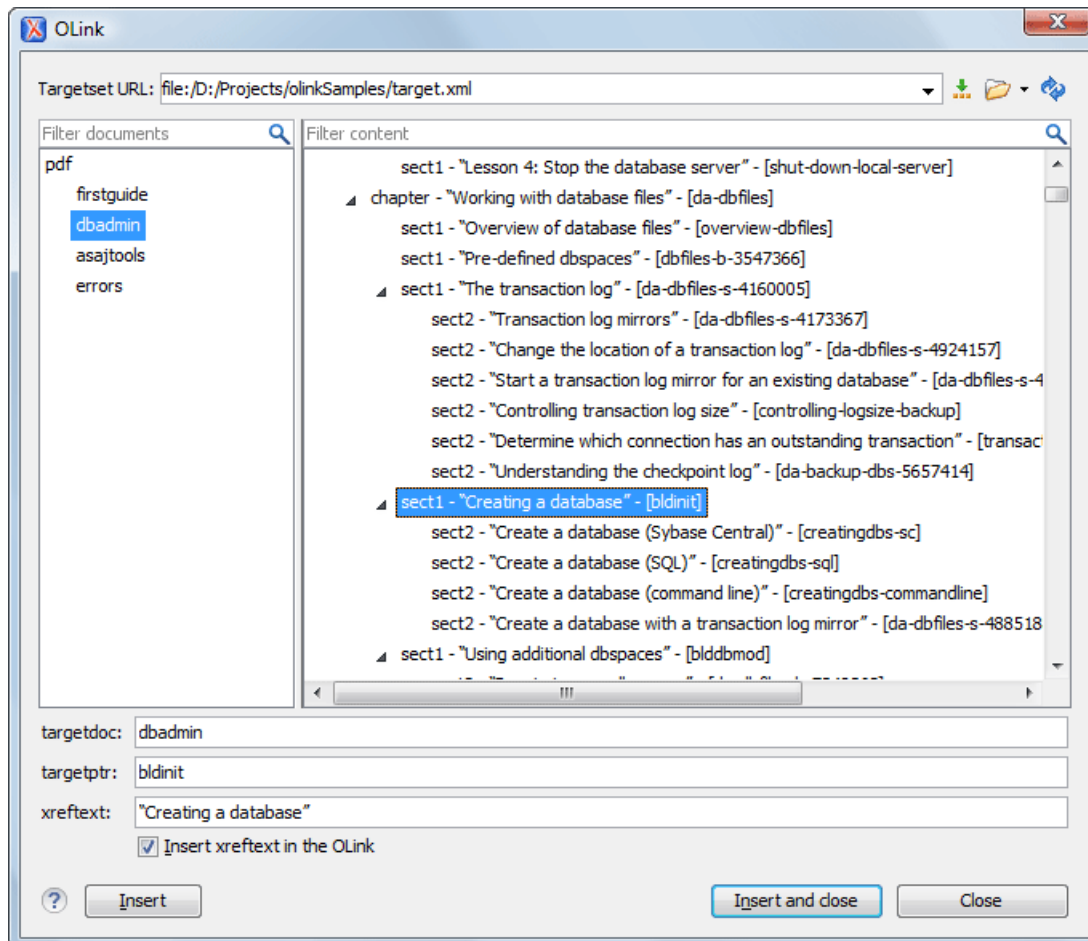
Figure 126: Insert OLink Dialog

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 `[OXYGEN_DIR]/frameworks/docbook/xsl/common/targetdatabase.dtd`.

DocBook Targetset Templates

There is a default template for *Targetset* documents in the `[OXYGEN_DIR]/frameworks/docbook/templates/Targetset` folder. It is available when creating *new documents from templates*.

• **Docbook Targetset - Map** - New Targetset Map.

The DITA Topics Document Type

The Darwin Information Typing Architecture (DITA) is an XML-based architecture oriented to authoring, producing, and delivering technical information. It divides content into small, self-contained topics that you can reuse in different deliverables. The extensibility of DITA permits organizations to define specific information structures and still use standard tools to work with them. Oxygen XML Author provides schema driven (DTD, RNG, XSD) templates for DITA documents.

A file is considered to be a DITA topic document when either of the following occurs:

• the root element name is one of the following: `concept`, `task`, `reference`, `dita`, `topic`;
• PUBLIC ID of the document is one of the PUBLIC ID's for the elements above;
• the root element of the file has an attribute named `DITAArchVersion` attribute from the "http://dita.oasis-open.org/architecture/2005/" namespace. This enhanced case of matching is only applied when the **Enable DTD processing** option from the *Document Type Association preferences page* is enabled.

The default schema used for DITA topic documents is located in `[OXYGEN_DIR]/frameworks/dita/dtd/ditabase.dtd`, where *${frameworks}* is a subdirectory of the Oxygen XML Author install directory.

The CSS file used for rendering DITA content in Author mode is `[OXYGEN_DIR]/frameworks/dita/css/dita.css`.

The default XML catalog is `[OXYGEN_DIR]/frameworks/dita/catalog.xml`.

DITA Author Extensions

The specific actions for a DITA topic are:

B **Bold**

Surrounds the selected text with a `b` tag.

I **Italic**

Surrounds the selected text with an `i` tag.

U **Underline**

Surrounds the selected text with a `u` tag.

Note:

Bold, **Italic**, and **Underline** are toggle actions.

For all of the above actions, if there is no selection in the document, then a new specific tag is inserted. These actions are available in any document context.

Style Guide

Opens the **DITA Style Guide Best Practices for Authors** in your browser and presents the description of the element at the caret position. When editing DITA documents, this action is available in the contextual menu of the editing area (under the **About Element** sub-menu), in the **DITA** menu, and in some of the documentation tips displayed by the **Content Completion Assistant**.

Browse reference manual

Opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position. When editing DITA documents, this action is available in the contextual menu of the editing area (under the **About Element** sub-menu) and in the documentation tip displayed by the **Content Completion Assistant**.

Cross Reference

Inserts an `xref` element with the value of the `format` attribute set to `dita`. The target of the `xref` element is selected in a dialog box that lists all the IDs extracted from the selected file. When you select an ID, you can preview the content in the **Preview** tab or the XML source in the **Source** tab. In case you have a large number of IDs in the target document, use the **Filter** field to search through the IDs.
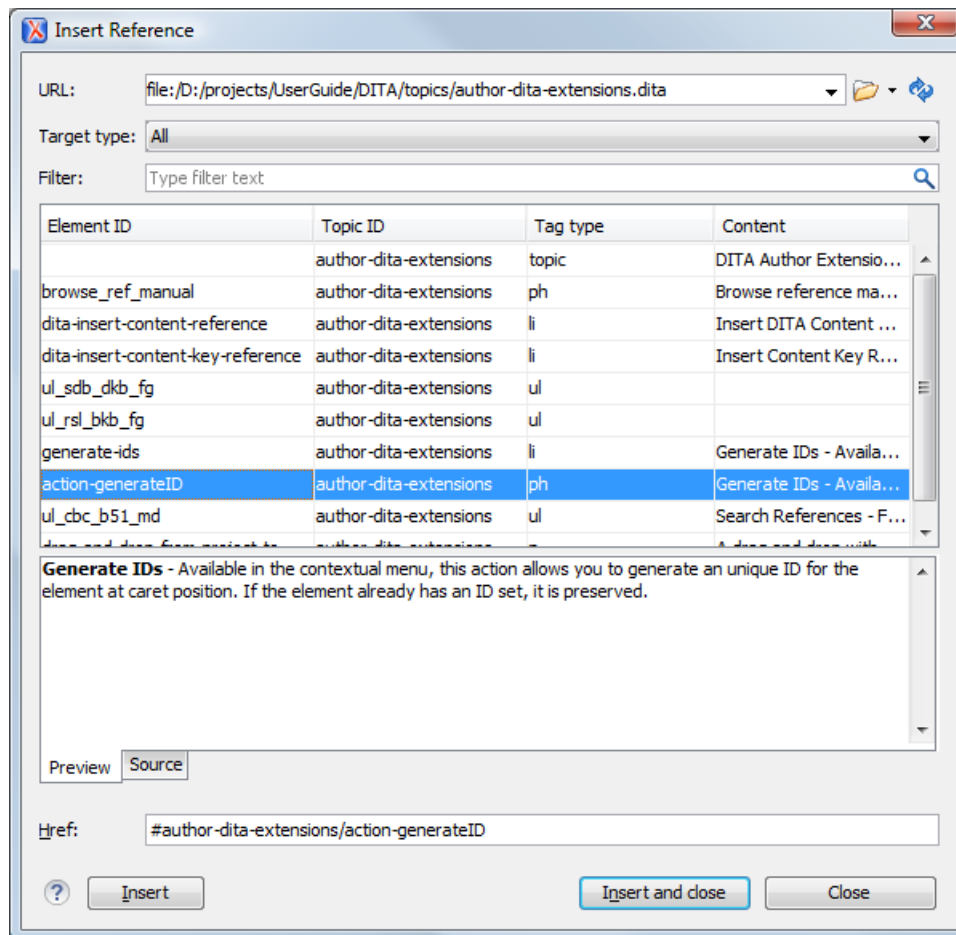
Figure 127: Insert a Cross Reference in a DITA Document

Note: The **Insert Reference** dialog is not modal. The dialog is closed automatically in case you switch to a different editor.

Key Reference

Inserts a user specified element with the value of the `keyref` attribute set to a specific key name. As stated in the DITA 1.2 specification, keys are defined at map level and referenced afterwards. You are able to select the target of the `keyref` element in the **Insert Key Reference** dialog box.

> Note: The **Insert Key Reference** dialog box presents the list of keys available in the current DITA Map. If the DITA Map is not opened in the **DITA Maps Manager** view, the **Insert Key Reference** dialog does not display any keys.

You can also reference elements at sub-topic level by pressing the **Sub-topic** button and choosing the target.

All keys which are presented in the dialog are gathered from the current opened DITA map. Elements which have the `keyref` attribute set are displayed as links. The current opened DITA map is also used to resolve references when navigating `keyref` links in the Author mode. Image elements which use key references are rendered as images.

File Reference

Inserts an `xref` element with the value of attribute `format` set to `xml`.

Web Link

Inserts an `xref` element with the value of attribute `format` set to `html`, and `scope` set to `external`.

Related Link to Topic

Inserts a `link` element inside a `related-links` parent.

Related Link to File

Inserts a `link` element with the `format` attribute set to `xml` inside a `related-links` parent.

Related Link to Web Page

Inserts a `link` element with the attribute `format` set to `html` and `scope` set to `external` inside a `related-links` parent.

> Note: The actions for inserting references described above are grouped in the **DITA** > **Link** menu.

Paste as content reference

Available on the contextual menu of Author editor for any topic file, this operation inserts a content reference (a DITA element with a `conref` attribute) to the DITA XML element from the clipboard. An entire DITA XML element with an ID attribute must be present in the clipboard when the action is invoked. The `conref` attribute will point to this ID value.

Paste as content key reference

Allows you to indirectly refer content using the `conkeyref` attribute. When the DITA content is processed, the key references are resolved using key definitions from DITA maps. To use this action, do the following:

- set the `id` attribute of the element holding the content you want to refer
- open the DITA Map in the **DITA Maps Manager** view
- right click the topic that holds the content you want to refer, select **Edit Properties**, and enter a value in the **Keys** field
- make sure the DITA Map remains open in the **DITA Maps Manger** view

Paste as link

Available on the contextual menu of Author editor for any topic file, this action inserts a `link` element or an `xref` one (depending on the location of the paste operation) that points to the DITA XML element from the clipboard. An entire DITA XML element with an ID attribute must be present in the clipboard when the action is invoked. The `href` attribute of `link/href` will point to this ID value.

Paste as link (keyref)

Inserts a link to the element that you want to refer. To use this action, do the following:

- set the `id` attribute of the element that you want to refer

- open the DITA Map in the **DITA Maps Manager** view
- right click the topic that holds the content you want to refer, select **Edit Properties**, and enter a value in the **Keys** field
- make sure the DITA Map remains open in the **DITA Maps Manger** view

§ **Insert Section / Step**

Inserts a new section / step in the document, depending on the current context. A new section will be inserted in either one of the following contexts:

- section context, when the value of `class` attribute of the current element or one of its ancestors contains `topic` or `section`.
- topic's body context, when the value of `class` attribute of the current element contains `topic/body`.

A new step will be inserted in either one of the following contexts:

- task step context, when the value of `class` attribute of the current element or one of its ancestors contains `task/step`.
- task steps context, when the value of `class` attribute of the current element contains `task/steps`.

Insert Concept

Inserts a new concept. Concepts provide background information that users must know before they can successfully work with a product or interface. This action is available in one of the following contexts:

- concept context, one of the current element ancestors is a `concept`. In this case an empty `concept` will be inserted after the current `concept`.
- concept or DITA context, current element is a `concept` or `dita`. In this case an empty `concept` will be inserted at current caret position.
- DITA topic context, current element is a `topic` child of a `dita` element. In this case an empty `concept` will be inserted at current caret position.
- DITA topic context, one of the current element ancestors is a DITA `topic`. In this case an empty `concept` will be inserted after the first `topic` ancestor.

Insert Task

Inserts a new task. Tasks are the main building blocks for task-oriented user assistance. They generally provide step-by-step instructions that will enable a user to perform a task. This action is available in one of the following contexts:

- task context, one of the current element ancestors is a `task`. In this case an empty `task` will be inserted after the last child of the first `concept`'s ancestor.
- task context, the current element is a `task`. In this case an empty `task` will be inserted at current caret position.
- topic context, the current element is a `dita topic`. An empty `task` will be inserted at current caret position.
- topic context, one of the current element ancestors is a `dita topic`. An empty `task` will be inserted after the last child of the first ancestor that is a `topic`.

Insert Reference

Inserts a new reference in the document. A reference is a top-level container for a reference topic. This action is available in one of the following contexts:

- reference context - one of the current element ancestors is a `reference`. In this case an empty `reference` will be inserted after the last child of the first ancestor that is a `reference`.
- `reference` or `dita` context - the current element is either a `dita` or a `reference`. An empty `reference` will be inserted at caret position.
- topic context - the current element is `topic` descendant of `dita` element. An empty `reference` will be inserted at caret position.
- topic context - the current element is descendant of `dita` element and descendant of `topic` element. An empty `reference` will be inserted after the last child of the first ancestor that is a `topic`.

Insert image reference

inserts an image reference at the caret position. Depending on the current context, an image-type DITA element is inserted. Also you can use this action to *refer to MathML files*.

Insert DITA Content Reference

Inserts a content reference at the caret position.

The DITA `conref` attribute provides a mechanism for reuse of content fragments. The `conref` attribute stores a reference to another element and is processed to replace the referencing element with the referenced element. The element containing the content reference acts as a placeholder for the referenced element. The identifier for the referenced element must be either absolute or resolvable in the context of the referencing element. See *here* for more details.

Oxygen XML Author *displays the referred content* of a DITA `conref` if it can resolve it to a valid resource. If you have URI's instead of local paths in the XML documents and your DITA OT transformation needs an XML catalog to map the URI's to local paths you have to *add the catalog to Oxygen XML Author*. If the URI's can be resolved the referred content will be displayed in Author mode and in the transformation output.

A content reference is inserted with the action **Insert a DITA Content Reference** available on the toolbar **Author custom actions** and on the menu **DITA** > **Insert**.
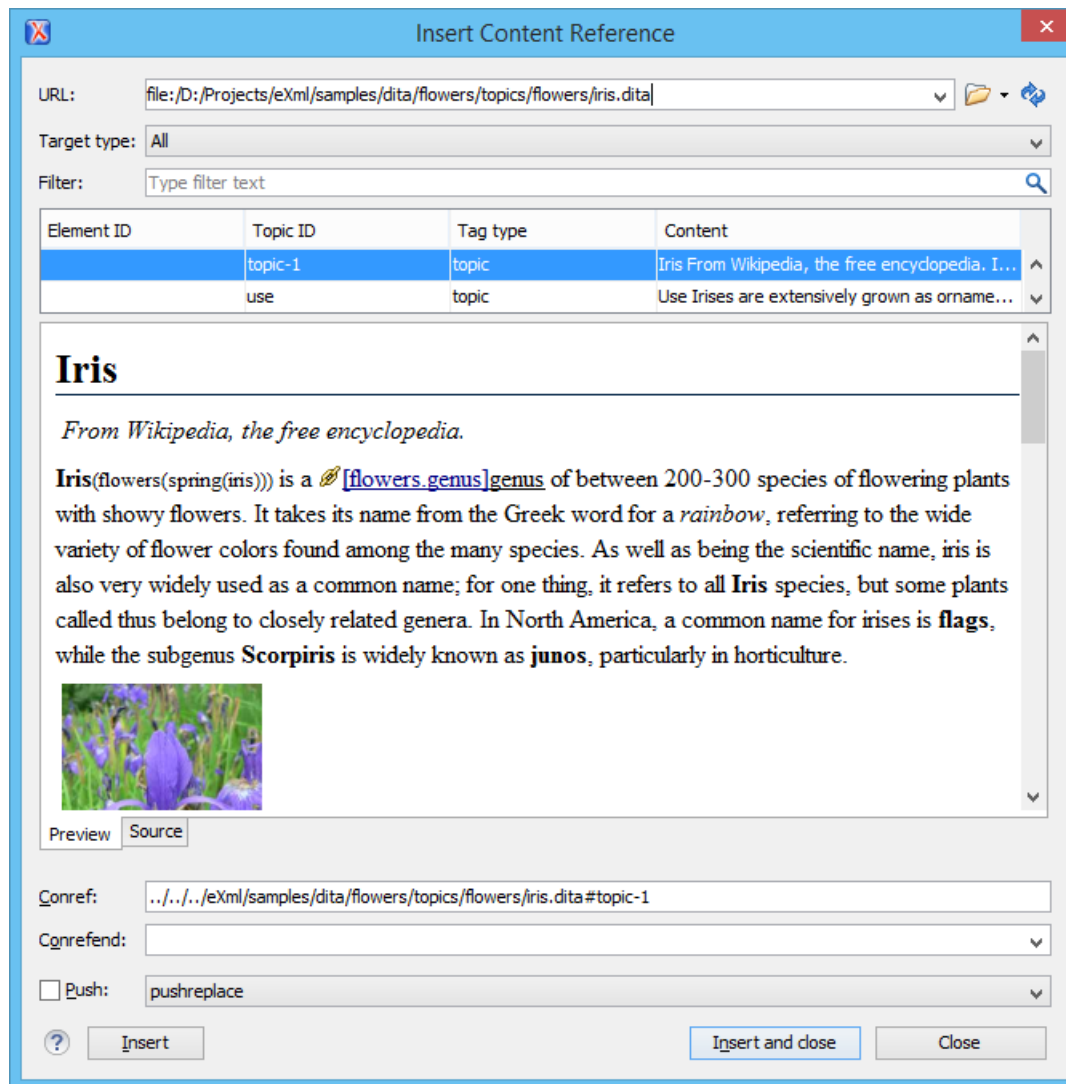
Figure 128: Insert Content Reference Dialog

Note: The **Insert Content Reference** dialog is not modal. The dialog is closed automatically in case you switch to a different editor.

In the URL chooser you set the URL of the file from which you want to reuse content. Depending on the **Target type** filter you will see a tree of elements which can be referred (which have ID's). For each element the XML content is shown in the preview area. The **Conref** value is computed automatically for the selected tree element. After pressing **Insert**, an element with the same name as the target element and having the attribute `conref` with the value specified in the **Conref** value field will be inserted at caret position.

According to the DITA 1.2 specification the `conrefend` attribute can be used to specify content reference ranges. This is a very useful feature when referencing multiple consecutive steps or list items. If you use multiple contiguous sibling selection the `conrefend` value will also be set to the value of the last selected ID path. Oxygen XML Author will present the entire referenced range as read-only content.

Insert Content Key Reference

Inserts a content key reference at the caret position.

As stated in the DITA 1.2 specification the `conkeyref` attribute provides a mechanism for reuse of content fragments similar with the `conref` mechanism. Keys are defined at map level which can be referenced using `conkeyref`. The `conkeyref` attribute contains a key reference to another element and is processed to replace the referencing element with the referenced element. The element containing the content key reference acts as a placeholder for the referenced element. The identifier for the referenced element must be either absolute or resolvable in the context of the referencing element.

Oxygen XML Author *displays the key referred content* of a DITA `conkeyref` if it can resolve it to a valid resource in the context of the current opened DITA map.

A content key reference is inserted with the action **Insert a DITA Content Key Reference** available on the toolbar **Author custom actions** and on the menu **DITA** > **Insert**.
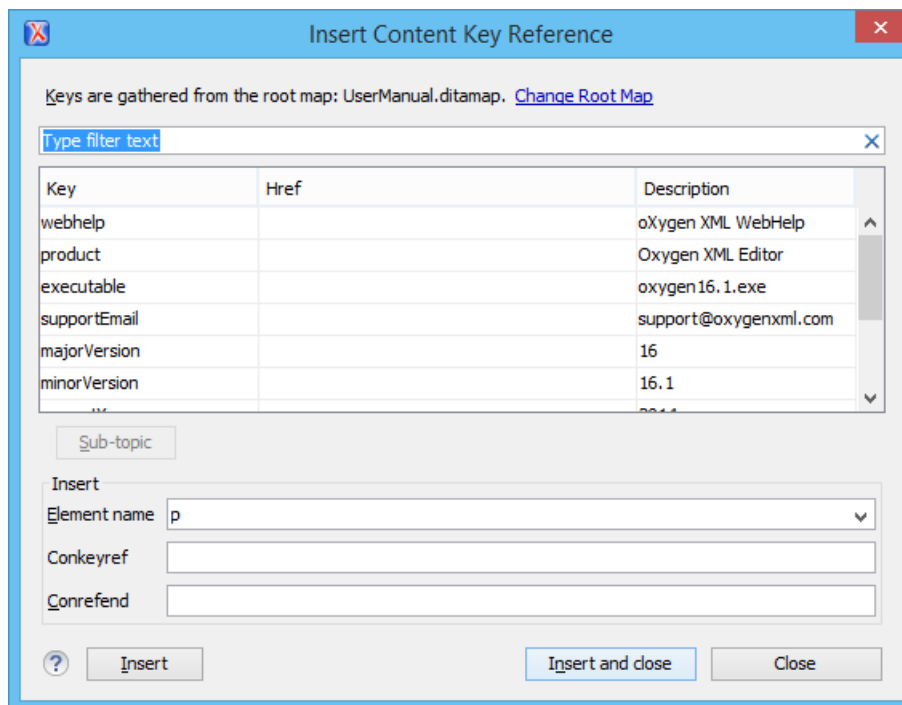
Figure 129: Insert Content Key Reference Dialog

Note: The **Insert Content Key Reference** dialog is not modal. The dialog is closed automatically in case you switch to a different editor.

To reference target elements at sub-topic level just press the **Sub-topic** button and choose the target.

According to the DITA 1.2 specification the `conrefend` attribute can be used to specify content reference ranges. This is a very useful feature when referencing multiple consecutive steps or list items. If you use multiple contiguous sibling selection for IDs at sub-topic level the `conrefend` value will also be set to the value of the last selected ID path. Oxygen XML Author will present the entire referenced range as read-only content.

> ⚠ **Important:** All keys which are presented in the dialog are gathered from the current opened DITA map. Elements which have the `conkeyref` attribute set are displayed by default with the target content expanded. The current opened DITA map is also used to resolve references when navigating `conkeyref` links in the Author 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. For more information, see section *Editing MathML Notations*.

Create Reusable Component

Creates a reusable component from a selected fragment of text. For more information, see *Reusing Content*.

Insert Reusable Component

Inserts a reusable component at cursor location. For more information, see *Reusing Content*.

Remove Content Reference

Removes the `conref` attribute of an element. For more information, see *Reusing Content*.

Add/Edit Content Reference

Add or edit the `conref` attribute of an element. For more information, see *Reusing Content*.

Insert Table

Opens a dialog that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

Insert Row

Inserts a new table row with empty cells. This action is available when the caret is positioned inside a table.

Insert Column

Inserts a new table column with empty cells after the current column. This action is available when the caret is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, Oxygen XML Author a new cell at caret position. If the caret is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at caret position.

Delete Row

Deletes the table row located at caret position.

Insert Row Above

Inserts a row above the current one.

Insert Row Below

Inserts a row below the current one.

Insert Column Before

Inserts a column before the current one.

Insert Column After

Inserts a column after the current one.

Join Row Cells

Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current caret position with the content of the cell above it. This action works only if both cells have the same column span.

Join Cell Below

Joins the content of the cell from the current caret position with the content of the cell below it. This action works only if both cells have the same column span.

> **Note:** When you use **Join Cell Above** and **Join Cell Below**, Oxygen XML Author deletes the source row is case it remains empty. The cells that span over multiple rows are also updated.

Split Cell To The Left

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current caret position in two cells, inserting a new empty table cell above. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current caret position in two, inserting a new empty table cell below. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also inserted automatically by default.

Insert an unordered list at the caret position

Inserts an itemized list. A child list item is also inserted automatically by default.

Insert a step or list Item

Inserts a new list item in any of the above list types.

- The following options are available for customizing the frame of a table:

  - `none` - the table has no frame
  - `all` - the table has top, bottom and side borders
  - `top` - the table has a top border
  - `topbot` - the table top and bottom borders
  - `bottom` - the table has a bottom border
  - `sides` - the table has side border
  - `-dita-use-conref-target` - it indicates that when the element uses *conref* to pull in content, the attribute with a value of **-dita-use-conref-target** should also be pulled in from the target
  - `unspecified` - if you select this option, when you insert the table, the frame attribute is not inserted

Note: DITA supports the CALS table model similar with DocBook document type in addition to the `simpletable` element specific for DITA.

Caution: Column specifications are required for table actions to work properly.

Generate IDs

This action generates and sets unique IDs for:

- the element at caret position
- all top-level elements found in the current selection. Additionally, if the selection contains elements from the **DITA** > **ID Options** list, they will all receive an unique ID

Note: IDs already set are preserved.

The action is available both in the contextual menu and in the **DITA** main menu.

ID Options

Action available in the **DITA** main menu, allows you to specify the elements for which Oxygen XML Author generates an unique ID if the **Auto generate IDs for elements** option is enabled. The configurable ID value pattern can accept most of the application supported *editor variables*.

To keep an already set element ID when copying content in the same document, make sure the **Remove IDs 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 (Command+Shift+G on OS X)** 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 OS X, etc) will insert an image element (the `image` DITA element with the `href` attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

DITA Transformation Scenarios

The following default transformation scenarios are available for DITA Topics:

- **DITA XHTML** - Transforms a DITA topic to XHTML using DITA Open Toolkit 1.6.1;
- **DITA PDF** - Transforms a DITA topic to PDF using the DITA Open Toolkit 1.6.1 and the Apache FOP engine.

DITA Templates

The default templates available for DITA topics are stored in `[OXYGEN_DIR]/frameworks/dita/templates/topic` folder. They can be used for easily creating a DITA `concept`, `reference`, `task` or `topic`.

Here are some of the DITA templates available when creating *new documents from templates*:

- **Composite** - New DITA Composite
- **Composite with MathML** - New DITA Composite with MathML
- **Concept** - New DITA Concept
- **General Task** - New DITA Task
- **Glossentry** - New DITA Glossentry
- **Glossgroup** - New DITA Glossgroup
- **Machinery Task** - New DITA Machinery Task

- **Reference** - New DITA Reference
- **Task** - New DITA Task
- **Topic** - New DITA Topic
- **Learning Assessment** - New DITA Learning Assessment (learning specialization in DITA 1.2)
- **Learning Content** - New DITA Learning Content (learning specialization in DITA 1.2)
- **Learning Summary** - New DITA Learning Summary (learning specialization in DITA 1.2)
- **Learning Overview** - New DITA Learning Overview (learning specialization in DITA 1.2)
- **Learning Plan** - New DITA Learning Plan (learning specialization in DITA 1.2)
- **Troubleshooting** - Experimental DITA 1.3 troubleshooting specialization

DITA for Publishers topic specialization templates:

- **D4P Article** - New DITA for Publishers article
- **D4P Chapter** - New DITA for Publishers chapter
- **D4P Concept** - New DITA for Publishers concept
- **D4P Conversion Configuration** - New DITA for Publishers conversion configuration
- **D4P Cover** - New DITA for Publishers cover
- **D4P Part** - New DITA for Publishers part
- **D4P Sidebar** - New DITA for Publishers sidebar
- **D4P Subsection** - New DITA for Publishers subsection
- **D4P Topic** - New DITA for Publishers topic

The DITA Map Document Type

DITA maps are documents that collect and organize references to DITA topics to indicate the relationships among the topics. They can also serve as outlines or tables of contents for DITA deliverables and as build manifests for DITA projects.

Maps allow scalable reuse of content across multiple contexts. They can be used by information architects, authors, and publishers to plan, develop, and deliver content.

A file is considered to be a DITA map document when either of the following is true:

- The root element name is one of the following: `map`, `bookmap`.
- The public id of the document is `-//OASIS//DTD DITA Map` or `-//OASIS//DTD DITA BookMap`.
- The root element of the file has an attribute named `class` which contains the value `map/map` and a `DITAArchVersion` attribute from the *http://dita.oasis-open.org/architecture/2005/* namespace. This enhanced case of matching is only applied when the **Enable DTD processing** option from the *Document Type Detection option page* is enabled.

The default schema used for DITA map documents is located in `[OXYGEN_DIR]/frameworks/dita/DITA-OT/dtd/map.dtd`.

The CSS file used for rendering DITA content is located in `[OXYGEN_DIR]/frameworks/dita/css/dita.css`.

The default XML catalog is stored in `[OXYGEN_DIR]/frameworks/dita/catalog.xml`.

DITA Map Author Extensions

Specific actions for DITA map documents are:

Insert New Topic

Creates a new topic and inserts a reference to it at the caret position.

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 Relationship 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.

Table Properties

Allows you to change the table's rows properties.

Insert Row

Inserts a new table row with empty cells. The action is available when the caret position is inside a table.

Insert Column

Inserts a new table column with empty cells after the current column. The action is available when the caret position is inside a table.

Delete Column

Deletes the table column where the caret is located.

Delete Row

Deletes the table row where the caret is located.

All actions described above are available in the contextual menu, the **DITA** submenu of the main menu and in the **Author custom actions** toolbar.

A drag and drop with a file from *the **Project** view* or from *the **DITA Maps Manager** view* to a DITA map document that is edited in Author mode will create a link to the dragged file (a `topicref` element, a `chapter` one, a `part` one, etc.) at the drop location.

DITA Map Transformation Scenarios

The following default transformations are available:

- Predefined transformation scenarios allow you to transform a DITA Map to PDF, ODF, XHTML, WebHelp, EPUB and CHM files.
- **Run DITA OT Integrator** - Use this transformation scenario if you want to integrate a DITA OT plugin. This scenario runs an ANT task that integrates all the plug-ins from DITA-OT/plugins directory.
- **DITA Map Metrics Report** - Use this transformation scenario if you want to generate a DITA Map statistics report containing information like:

  - the number of processed maps and topics
  - content reuse percentage
  - number of elements, attributes, words, and characters used in the entire DITA Map structure
  - DITA conditional processing attributes used in the DITA Maps
  - words count
  - information types like number of containing maps, bookmaps, or topics

Many more output formats are available by clicking the **New** button. The transformation process relies on DITA Open Toolkit 1.6.1.

WebHelp Output Format

WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser.

Oxygen XML Author allows you to publish a DITA Map into a WebHelp format that provides both Table of Contents and advanced search capabilities.

The layout is composed of two frames:

• the left frame, containing separate tabs for **Content**, **Search**, and **Index**;

> **Note:** In case your documents contain no `indexterm` elements, the **Index** tab is not generated.

> **Note:** You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

• the right frame where help pages are displayed.

You can navigate through the content of your output using the arrows in the upper right part of the page. These arrows allow you to move to the parent, previous, and next topic. The parents of the currently opened topic are also presented at the top of the page.

> **Note:** You can edit the `args.hide.parent.link` parameter to hide the **Parent**, **Next**, and **Previous** links.

You can use this button , displayed in the **Content** tab, to collapse all the topics presented the table of contents.

The top right corner of the page contains the following options:

• **With frames** - displays the output using HTML frames to render two separate sections: a section that presents the table o contents in the left side and a section that presents the content of a topic in the right side;
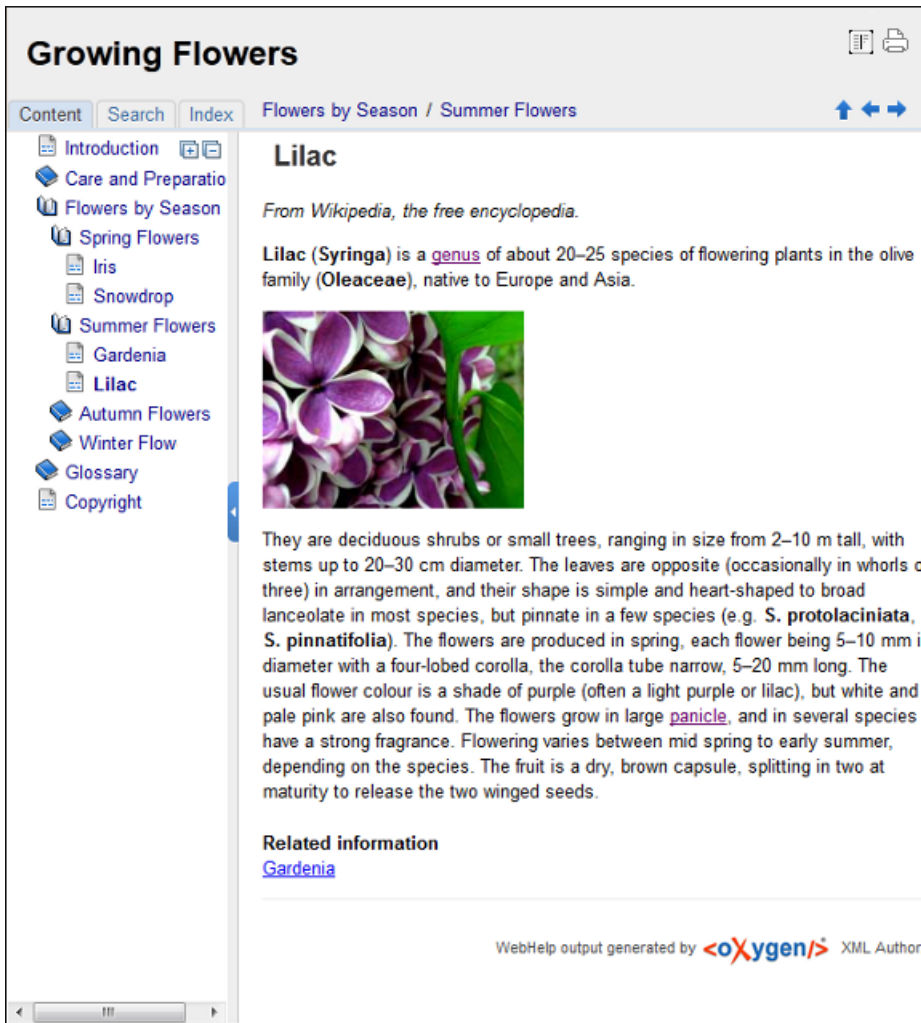
Figure 130: WebHelp Output

To publish the DITA map to WebHelp, you can use the **DITA Map WebHelp** transformation. The *WebHelp Customization* topic describes each parameter which can be configured in order to customize the output.

The **Search** tab is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on:

- number of keywords found in a single page. The higher the number, the better;
- context - if a word is found in a title or emphasized section of text it scores better than a word found in an unformatted text.
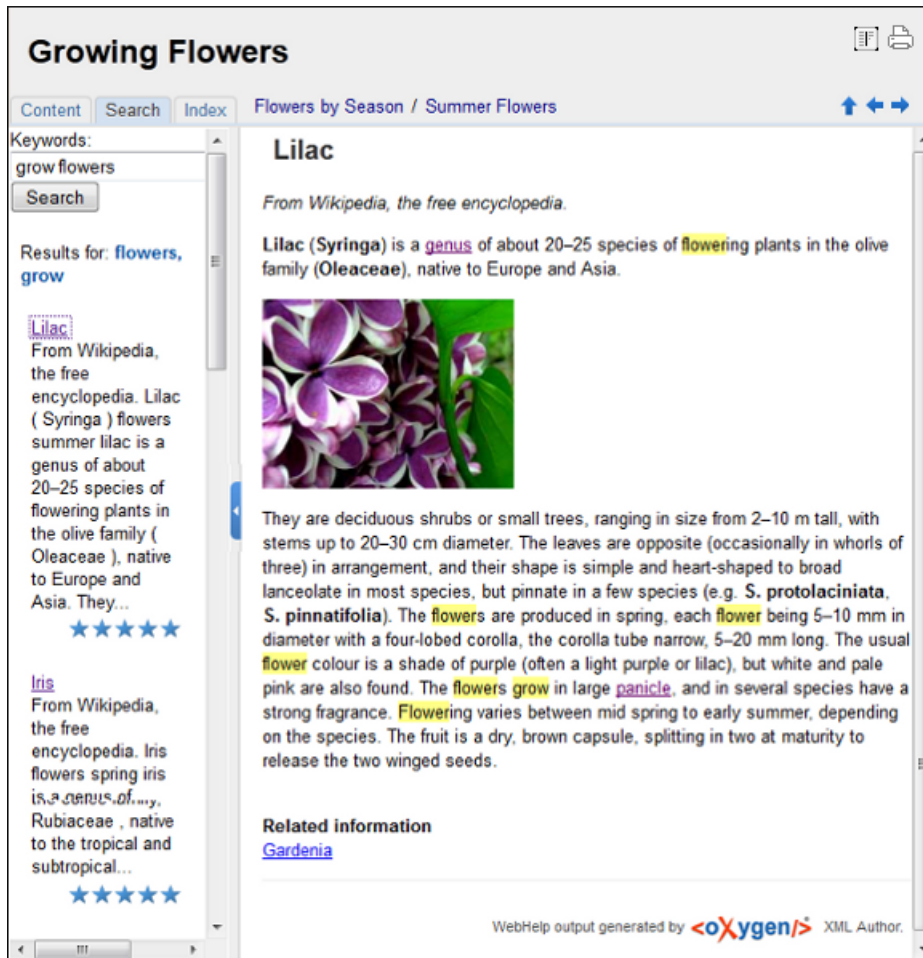
Figure 131: WebHelp Search with Stemming Enabled

Rules applied during search:

- the space character separates keywords. An expression like *grow flowers* counts as two separate keywords: *grow* and *flowers*;
- do not use quotes to perform exact search for multiple-word expressions. An expression like *"grow flowers"*, returns no results in our case, because it searches for two separate words: *"grow* and *flowers"* (note the quote signs attached to each word);
- `indexterm` and `keywords` DITA elements are an effective way to increase the ranking of a page. For example, content inside *keywords* elements weighs twice as much as content inside a *H1* HTML element;
- words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters, count as a single word;
- search for words containing three or more characters. Shorter words, like *to*, or *of* are ignored. This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

Note: This output format is compatible with the following browsers:

- Internet Explorer 8 or newer;
- Chrome;
- Firefox;
- Safari;
- Opera.

⚠️ **Important:** Due to some security restrictions in Google Chrome, WebHelp pages loaded from the local system (through URLs of the `file:///...` format) may not work properly. We recommend you to load WebHelp pages in Google Chrome only from a web server.

WebHelp Customizations

This section takes you through the customizations that you can make to the output of your WebHelp transformation.

To change the overall appearance of the WebHelp output, you can use the visual *WebHelp Skin Builder tool*, which require no knowledge of CSS language.

If you are familiar with CSS and you are not afraid of coding, the following topics describe how you can improve the appearance of the table of contents, add logo images in the title area, remove the navigation buttons, and add custom headers and footers. Also, an additional list of WebHelp related parameters is presented.

CSS Customizations

Adding your own CSS stylesheet, enables you to customize the WebHelp output. To do this, edit the transformation scenario and set the `args.css` parameter to point to your custom CSS document. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario runs.

Table of Contents Customization

The appearance of the selected item in the table of contest can be enhanced. To highlight the background of the selected item, go to the output folder of the WebHelp transformation, and locate the `toc.css` files in the **oxygen-webhelp** > **resources** > **skins** > **desktop** and **oxygen-webhelp** > **resources** > **skins** > **desktop-frames** folders. Open them, find the `menuItemSelected` class and change the value of the `background` property.

📝 **Note:** Also, you can overwrite the same value from your own CSS.

Adding a Logo Image in the Title Area

You are able to customize the title of your WebHelp output, using a custom CSS.

For example, to add a logo image before the title, use the following code:

```
h1:before {
    display:inline;
    content:url('../img/myLogoImage.gif');
}
```

In the example above, **myLogoImage.gif** is an image file which you place in the `[OXYGEN_DIR]\frameworks\dita\DITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\img` directory so it is copied automatically by the WebHelp transformation to the output directory.

Removing the Previous/Next Links from Each WebHelp Page

The **Previous** and **Next** links, that are created at the top area of each WebHelp page, can be hidden with the following CSS code:

```
.navparent, .navprev, .navnext {
    visibility:hidden;
}
```

ℹ️ **Tip:** Add the above code in a custom CSS stylesheet, set in the WebHelp transformation scenario using the **args.css** parameter.

Adding Custom Headers and Footers

In the transformation scenario, you can use the `args.hdr` and `args.ftr` parameters to point to resources which contain your custom HTML `<div>` blocks. These are included in the header and footer of each generated topic.

To hide the horizontal separator line between the content and footer, edit the DITA transformation scenario and configure the following parameters:

- `args.css` parameter to refer a CSS file containing the following CSS snippet:

```
.footer_separator {
  display:none;
}
```

- `args.copycss` parameter set to `true`.

Change numbering styles for ordered lists

Usually `ol` ordered lists are numbered in the XHTML output using numerals. If you want to change the numbering to alphabetical, do the following:

- define a custom `outputclass` value and set it as an attribute of the ordered list like in the following example:

```
<ol outputclass="number-alpha">
    <li>A</li>
    <li>B</li>
    <li>C</li>
</ol>
```

- add the following code snippet in a custom CSS file:

```
ol.number-alpha{
    list-style-type:lower-alpha;
}
```

- edit the DITA transformation scenario and configure the following parameters:

  - `args.css` parameter to refer the custom CSS file appended earlier;
  - `args.copycss` parameter set to `true`.

WebHelp Additional Parameters

The following additional WebHelp related parameters can be configured in the transformation scenario:

- `webhelp.copyright` - add a small copyright text which appears at the end of the table of contents;
- You can edit the `args.hide.parent.link` parameter to hide the **Parent**, **Next**, and **Previous** links;
- `args.xhtml.toc` - name of the table of contents file. Default setting is `toc.html`;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. Default setting is `false`;
- `clean.output` - deletes all files from the output folder before the transformation is performed. Only the `no` and `yes` values are valid. The default value is `no`.

How to Localize WebHelp Output

Static labels used in the WebHelp output are kept in translation files in the [OXYGEN_DIR]/frameworks/dita/DITA_OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization folder. The DITA-OT folder is by default `[OXYGEN_DIR]/frameworks/dita/DITA-OT`, or elsewhere if you are using a different DITA-OT distribution. Translation files have the *strings-lang1-lang2.xml* name format, where *lang1* and *lang2* are ISO language codes. For example, the US English text is kept in the *strings-en-us.xml* file.

Follow these steps to localize the interface of the WebHelp output (for simplicity sake, let us suppose you want to localize the WebHelp interface into Canadian French.):

1. Look for the *strings-fr-ca.xml* file in [OXYGEN_DIR]/frameworks/dita/DITA_OT/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization. If it does not exist, create one starting from *strings-en-us.xml*.
2. Translate all the labels from the above language file. Labels are stored in XML elements that have the following format: `<str name="Label name">Caption</str>`.
3. Edit the **DITA Map WebHelp/DITA Map WebHelp with Feedback** transformation scenario and set the args.default.language parameter to the code of the language you want to localize the interface into (in our case, it is *fr-ca*).

4. Run the transformation scenario to produce the WebHelp output.

WebHelp with Feedback Output Format

This section presents the feedback-enabled WebHelp system support.

Introduction

Oxygen XML Author offers support to transform DITA documents into feedback-enabled WebHelp systems.

WebHelp is a form of online help consisting of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser. It also provides table of contents and advanced search capabilities. The feedback system allows you to view discussion threads in a tree-like representation, post comments, reply to already posted comments, use stylized comments, and define administrators and moderators.

> **Note:** This output format is compatible with the following browsers:
>
> - Internet Explorer 8 or newer;
> - Chrome;
> - Firefox;
> - Safari;
> - Opera.

The DITA map WebHelp with Feedback transformation

To publish a DITA map to WebHelp with Feedback, use the **DITA Map WebHelp with Feedback** transformation. You can customize the out-of-the-box transformation by editing some of its parameters:

- `args.xhtml.toc` - name of the table of contents file. Default setting is `toc.html`;
- `use.stemming` - controls whether you want to include stemming search algorithms into the published output. Stemming is the process for reducing inflected (or sometimes derived) words to their `stem`, `base` or `root` form – generally a written word form. Default setting is `false`;
- `clean.output` - deletes all files from the output folder before the transformation is performed. Only the `no` and `yes` values are valid. The default value is `no`.

Before the transformation starts, enter the documentation product ID and the documentation version. After you run a **DITA Map WebHelp with Feedback** transformation, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions and deployment of the output.

To watch our video demonstration about the feedback-enabled WebHelp system, go to *http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html*.

> **Note:** This output format is compatible with the following browsers:
>
> - Internet Explorer 8 or newer;
> - Chrome;
> - Firefox;
> - Safari;
> - Opera.

> **Note:** In case you need to automate the transformation process and use it outside of Oxygen XML Author, you can use *the Oxygen XML WebHelp plugin*.

Installation

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Author requires the following system components:

- Apache Web Server running

- MySQL server running

- PHP Version 5.1.6 or later

- PHP MySQL Support

- Oxygen XML WebHelp system supports the following browsers:IE7+, Chrome 19+, Firefox 11+, Safari 5+, Opera 11+

Installation Instructions

Note: These instructions were written for XAMPP 1.7.7 with PHP 5.3.8 and for *phpMyAdmin* 3.4.5. Later versions of these packages may change the location or name of some options, however the following installation steps should remain valid and basically the same.

In case you have a web server configured with PHP, MySQL, you can deploy the WebHelp output directly. Otherwise, install XAMPP. XAMPP is a free and open source cross-platform web server solution stack package. It consists mainly of the Apache HTTP Server, MySQL database, and interpreters for scripts written in PHP.

Install XAMPP

1. Go to *https://www.apachefriends.org/download.html* and download XAMPP, for instance for a Windows system
2. Install it in `C:\xampp`
3. From the XAMPP control panel, start **MySQL**, and then **Apache**
4. Open `http://localhost/xampp/index.php` in your browser to check whether the HTTP server is working

Create the WebHelp Feedback database

The WebHelp system needs a database to store user details and the actual feedback they provide. The following procedure creates a database for the feedback system and a MySQL user with privileges on that database. The feedback system uses these credentials to connect to the database.

Use *phpMyAdmin* to create a database:

1. Type *localhost* in your browser
2. In the left area, select: *phpMyAdmin*
3. Click *Databases* (in the right frame) and then create a *database*. You can give any name you want to your database, for example *comments*
4. Create a user with connection privileges for this database. In the **SQL** tab, paste the following text:

```
  INSERT INTO `mysql`.`user`
(`Host`,`User`,`Password`,`Select_priv`,`Insert_priv`,`Update_priv`,`Delete_priv`,`Create_priv`,

`Drop_priv`,`Reload_priv`,`Shutdown_priv`,`Process_priv`,`File_priv`,`Grant_priv`,`References_priv`,`Index_priv`,`Alter_priv`,

`Show_db_priv`,`Super_priv`,`Create_tmp_table_priv`,`Lock_tables_priv`,`Execute_priv`,`Repl_slave_priv`,`Repl_client_priv`,

      `Create_view_priv`,
`Show_view_priv`,`Create_routine_priv`,`Alter_routine_priv`,`Create_user_priv`,`Event_priv`,`Trigger_priv`,

`Create_tablespace_priv`,`ssl_type`,`max_questions`,`max_updates`,`max_connections`,`max_user_connections`,`plugin`,

      `authentication_string`) VALUES ('localhost', 'user_name', PASSWORD('user_password'),
'Y','Y','Y','Y','Y','Y','Y', 'N','N','N',
      'N','Y','Y','Y','Y','N','Y','Y','Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y','', '0', '0',
'0', '0', '', '');
```

5. Change the *user_name* and the *user_password* values
6. Under *localhost* in the right frame click *Privileges* and then at the bottom of the page click the **reload the privileges** link

Deploying the WebHelp output

To deploy the WebHelp output, follow these steps:

1. Locate the directory of the HTML documents. Open `http://localhost/xampp/phpinfo.php` in your browser and see the value of the DOCUMENT_ROOT variable. In case you installed XAMPP in `C:\xampp`, the value of DOCUMENT_ROOT is `C:/xampp/htdocs`
2. Copy the transformation output folder in the `DOCUMENT_ROOT`
3. Rename it to a relevant name, for example, `webhelp_1`
4. Open `http://localhost/webhelp_1/`. You are redirected to `http://localhost/webhelp_1/oxygen-webhelp/install/`

   - Verify that the prerequisites are met
   - Press **Start Installation**
   - Configure the **Deployment Settings** section. Default values are provided, but you should adjust them as needed
   - Configure the **MySql Database Connection Settings** section. Use the details from the Create the WebHelp Feedback database section to fill-in the appropriate text boxes

     > ⛔ **Warning:** Checking the **Create new database structure** option will overwrite any existing data in the selected database, if it already exists.

   - If the **Create new database structure** option is checked, the **Create WebHelp Administrator Account** section becomes available. Here you can set the administrator account data. The administrator is able to moderate new posts and manage WebHelp users.

     The same database can be used to store comments for different WebHelp deployments. If a topic is available in more than one WebHelp deployments and there are comments associated with it, you can choose to display the comments in all deployments that share the database. To do this, enable the **Display comments from other products** option. In the **Display comments from** section a list with the deployments sharing the same database is displayed. Select the deployments allowed to share common feedback.

     > 📝 **Note:** You can restrict the displayed comments of a product depending on its version. In case you have two products that use the same database and you restrict one of them to display comments starting from a certain version, the comments of the other product are also displayed from the specified version onwards.

   - Press **Next Step**
   - Remove the installation folder from your web server
   - Click the link pointing to the index of the documentation, or visit: `http://localhost/webhelp_1/`

To test your system, create a user and post a comment. Check if the notification emails are delivered to your inbox.

> 📝 **Note:** To read debug messages generated by the system:
>
> 1. Enable *JScript* logging:
>
>   - Either: open the `log.js` file, locate the `var log= new Log(Level.NONE);` line, and change the logging level to: `Level.INFO`, `Level.DEBUG`, `Level.WARN`, or `Level.ERROR`
>   - or append `?log=true` to the WebHelp URL
>
> - Inspect the PHP and Apache server log files.

Layout of the Feedback-Enabled WebHelp System

The layout of the feedback-enabled WebHelp system resembles the layout of the basic WebHelp, the left frame remaining the same. However, the bottom of the right frame contains a **comments** bar. Select **Log in** from this bar to authenticate as a user of the WebHelp system. In case you do not have a user name, complete the fields in the dialog box that opens to create a user. Under the **comments** bar, you can click the **Add New Comment** button to add a comment whether you are logged in or not. The tabs in the left frame have the same functionality as the Content, Search, and Index tab of the basic WebHelp.

Note: You can choose to enhance the appearance of the selected item in the table of contents. The *WebHelp customization* topic contains more details about this.

Figure 132: The layout of the Feedback-Enabled WebHelp System

After you log in, your name and user name are displayed in the **Comments** bar together with the **Log of** and **Edit** buttons. Click the **Edit** button to open the **User Profile** dialog. In this dialog you can customize the following options:

- **Your Name** - you can use this field to edit the initial name that you used to create your user profile;
- **Your e-mail address** - you can use this field to edit the initial e-mail address that you used to create your profile;
- When to receive an e-mail:
  - when a comment is left on a page that you commented on;
  - when a comment is left on any topic in the Help system ;
  - when a reply is left to one of my comments.

- **New Password** - allows you to enter a new password for your user account.

    Note: The **Current Password** field from the top of the **User Profile** is mandatory in case you want to save the changes you make.

Advanced Customization and Management

Apart from the options available for a regular user, you can also use the administrative page for advanced customization and management. As an administrator, you have full access to all the features of the feedback-enabled WebHelp system. To access the administrative page, select **Admin Panel** from the **Comments** bar.

User Name	Name	Level	Company	E-Mail	Date	Web Help Notification	Reply Notification	Page Notification	Status
admin	Administrator	admin	NA	john@oxygenxml.com	2012-06-25 07:04:13	yes	yes	yes	validated
oXygen		moderator	noCompany	chris@oxygenxml.com	2012-06-26 01:37:07	yes	no	no	validated

Figure 133: The Administrative Page

This page allows you to view all posts, export comments and set the version of the WebHelp system. You can also view the details of each user and search through these details using the **Search User Information** filter.

The upper part of the page contains the following actions:

- **Delete Orphan Comments** - deletes comments associated with topics that are no longer available
- **Delete Pending Users** - deletes all unconfirmed users that registered more than a week ago
- **View All Posts** - allows you to view all posts associated with a product and version
- **Export Comments** - allows you to export in XML format all posts associated with a product and version
- **Set Version** - use this action to display comments starting from a particular version

To edit the details of a user, click the corresponding row. Use the **Edit User** dialog to customize all the information associated with an user:

- **Name** - The user's full name
- **Level** - Use this field to modify the privilege level of the currently edited user. You can choose from:

  - **User** - regular user, able to post comments and receive e-mail notifications
  - **Moderator** - in addition to the regular **User** rights, this type of user has access to the **Admin Panel**. In the administrative page a moderator can view, delete, export comments and set the version of the feedback-enabled WebHelp system.
  - **Admin** - full administrative privileges. Can manage WebHelp-specific settings, users and their comments.

- **Company** - User's organization name
- **E-mail** - User's contact e-mail address. This is also the address where the WebHelp system sends notifications:

  - **WebHelp Notification** - when enabled, the user receives notifications when comments are posted anywhere in the feedback-enabled WebHelp system
  - **Reply Notification** - when enabled, the user receives notifications when comments are posted as a reply to one of his or hers comments
  - **Page Notification** - when enabled, the user receives notifications when comments are posted on a topic where he or she posted a comment

- **Date** - User registration date
- **Status** - Specifies the status of the currently edited user:

  - **Created** - the user is created but does not have any rights over the feedback-enabled WebHelp system
  - **Validated** - the user is able to use the feedback-enabled WebHelp system
  - **Suspended** - the user has no rights over the feedback-enabled WebHelp system

Localizing the Email Notifications for DITA to WebHelp with Feedback Transformation Scenario

The WebHelp with Feedback system uses emails to notify users when comments are posted. These emails are based on templates stored in the WebHelp directory. The default messages are in English, French, German and Japanese and they are stored in the WebHelp directory. For example, the English messages are stored in this directory:

`[OXYGEN_DIR]\frameworks\dita\DITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\en`

We'll suppose that you want to localize the emails into Dutch. Follow these steps:

*   create the following directory:

    [OXYGEN_DIR]\frameworks\dita\DITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl

*   copy all English template files from
    [OXYGEN_DIR]\frameworks\dita\DITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\en
    and paste them into the directory you just created
*   edit the HTML files from the
    [OXYGEN_DIR]\frameworks\dita\DITA-OT\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl
    directory and translate the content into Dutch
*   start Oxygen XML Author and edit the *WebHelp with Feedback* transformation scenario
*   in the **Parameters** tab look for the `args.default.language` parameter and set its value to the appropriate
    language code. In our example, use the value `nl` for Dutch

    > **Note:** If you set the parameter to a value such as `LanguageCode-CountryCode` (for example, `en-us`),
    > the transformation scenario will only use the language code

*   execute the transformation scenario to obtain the *WebHelp with Feedback* output

Mobile WebHelp Output Format

To further improve its ability to create online documentation, Oxygen XML Author offers support to transform DITA
documents into mobile WebHelp systems. This feature generates an output that works on multiple platforms (Android,
iOS, BlackBerry, Windows Mobile) and is specially designed for mobile devices. All the specific touch screen gestures
are supported. The functionality of the desktop WebHelp layout is preserved, offering table of contents, search capabilities,
and index navigation, organized in an intuitive layout.
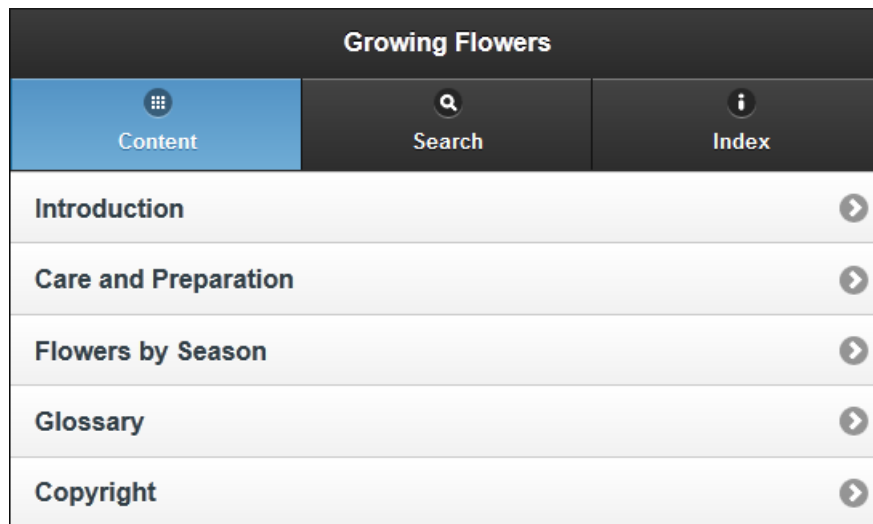
Figure 134: Mobile WebHelp

To generate a mobile WebHelp system from your DITA MAP, go to the **DITA Maps Manager** view, click
Configure Transformation Scenarios() and select the **DITA Map WebHelp - Mobile** transformation scenario. Click
Apply associated. Once Oxygen XML Author finishes the transformation process, the output is opened in your default
browser automatically.

To customize the TOC section of the output, you need to alter the
`oxygen-webhelp\resources\skins\mobile\toc.css` stylesheet.

DITA Map Templates

The default templates available for DITA maps are stored in `[OXYGEN_DIR]/frameworks/dita/templates/map` folder. They can be used for easily creating 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

DITA for Publishers Map specialization templates:

- **D4P Map** - New DITA for Publishers Map;
- **D4P Pub-component-map** - New DITA for Publishers pub-component-map;
- **D4P Pubmap** - New DITA for Publishers pubmap.

The XHTML Document Type

The Extensible HyperText Markup Language, or XHTML, is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax.

A file is considered to be a XHTML document when the root element name is a `html`.

The schema used for these documents is located in `[OXYGEN_DIR]/frameworks/xhtml/dtd/xhtml1-strict.dtd`, where *${frameworks}* is a subdirectory of the Oxygen XML Author install directory.

The default CSS options for the XHTML document type are set to merge the CSS stylesheets specified in the document with the CSS stylesheets defined in the XHTML document type.

The CSS file used for rendering XHTML content is located in `[OXYGEN_DIR]/frameworks/xhtml/css/xhtml.css`.

There are three default catalogs for XHTML document type:

- `[OXYGEN_DIR]/frameworks/xhtml/dtd/xhtmlcatalog.xml`
- `[OXYGEN_DIR]/frameworks/xhtml11/dtd/xhtmlcatalog.xml`
- `[OXYGEN_DIR]/frameworks/xhtml11/schema/xhtmlcatalog.xml`

XHTML Author Extensions

Specific actions are:

B **Bold**

Changes the style of the selected text to `bold` by surrounding it with `b` tag;

I **Italic**

Changes the style of the selected text to `italic` by surrounding it with `i` tag;

U **Underline**

Changes the style of the selected text to `underline` by surrounding it with `u` tag.

> **Note:**
>
> **Bold**, **Italic**, and **Underline** are toggle actions.
>
> For all of the above actions, if there is no selection, then a new specific tag will be inserted. These actions are available in any document context.

H **Headings**

Groups actions for inserting h1, h2, h3, h4, h5, h6 elements;

Insert image reference

Inserts a graphic object at the caret position. This is done by inserting an img element regardless of the current context. The following graphical formats are supported: GIF, JPG, JPEG, BMP, PNG, SVG.

Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also inserted automatically by default.

Insert an unordered list at the caret position

Inserts an itemized list. A child list item is also inserted automatically by default.

Insert a step or list Item

Inserts a new list item in any of the above list types.

Insert Definition List

Inserts a definition list (dl element) with one list item (a dt child element and a dd child element).

Insert Table

Opens a dialog that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

Insert Row

Inserts a new table row with empty cells. This action is available when the caret is positioned inside a table.

Insert Column

Inserts a new table column with empty cells after the current column. This action is available when the caret is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the caret is positioned between two cells, Oxygen XML Author a new cell at caret position. If the caret is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at caret position.

Delete Row

Deletes the table row located at caret position.

Insert Row Above

Inserts a row above the current one.

Insert Row Below

Inserts a row below the current one.

Insert Column Before

Inserts a column before the current one.

Insert Column After

Inserts a column after the current one.

Join Row Cells

Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.

⊞ Join Cell Above

Joins the content of the cell from the current caret position with the content of the cell above it. This action works only if both cells have the same column span.

⊞ Join Cell Below

Joins the content of the cell from the current caret position with the content of the cell below it. This action works only if both cells have the same column span.

> **Note:** When you use ⊞ **Join Cell Above** and ⊞ **Join Cell Below**, Oxygen XML Author deletes the source row is case it remains empty. The cells that span over multiple rows are also updated.

⬓ Split Cell To The Left

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

⬓ Split Cell To The Right

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

⬒ Split Cell Above

Splits the cell from current caret position in two cells, inserting a new empty table cell above. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

⬓ Split Cell Below

Splits the cell from current caret position in two, inserting a new empty table cell below. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

All actions described above are available in the contextual menu, the **XHTML** submenu of the main menu and in the **Author custom actions** toolbar.

A drag and drop with a file from *the Project view* or from *the DITA Maps Manager view* to a DITA topic document that is edited in Author mode will create a link to the dragged file (the xref DITA element with the **href** attribute) at the drop location. A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on Mac OS X, etc) will insert an image element (the image DITA element with the href attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

XHTML Transformation Scenarios

The following default transformation scenarios are available for XHTML:

- **XHTML to DITA concept** - Converts an XHTML document to a DITA concept document
- **XHTML to DITA reference** - Converts an XHTML document to a DITA reference document
- **XHTML to DITA task** - Converts an XHTML document to a DITA task document
- **XHTML to DITA topic** - Converts an XHTML document to a DITA topic document

XHTML Templates

Default templates are available for XHTML. They are stored in [OXYGEN_DIR]/frameworks/xhtml/templates folder and they can be used for easily creating basic XHTML documents.

Here are some of the XHTML templates available when creating *new documents from templates*.

- **XHTML - 1.0 Strict** - New Strict XHTML 1.0
- **XHTML - 1.0 Transitional** - New Transitional XHTML 1.0

- **XHTML - 1.1 DTD Based** - New DTD based XHTML 1.1
- **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
`[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/brown_odds.rng`.

The CSS file used for rendering TEI ODD content is located in
`[OXYGEN_DIR]/frameworks/tei/xml/tei/css/tei_oxygen_odd.css`.

There are two default catalogs for TEI ODD document type:

- `[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/tei/schema/catalog.xml`

To watch our video demonstration about TEI editing, go to *http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html*.

TEI ODD Author Extensions

The specific actions for TEI ODD documents are:

B **Bold**

Changes the style of the selected text to `bold` by surrounding it with `hi` tag and setting the `rend` attribute to `bold`;

I **Italic**

Changes the style of the selected text to `italic` by surrounding it with `hi` tag and setting the `rend` attribute to `italic`;

U **Underline**

Changes the style of the selected text to `underline` by surrounding it with `hi` tag and setting the `rend` attribute to `ul`;

> **Note:**
>
> **Bold**, **Italic**, and **Underline** are toggle actions.
>
> For all of the above actions, if there is no selection, then a new specific tag will be inserted. These actions are available in any document context.

§ **Insert Section**

Inserts a new section / subsection, depending on the current context. For example if the current context is `div1` then a `div2` will be inserted and so on;

Insert image reference

inserts an image reference at the caret position;

Insert Table

Opens a dialog that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

Insert Row

Inserts a new table row with empty cells. This action is available when the caret is positioned inside a table.

Insert Column

Inserts a new table column with empty cells after the current column. This action is available when the caret is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the caret is positioned between two cells,Oxygen XML Author a new cell at caret position. If the caret is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at caret position.

Delete Row

Deletes the table row located at caret position.

Insert Row Above

Inserts a row above the current one.

Insert Row Below

Inserts a row below the current one.

Insert Column Before

Inserts a column before the current one.

Insert Column After

Inserts a column after the current one.

Join Row Cells

Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current caret position with the content of the cell above it. This action works only if both cells have the same column span.

Join Cell Below

Joins the content of the cell from the current caret position with the content of the cell below it. This action works only if both cells have the same column span.

> **Note:** When you use **Join Cell Above** and **Join Cell Below**, Oxygen XML Author deletes the source row is case it remains empty. The cells that span over multiple rows are also updated.

Split Cell To The Left

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current caret position in two cells, inserting a new empty table cell above. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current caret position in two, inserting a new empty table cell below. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also inserted automatically by default.

Insert an unordered list at the caret position

Inserts an itemized list. A child list item is also inserted automatically by default.

Insert a step or list Item

Inserts a new list item in any of the above list types.

Generate IDs

This action generates and sets unique IDs for:

- the element at caret position
- all top-level elements found in the current selection. Additionally, if the selection contains elements from the **TEI** > **ID Options** list, they will all receive an unique ID

> **Note:** IDs already set are preserved.

The action is available both in the contextual menu and in the **TEI** main menu.

ID Options

Action available in the **TEI** main menu, allows you to specify the elements for which Oxygen XML Author generates an unique ID if the **Auto generate IDs for elements** option is enabled. The configurable ID value pattern can accept most of the application supported *editor variables*.

To keep an already set element ID when copying content in the same document, make sure the **Remove IDs 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 (Command+Shift+G on OS X)** and can be changed in the **DITA Topic** document type.

All actions described above are available in the contextual menu, the **TEI ODD** submenu of the main menu or in the **Author custom actions** toolbar.

A drag and drop with a file from *the Project view* or from *the DITA Maps Manager view* to a **TEI ODD** document that is edited in Author mode will create a link to the dragged file (the ptr element with the target attribute) at the drop location.

TEI ODD Transformation Scenarios

The following default transformations are available:

- **TEI ODD XHTML** - Transforms a TEI ODD document into an XHTML document
- **TEI ODD PDF** - Transforms a TEI ODD document into a PDF document using the Apache FOP engine
- **TEI ODD EPUB** - Transforms a TEI ODD document into an EPUB document
- **TEI ODD DOCX** - Transforms a TEI ODD document into a DOCX document
- **TEI ODD ODT** - Transforms a TEI ODD document into an ODT document

- **TEI ODD RelaxNG XML** - Transforms a TEI ODD document into a RelaxNG XML document
- **TEI ODD to DTD** - Transforms a TEI ODD document into a DTD document
- **TEI ODD to XML Schema** - Transforms a TEI ODD document into an XML Schema document
- **TEI ODD to RelaxNG Compact** - Transforms a TEI ODD document into an RelaxNG Compact document

TEI ODD Templates

There is only one default template which is stored in the `[OXYGEN_DIR]/frameworks/tei/templates/TEI ODD` folder and can be used for easily creating a basic TEI ODD document. This template is available when creating *new documents from templates*.

- **TEI ODD** - New TEI ODD document

The TEI P4 Document Type

The **Text Encoding Initiative (TEI) Guidelines** is an international and interdisciplinary standard that enables libraries, museums, publishers, and individual scholars to represent a variety of literary and linguistic texts for online research, teaching, and preservation.

A file is considered to be a TEI P4 document when 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 `[OXYGEN_DIR]/frameworks/tei/tei2xml.dtd`.

The CSS file used for rendering TEI P4 content is located in `[OXYGEN_DIR]/frameworks/tei/xml/tei/css/tei_oxygen.css`.

There are two default catalogs for TEI P4 document type:

- `[OXYGEN_DIR]/frameworks/tei/xml/teip4/schema/dtd/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/teip4/custom/schema/dtd/catalog.xml`

To watch our video demonstration about TEI editing, go to *http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html*.

TEI P4 Author Extensions

The specific actions for TEI P4 documents are:

B Bold

Changes the style of the selected text to `bold` by surrounding it with `hi` tag and setting the `rend` attribute to `bold`.

I Italic

Changes the style of the selected text to `italic` by surrounding it with `hi` tag and setting the `rend` attribute to `italic`.

U Underline

Changes the style of the selected text to `underline` by surrounding it with `hi` tag and setting the `rend` attribute to `ul`.

> **Note:**
>
> **Bold**, **Italic**, and **Underline** are toggle actions.
>
> For all of the above actions, if there is no selection, then a new specific tag will be inserted. These actions are available in any document context.

Browse reference manual

Opens in your web browser of choice a reference to the documentation of the XML element closest to the caret position. When editing DITA documents, this action is available in the contextual menu of the editing area (under the **About Element** sub-menu) and in the documentation tip displayed by the **Content Completion Assistant**.

§   **Insert Section**

Inserts a new section / subsection, depending on the current context. For example if the current context is `div1` then a `div2` will be inserted and so on.

Insert image reference

inserts an image reference at the caret position;

Insert Table

Opens a dialog that allows you to configure and insert a table. You can generate a header and footer, set the number of rows and columns of the table and decide how the table is framed.

Insert Row

Inserts a new table row with empty cells. This action is available when the caret is positioned inside a table.

Insert Column

Inserts a new table column with empty cells after the current column. This action is available when the caret is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the caret is positioned between two cells,Oxygen XML Author a new cell at caret position. If the caret is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at caret position.

Delete Row

Deletes the table row located at caret position.

Insert Row Above

Inserts a row above the current one.

Insert Row Below

Inserts a row below the current one.

Insert Column Before

Inserts a column before the current one.

Insert Column After

Inserts a column after the current one.

Join Row Cells

Joins the content of the selected cells. The operation is available if the selected cells are from the same row and they have the same row span. The action is also available when the selection is missing, but the caret is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current caret position with the content of the cell above it. This action works only if both cells have the same column span.

Join Cell Below

Joins the content of the cell from the current caret position with the content of the cell below it. This action works only if both cells have the same column span.

> **Note:** When you use ⬆ **Join Cell Above** and ⬇ **Join Cell Below**, Oxygen XML Author deletes the source row is case it remains empty. The cells that span over multiple rows are also updated.

Split Cell To The Left

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the left. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current caret position in two cells, inserting a new empty table cell to the right. This action works only if the current cell spans over more than one column.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current caret position in two cells, inserting a new empty table cell above. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current caret position in two, inserting a new empty table cell below. This action works only if the current cell spans over more than one row.Oxygen XML Author decreases the column span of the source cell with one.

Insert an ordered list at the caret position

Inserts an ordered list. A child list item is also inserted automatically by default.

Insert an unordered list at the caret position

Inserts an itemized list. A child list item is also inserted automatically by default.

Insert a step or list Item

Inserts a new list item in any of the above list types.

Generate IDs

This action generates and sets unique IDs for:

- the element at caret position
- all top-level elements found in the current selection. Additionally, if the selection contains elements from the **TEI** > **ID Options** list, they will all receive an unique ID

> **Note:** IDs already set are preserved.

The action is available both in the contextual menu and in the **TEI** main menu.

ID Options

Action available in the **TEI** main menu, allows you to specify the elements for which Oxygen XML Author generates an unique ID if the **Auto generate IDs for elements** option is enabled. The configurable ID value pattern can accept most of the application supported *editor variables*.

To keep an already set element ID when copying content in the same document, make sure the **Remove IDs 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 (Command+Shift+G on OS X)** and can be changed in the **DITA Topic** document type.

All actions described above are available in the contextual menu, the **TEI P4** submenu of the main menu or in the **Author custom actions** toolbar.

A drag and drop with a file from *the **Project** view* or from *the **DITA Maps Manager** view* to a TEI P4 document that is edited in Author mode will create a link to the dragged file (the `ptr` element with the `target` attribute) at the drop location.

TEI P4 Transformation Scenarios

The following default transformations are available:

- **TEI HTML** - Transforms a TEI document into a HTML document;
- **TEI P4 -> TEI P5 Conversion** - Convert a TEI P4 document into a TEI P5 document;
- **TEI PDF** - Transforms a TEI document into a PDF document using the Apache FOP engine.

TEI P4 Templates

The default templates are stored in `[OXYGEN_DIR]/frameworks/tei/templates/TEI P4` folder and they can be used for easily creating basic TEI P4 documents. These templates are available when creating *new documents from templates*.

- **TEI P4 - Lite** - New TEI P4 Lite
- **TEI P4 - New Document** - New TEI P4 standard document

Customization of TEI Frameworks Using the Latest Sources

The *TEI P4* and *TEI P5* frameworks are available as a public project at the following SVN repository:

```
https://oxygen-tei.googlecode.com/svn/trunk/
```

This project is the base for customizing a TEI framework.

1. Check out the project on a local computer from the SVN repository.

   This action is done with an SVN client application that creates a working copy of the SVN repository on a local computer.

2. Customize the TEI framework in Oxygen XML Author.

   a) Set the Oxygen XML Author `frameworks` folder to the `oxygen/frameworks` subfolder of the folder of the SVN working copy.

   *Open the **Preferences** dialog* and go to **Global** and set the path of the SVN working copy in the option **Use custom frameworks**.

   b) *Open the **Preferences** dialog* , go to **Document Type Association/Locations** and select **Custom**.

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 XML Author install folder.

The TEI P5 Document Type

The TEI P5 document type is similar with the TEI P4 one, with the following exceptions:

- A file is considered to be a TEI P5 document when the namespace is *http://www.tei-c.org/ns/1.0*.
- The schema is located in `[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/tei_allPlus.rng`.

- A drag and drop with an image file from the default file system application (Windows Explorer on Windows, Finder on OS X, etc) will insert an image element (the graphic DITA element with the url attribute) with the location of the dragged file at the drop location, like the **Insert Graphic** toolbar action.

To watch our video demonstration about TEI editing, go to *http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html*.

TEI P5 Transformation Scenarios

The following default transformations are available:

- **TEI P5 XHTML** - transforms a TEI P5 document into a XHTML document;
- **TEI P5 PDF** - transforms a TEI P5 document into a PDF document using the Apache FOP engine;
- **TEI EPUB** - transforms a TEI P5 document into an EPUB output. The EPUB output will contain any images referenced in the TEI XML document;
- **TEI DOCX** - transforms a TEI P5 document into a DOCX (OOXML) document. The DOCX document will contain any images referenced in the TEI XML document;
- **TEI ODT** - transforms a TEI P5 document into an ODT (ODF) document. The ODT document will contain any images referenced in the TEI XML document.

TEI P5 Templates

The default templates are stored in [OXYGEN_DIR]/frameworks/tei/templates/TEI P5 folder and they can be used for easily creating basic TEI P5 documents. These templates are available when creating *new documents from templates*:

- **TEI P5 - All** - New TEI P5 All;
- **TEI P5 - Bare** - New TEI P5 Bare;
- **TEI P5 - Lite** - New TEI P5 Lite;
- **TEI P5 - Math** - New TEI P5 Math;
- **TEI P5 - Speech** - New TEI P5 Speech;
- **TEI P5 - SVG** - New TEI P5 with SVG extensions;
- **TEI P5 - XInclude** - New TEI P5 XInclude aware.

Customization of TEI Frameworks Using the Latest Sources

The *TEI P4* and *TEI P5* frameworks are available as a public project at the following SVN repository:

```
https://oxygen-tei.googlecode.com/svn/trunk/
```

This project is the base for customizing a TEI framework.

1. Check out the project on a local computer from the SVN repository.

   This action is done with an SVN client application that creates a working copy of the SVN repository on a local computer.

2. Customize the TEI framework in Oxygen XML Author.

   a) Set the Oxygen XML Author frameworks folder to the oxygen/frameworks subfolder of the folder of the SVN working copy.

      Open the Preferences dialog and go to **Global** and set the path of the SVN working copy in the option **Use custom frameworks**.

   b) *Open the Preferences dialog* , go to **Document Type Association/Locations** and select **Custom**.

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 XML Author install folder.

Customization of TEI Frameworks Using the Compiled Sources

The following procedure describes how to update to the latest stable version of TEI Schema and TEI XSL, already integrated in the TEI framework for Oxygen XML Author.

1. Go to *https://code.google.com/p/oxygen-tei/*;
2. Go to **Downloads**;
3. Download the latest uploaded `.zip` file;
4. Unpack the `.zip` file and copy its content in the Oxygen XML Author `frameworks` folder.

The EPUB Document Type

Three distinct frameworks support the EPUB document type:

- **NCX** - A declarative global navigation definition.
- **OCF** - The Open Container Format(OCF) defines a mechanism by which all components of an Open Publication Structure(OPS) can be combined into a single file-system entity.
- **OPF**: - The Open Packaging Format(OPF) defines the mechanism by which all components of a published work conforming to the Open Publication Structure(OPS) standard including metadata, reading order and navigational information are packaged into an OPS Publication.

    Note: Oxygen XML Author supports both OPF 2.0 and OPF 3.0.

Chapter

9

Authoring Customization Guide

Topics:

The Author mode editor of Oxygen XML Author was designed to provide a friendly user interface to editing XML documents. Author combines the power of source editing with the intuitive interface of a word processor. You can customize the author mode editor to support new custom XML formats or to change how standard XML formats are edited.
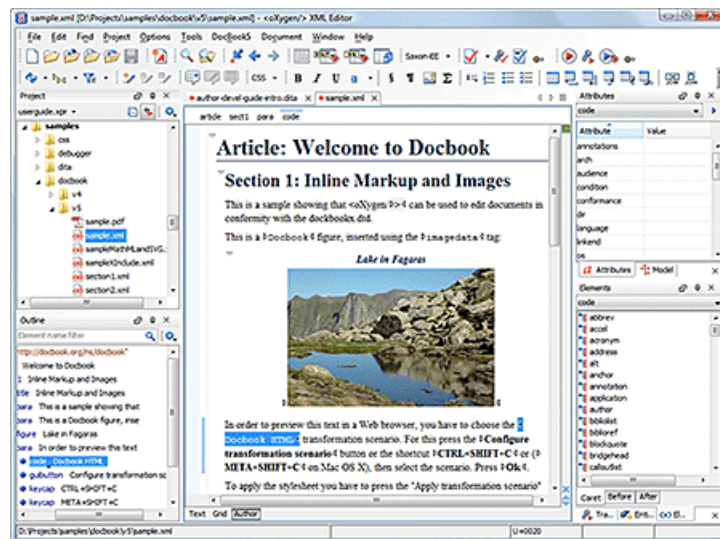
Figure 135: Oxygen XML Author Author Visual Editor

Although Oxygen XML Author 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 CSS stylesheets, 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 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;
}
```
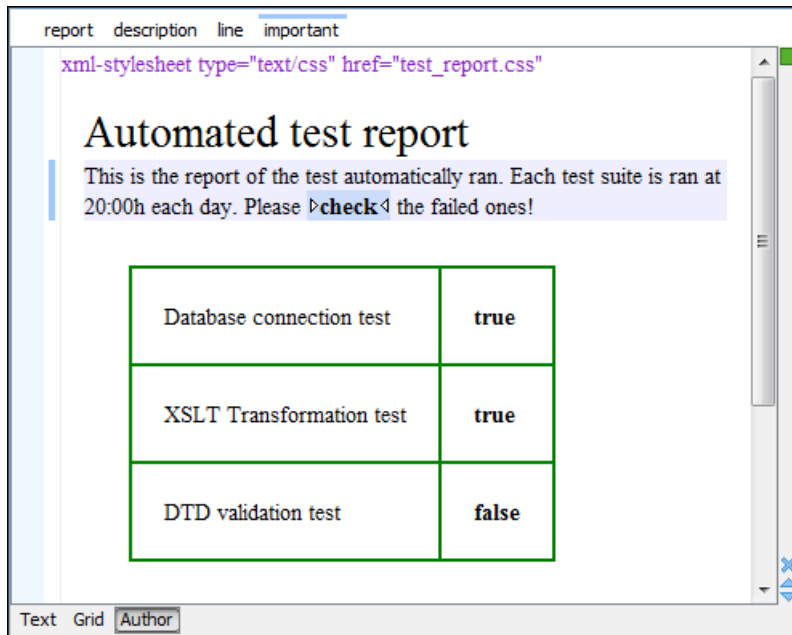
Figure 136: A report rendered in Author mode

📝 **Note:** You can edit attributes in-place in the Author mode using *form-based controls*.

The XML Instance Template

Based on the XML Schema and the CSS file the Author can help the content author in loading, editing and validating the test reports. An XML file template must be created, a kind of skeleton, that the users can use as a starting point for creating new test reports. The template must be generic enough and refer the XML Schema file and the CSS stylesheet.

This is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="test_report.css"?>
<report xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="test_report.xsd">
  <title>Automated test report</title>
  <description>
    <line>This is the report of the test automatically ran. Each test suite is ran at 20:00h each
      day. Please <important>check</important> the failed ones!</line>
  </description>
  <results>
    <entry>
      <test_name>Database connection test</test_name>
      <passed>true</passed>
    </entry>
    <entry>
      <test_name>XSLT Transformation test</test_name>
      <passed>true</passed>
    </entry>
    <entry>
      <test_name>DTD validation test</test_name>
      <passed>false</passed>
    </entry>
  </results>
</report>
```

The processing instruction `xml-stylesheet` associates the CSS stylesheet to the XML file. The href pseudo attribute contains the URI reference to the stylesheet file. In our case the CSS is in the same directory as the XML file.

The next step is to place the XSD file and the CSS file on a web server and modify the template to use the HTTP URLs, like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
      href="http://www.mysite.com/reports/test_report.css"?>
```

```
<report xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation=
        "http://www.mysite.com/reports/test_report.xsd">
    <title>Test report title</title>
    <description>
.......
```

The alternative is to create an archive containing the `test_report.xml`, `test_report.css` and `test_report.xsd` and send it to the content authors.

Advanced Customization Tutorial - Document Type Associations

Oxygen XML Author supports individual document types and classes of document types through frameworks. A framework associates a document type or a class of documents with CSS stylesheets, validation schemas, catalog files, new files templates, transformation scenarios and custom actions.

In this tutorial, we create a framework for a set of documents. As an example, we create a light documentation framework (similar to DocBook), then we set up a complete customization of the **Author** mode.

You can find the samples used in this tutorial in the *Example Files Listings* and the complete source code in the Simple Documentation Framework project. This project is included in the *Oxygen SDK*, available as a Maven archetype. More information about the Oxygen SDK setup can be found *here*.

> **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Author website*. Also it is available in the *Oxygen SDK Maven Project*.

Document Type Settings

To add or edit a *Document Type Association* *open the Preferences dialog* and go to **Document Type Association**. All the changes can be made in the *Document type* edit dialog.
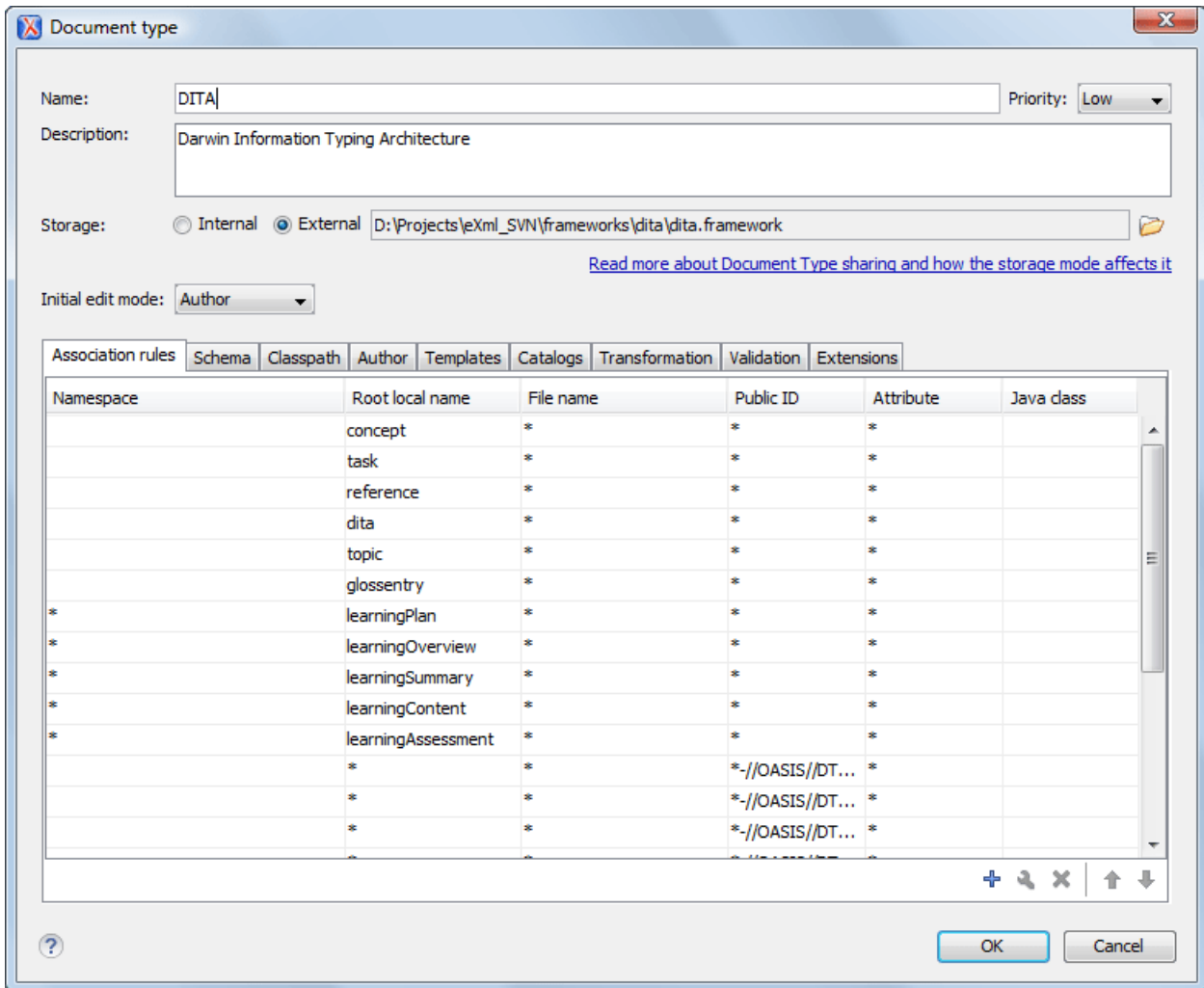
Figure 137: The Document Type

You can specify the following properties for a document type:

- **Name** - The name of the document type.
- **Priority** - When multiple document types match the same document, the priority determines the order in which they are applied. It can be one of: Lowest, Low, Normal, High, Highest. The predefined document types that are already configured when the application is installed on the computer have the default Low priority.

    Note: The frameworks having the same priority are alphabetically sorted.

- **Description** - The document type description displayed as a tool tip 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 subdirectory of the current `frameworks` directory. If you select the **Internal** storage option, the document type data is saved in the current `.xpr` Oxygen XML Author project file (for Project-level Document Type Association Options) or in the Oxygen XML Author internal options (for Global-level Document Type Association Options). You can change the Document Type Association Options level in the *Document Type Association options*.
- **Initial edit mode** - Allows you to select the initial editing mode (**Editor specific**, **Text**, **Author, Grid** and **Design** (available only for the W3C XML Schema editor)) for this document type. If the **Editor specific** option is selected, the initial edit mode is determined depending on the editor type. You can find the mapping between editors and edit modes in the *Edit modes preferences page.* You can decide to impose an initial mode for opening files which match the association rules of the document type. For example if the files are usually edited in the *Author* mode you can set it in the **Initial edit mode** combo box.

Note: You can also customize the initial mode for a document type in the **Edit modes** preferences page. *Open the Preferences dialog* and go to **Editor** > **Edit modes**.

You can specify the association rules used for determining a document type for an opened XML document. A rule can define one or more conditions. All conditions need to be fulfilled in order for a specific rule to be chosen. Conditions can specify:

- **Namespace** - The namespace of the document that matches the document type.
- **Root local name of document** - The local name of the document that matches the document type.
- **File name** - The file name (including the extension) of the document that matches the document type.
- **Public ID** (for DTDs) - The PUBLIC identifier of the document that matches the document type.
- **Attribute** - This field allows you to associate a document type depending on a certain value of the attribute in the root.
- **Java class** - Name of Java class that is called for finding if the document type should be used for an XML document. Java class must implement `ro.sync.ecss.extensions.api.DocumentTypeCustomRuleMatcher` interface from *Author API*.

In the **Schema** tab, you can specify the type and URI of schema used for validation and content completion of all documents from the document type, when there is no schema detected in the document.

You can choose one of the following schema types:

- DTD;
- Relax NG schema (XML syntax);
- Relax NG schema (XML syntax) + Schematron;
- Relax NG schema (compact syntax);
- XML Schema;
- XML Schema + Schematron rules;
- NVDL schema.

Configuring Actions, Menus and Toolbars

You can change the Author toolbars and menus to gain a productive editing experience. You can create a set of actions that are specific to a document type, using the **Document Type** dialog.

In the example with the `sdf` framework, you created the stylesheet and the validation schema. Now let's add some actions to insert a `section` and a `table`. To add a new action, follow the procedure:

1. *Open the Preferences dialog* and go to **Document Types Association** and click the framework for which you want to create an action.
2. Click **Edit** and in the **Document Type** dialog go to the **Author** tab, then go to **Actions**.
3. Click the **＋ New** and use the *Action dialog***Action** dialog to create an action.

Creating the Insert Section Action

This section presents all the steps that you need to follow, to define the **Insert Section** action. We assume the icon files

§ (`Section16.png`) for the menu item and § (`Section20.png`) for the toolbar, are already available. Although you could use the same icon size for both menu and toolbar, usually the icons from the toolbars are larger than the ones found in the menus. These files should be placed in the `frameworks/sdf` directory.
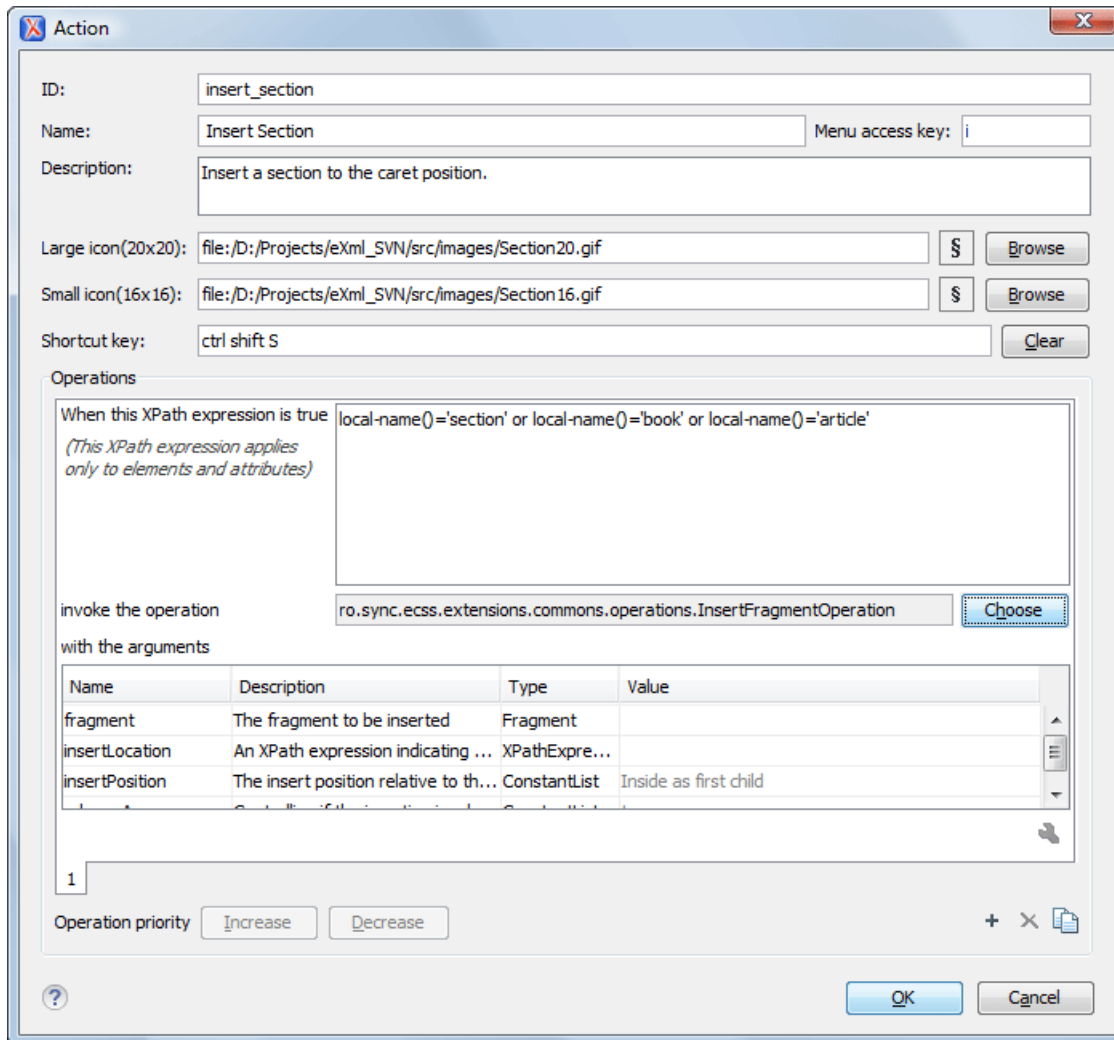
Figure 138: The Action Dialog

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** keys 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 ${framework}* to make the image relative to the framework location.

   If the images are bundled in a jar archive together with some Java operations implementation for instance, it might be convenient for you to refer the images not by the file name, but by their relative path location in the class-path.

   If the image file `Section20.png` is located in the **images** directory inside the jar archive, you can refer to it by using **/images/Section20.png**. The jar file must be added into the **Classpath** list.

6. Set the **Small icon (16x16)** field to `${frameworks}/sdf/Section16.png`.

7. Click the text field next to **Shortcut key** and set it to **Ctrl (Meta on Mac OS)+Shift+S**. This will be the key combination to trigger the action using the keyboard only.

The shortcut is enabled only by *adding the action to the main menu of the Author mode* which contains all the actions that the author will have in a menu for the current document type.

8. At this time the action has no functionality added to it. Next you must define how this action operates. An action can have multiple operation modes, each of them activated by the evaluation of an XPath version 2.0 expression. The first enabled action mode will be executed when the action is triggered by the user. The scope of the XPath expression must be only element nodes and attribute nodes of the edited document, otherwise the expression will not return a match and will not fire the action. For this example we'll suppose you want allow the action to add a section only if the current element is either a book, article or another section.

a) Set the XPath expression field to:

```
local-name()='section' or local-name()='book' or
 local-name()='article'
```

b) Set the **invoke operation** field to InsertFragmentOperation built-in operation, designed to insert an XML fragment at caret position. This belongs to a set of built-in operations, a complete list of which can be found in the *Author Default Operations* section. This set can be expanded with your own Java operation implementations.

c) Configure the arguments section as follows:

```
<section xmlns=
"http://www.oxygenxml.com/sample/documentation">
     <title/>
</section>
```

insertLocation - leave it empty. This means the location will be at the caret position.

insertPosition - select "**Inside**".

The Insert Table Action

You will create an action that inserts into the document a table with three rows and three columns. The first row is the table header. Similarly to the insert section action, you will use the InsertFragmentOperation.

Place the icon files for the menu item and for the toolbar in the frameworks/sdf directory.

1. Set **ID** field to **insert_table**.
2. Set **Name** field to **Insert table**.
3. Set **Menu access key** field to **t**.
4. Set **Description** field to **Adds a section element**.
5. Set **Toolbar icon** to **${framework} / toolbarIcon.png**.
6. Set **Menu icon** to **${framework} / menuIcon.png**.
7. Set **Shortcut key** to **Ctrl+Shift+T (Command+Shift+T on OS X)**.
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.

1. Open the Document Type edit dialog for the **SDF** framework and select on the **Author** tab. Next click on the **Toolbar** label.
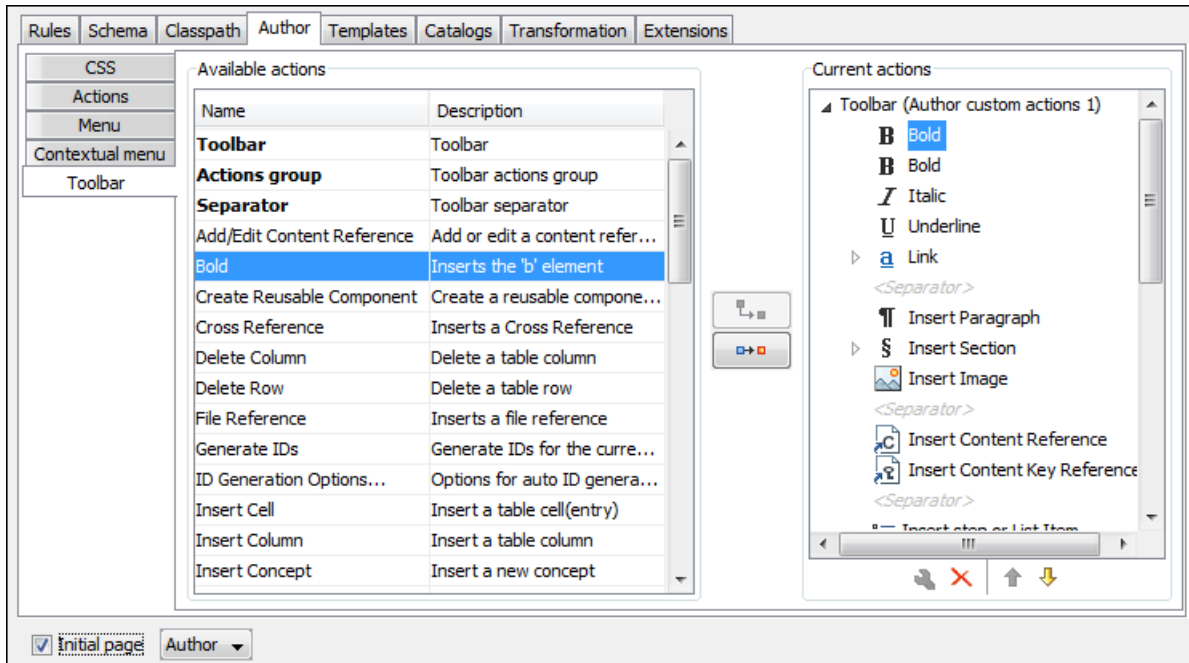
Figure 139: Configuring the Toolbar

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 140: Author Custom Actions Toolbar
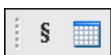
> ⓘ **Tip:** If you have many custom toolbar actions, or want to group actions according to their category, add additional toolbars with custom names and split the actions to better suit your purpose. In case your toolbar is not displayed when switching to the **Author** mode, right click the main toolbar and make sure the entry labeled **Author custom actions 1** is enabled.

Configuring the Main Menu

Defined actions can be grouped into customized menus in the Oxygen XML Author menu bar.

1. Open the Document Type dialog for the **SDF** framework and click on the **Author** tab.

2. Click on the **Menu** label. In the left side you have the list of actions and some special entries:

   - **Submenu** - Creates a submenu. You can nest an unlimited number of menus.
   - **Separator** - Creates a separator into a menu. This way you can logically separate the menu entries.

3. The right side of the panel displays the menu tree with **Menu** entry as root. To change its name click on this label to select it, then press the 🔧 **Edit** button. Enter **SD Framework** as name, and **D** as menu access key.

4. Select the **Submenu** label in the left panel section and the **SD Framework** label in the right panel section, then press the ⤷ **Add as child** button. Change the submenu name to **Table**, using the 🔧 **Edit** button.

5. Select the **Insert section** action in the left panel section and the **Table** label in the right panel section, then press the 🔽 **Add as sibling** button.

6. Now select the **Insert table** action in the left panel section and the **Table** in the right panel section. Press the ⤷ **Add as child** button.
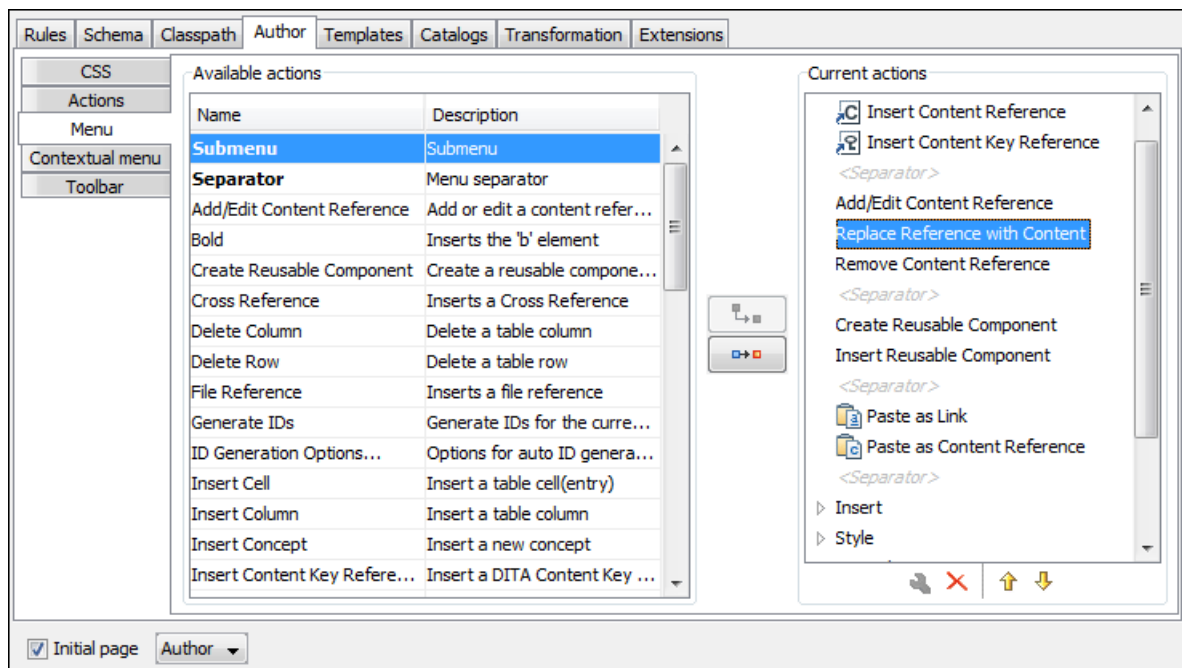
Figure 141: Configuring the Menu

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).
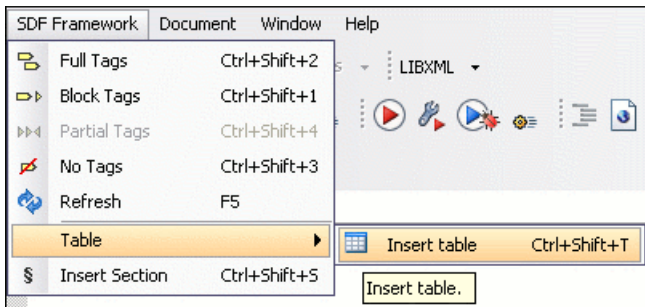
Figure 142: Author Menu

Configuring the Contextual Menu

The contextual menu is shown when you right click (**Ctrl** (**Meta** on Mac OS) + mouse click on Mac) in the Author editing area. In fact you are configuring the bottom part of the menu, since the top part is reserved for a list of generic actions like Copy, Paste, Undo, etc.

1. Open the Document Type dialog for the **SDF** framework and click on the **Author** tab. Next click on the **Contextual Menu** label.
2. Follow the same steps as explained in the *Configuring the Main Menu*, except changing the menu name because the contextual menu does not have a name.

   Note: You can choose to reuse a submenu that contains general authoring actions. In this case, all actions (both general and document type-specific ones) are grouped together under the same submenu.
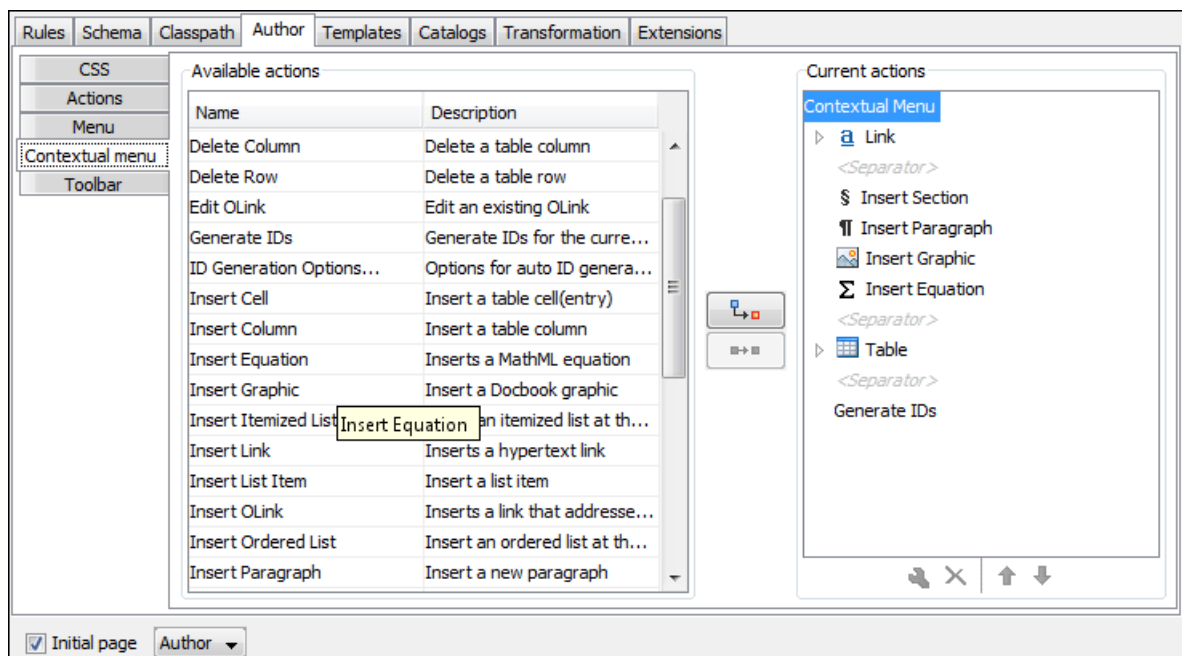
Figure 143: Configuring the Contextual Menu

To test it, open the test file, and open the contextual menu. In the lower part there is shown the **Table** sub-menu and the **Insert section** action.

Customize Content Completion

You can customize the content of the following **Author** controls, adding items (which, when invoked, perform custom actions) or filtering the default contributed ones:

* **Content Completion** window
* **Elements** view

- **Element Insert** menus (from the **Outline** view or breadcrumb contextual menus)

You can use the content completion customization support in the *Simple Documentation Framework* following the next steps:

1. Open the **Document type** edit dialog for the **SDF** framework and select the **Author** tab. Next click on the **Content Completion** tab.
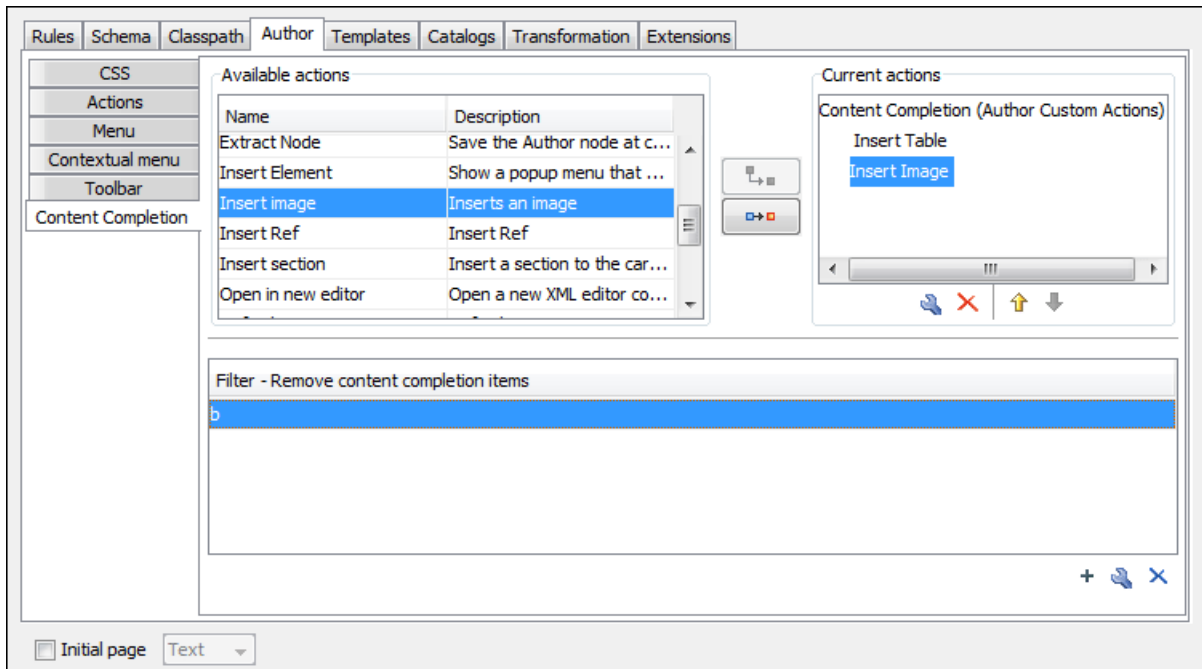
Figure 144: Customize Content Completion

The top side of the **Content Completion** section contains the list with all the actions defined within the simple documentation framework and the list of actions that you decided to include in the **Content Completion Assistant** list of proposals. The bottom side contains the list with all the items that you decided to remove from the **Content Completion Assistant** list of proposals.

2. If you want to add a custom action to the list of current **Content Completion** items, select the action item from the **Available actions** list and press the ⬆ **Add as child** or ⬆ **Add as sibling** button to include it in the **Current actions** list. The following dialog appears, giving you the possibility to select where to provide the selected action:
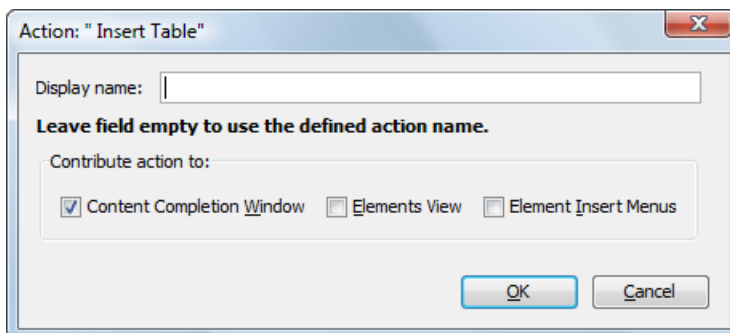
Figure 145: Insert action dialog

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. The **Item name** combo box accepts wildcards.
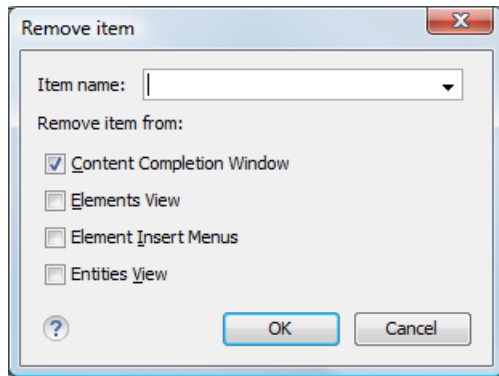
Figure 146: Remove item dialog

Author Mode Default Operations

Below are listed all the operations and their arguments:

• **InsertFragmentOperation**

Inserts an XML fragment at the current cursor position. The selection - if there is one, remains unchanged. The fragment will be inserted in the current context of the cursor position meaning that if the current XML document uses some namespace declarations then the inserted fragment must use the same declarations. The inserted fragment will not be copied and pasted to the cursor position, but the namespace declarations of the fragment will be adapted if needed to the existing namespace declarations of the XML document. For more details about the list of parameters go to *The arguments of InsertFragmentOperation operation* on page 339.

• **InsertOrReplaceFragmentOperation**

Similar to **InsertFragmentOperation**, except it removes the selected content before inserting the fragment.

• **InsertOrReplaceTextOperation**

Inserts a text at current position removing the selected content, if any. The argument of this operation is:

• **text** - the text section to insert.

• **SurroundWithFragmentOperation**

Surrounds the selected content with a text fragment. Since the fragment can have multiple nodes, the surrounded content will be always placed in the first leaf element. If there is no selection, the operation will simply insert the fragment at the caret position. For more details about the list of parameters go to *The arguments of SurroundWithFragmentOperation* on page 341.

• **SurroundWithTextOperation**

This operation has two arguments (two text values) that will be inserted before and after the selected content. If there is no selected content, the two sections will be inserted at the caret position. The arguments of the operation are:

• **header** - the text that is placed before the selection;
• **footer** - the text that is placed after the selection.

• **InsertEquationOperation**

Inserts a fragment containing a MathML equation at caret offset. The argument of this operation is:

• **fragment** - the XML fragment containing the MathML content which should be inserted.

• **InsertXIncludeOperation**

Insert an **XInclude** element at caret offset.

• **ChangeAttributeOperation**

This operation allows adding/modifying/removing an attribute. You can use this operation in your own Author action to modify the value for a certain attribute on a specific XML element. The arguments of the operation are:

- **name** - the attribute local name;
- **namespace** - the attribute namespace;
- **elementLocation** - the XPath location that identifies the element;
- **value** - the new value for the attribute. If empty or null the attribute will be removed;
- **editAttribute** - if an in-place editor exists for this attribute, it will automatically activate the in-pace editor and start editing;
- **removeIfEmpty** - the possible values are `true` and `false`. True means that the attribute should be removed if an empty value is provided. The default behavior is to remove it.

- **UnwrapTagsOperation**

  This operation allows removing the element tags either from the current element or for an element identified with an XPath location. The argument of the operation is

  - **unwrapElementLocation** - an XPath expression indicating the element to unwrap. If it is not defined, the element at caret is unwrapped.

- **ToggleSurroundWithElementOperation**

  This operation allows wrapping and unwrapping content in a specific element with specific attributes. It is useful to implement toggle actions like highlighting text as bold, italic, or underline. The arguments of the operation are:

  - **element** - the element to wrap or unwrap content;
  - **schemaAware** - this argument applies only on the surround with element operation and controls if the insertion is schema aware or not.

- **RenameElementOperation**

  This operation allows you to rename all occurrences of the elements identified by an XPath expression. The operation requires two parameters:

  - **elementName** - The new element name
  - **elementLocation** - The XPath expression that identifies the element occurrences to be renamed. If this parameter is missing, the operation renames the element at current caret position.

- **ExecuteTransformationScenariosOperation**

  This operation allows running one or more transformation scenarios defined in the current document type association. It is useful to add to the toolbar buttons that trigger publishing to various output formats. The argument of the operation is:

  - **scenarioNames** - the list of scenario names that will be executed, separated by new lines.

- **XSLTOperation** and **XQueryOperation**

  Applies an XSLT or XQuery script on a source element and then replaces or inserts the result in a specified target element.

  This operation has the following parameters:

  - **sourceLocation**

    An XPath expression indicating the element that the script will be applied on. If it is not defined then the element at the caret position will be used.

    There may be situations in which you want to look at an ancestor of the current element and take decisions in the script based on this. In order to do this you can set the sourceLocation to point to an ancestor node (for

example /) then declare a parameter called currentElementLocation in your script and use it to re-position in the current element like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
  xpath-default-namespace="http://docbook.org/ns/docbook"
  xmlns:saxon="http://saxon.sf.net/" exclude-result-prefixes="saxon">
    <!-- This is an XPath location which will be sent by the operation to the script -->
    <xsl:param name="currentElementLocation"/>

    <xsl:template match="/">
        <!-- Evaluate the XPath of the current element location -->
        <xsl:apply-templates
        select="saxon:eval(saxon:expression($currentElementLocation))"/>
    </xsl:template>

    <xsl:template match="para">
        <!-- And the context is again inside the current element,
        but we can use information from the entire XML -->
        <xsl:variable
            name="keyImage" select="//imagedata[@fileref='images/lake.jpeg']
                /ancestor::inlinemediaobject/@xml:id/string()"/>
                <xref linkend="{$keyImage}" role="key_include"
                  xmlns="http://docbook.org/ns/docbook">
                    <xsl:value-of
                        select="$currentElementLocation"></xsl:value-of>
        </xref>
    </xsl:template>
</xsl:stylesheet>
```

- **targetLocation**

  An XPath expression indicating the insert location for the result of the transformation. If it is not defined then the insert location will be at the caret.

- **script**

  The script content (XSLT or XQuery). The base system ID for this will be the framework file, so any include/import reference will be resolved relative to the .framework file that contains this action definition.

  For example, for the following script, the imported xslt_operation.xsl needs to be located in the current framework's directory.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
    <xsl:import href="xslt_operation.xsl"/>
</xsl:stylesheet>
```

- **action**

  The insert action relative to the node determined by the target XPath expression. It can be: Replace, At caret position, Before, After, Inside as first child or Inside as last child.

- **caretPosition**

  The position of the caret after the action is executed. It can be: Preserve, Before, Start, First editable position, End or After. If not specified the caret position ca be specified by outputting in the XSLT script a **${caret}** editor variable.

- **expandEditorVariables**

  Parameter controlling the expansion of editor variables returned by the script processing. Expansion is enabled by default.

- **ExecuteMultipleActionsOperation**

  This operation allows the execution of a sequence of actions, defined as a list of action IDs. The actions must be defined by the corresponding framework, or one of the common actions for all frameworks supplied by Oxygen XML Author.

  - **actionIDs** - the action IDs list which will be executed in sequence, the list must be a string sequence containing the IDs separated by new lines.

- **MoveElementOperation**

Flexible operation for moving an XML element to another location from the same document. XPath expressions are used to identify the source element and the target location. The operation takes the following parameters:

- **sourceLocation** - XPath expression that identifies the content to be moved.
- **deleteLocation** - XPath expression that identifies the node to be removed. This parameter is optional. If missing, the **sourceLocation** parameter will also identify the node to be deleted.
- **surroundFragment** - a string representation of an XML fragment. The moved node will be wrapped in this string before moving it in the destination.
- **targetLocation** - XPath expression that identifies the location where the node must be moved to.
- **insertPosition** - argument that indicates the insert position.
- **moveOnlySourceContentNodes** - when true, only the content of the source element is moved.

- **ChangePseudoClassesOperation**

  Operation that sets a list of pseudo class values to nodes identified by an XPath expression. It can also remove a list of values from nodes identified by an XPath expression. The operation accepts the following parameters:

  - **setLocations** - An XPath expression indicating a list of nodes on which the specified list of pseudo classes will be set. If it is not defined, then the element at the caret position will be used.
  - **setPseudoClassNames** - A space-separated list of pseudo class names which will be set on the matched nodes.
  - **removeLocations** - An XPath expression indicating a list of nodes from which the specified list of pseudo classes will be removed. If it is not defined, then the element at the caret position will be used.
  - **removePseudoClassNames** - A space-separated list of pseudo class names which will be removed from the matched nodes.

- **SetPseudoClassOperation**

  An operation that sets a pseudo-class to an element. The operation accepts the following parameters:

  - **elementLocation** - An XPath expression indicating the element on which the pseudo-class will be set. If it is not defined, then the element at caret position will be used.
  - **name** - The pseudo-class local name.

Author operations can take parameters that might contain the following editor variables:

- *${caret}* - The position where the caret is inserted. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- *${selection}* - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- *${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}* - To prompt for values at runtime, use the *ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value'')* editor variable. You can set the following parameters:

  - `'message'` - The displayed message. Note the quotes that enclose the message.
  - type - Optional parameter. Can have one of the following values:
    - url - Input is considered an URL. Oxygen XML Author checks that the URL is valid before passing it to the transformation.
    - password - Input characters are hidden.
    - generic - The input is treated as generic text that requires no special handling.
    - relative_url - Input is considered an URL. Oxygen XML Author tries to make the URL relative to that of the document you are editing.

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

    - combobox - Displays a dialog that contains a non-editable combo-box.
    - editable_combobox - Displays a dialog that contains an editable combo-box.

- `radio` - Displays a dialog that contains radio buttons.
- `'default-value'` - optional parameter. Provides a default value in the input text box;

  Examples:

  - `${ask('message')}` - Only the message displayed for the user is specified.
  - `${ask('message', generic, 'default')}` - `'message'` is displayed, the type is not specified (the default is string), the default value is `'default'`.
  - `${ask('message', password)}` - `'message'` is displayed, the characters typed are masked with a circle symbol.
  - `${ask('message', password, 'default')}` - same as before, the default value is `'default'`.
  - `${ask('message', url)}` - `'message'` is displayed, the parameter type is URL.
  - `${ask('message', url, 'default')}` - same as before, the default value is `'default'`.

- *${timeStamp}* - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- *${uuid}* - Universally unique identifier;An unique sequence of 32 hexadecimal digits generated by the Java *UUID* class.
- *${id}* - Application-level unique identifier;A short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
- *${cfn}* - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- *${cfne}* - Current file name with extension. The current file is the one currently opened and selected.
- *${cf}* - Current file as file path, that is the absolute file path of the current edited document.
- *${cfd}* - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- *${frameworksDir}* - The path (as file path) of the `[OXYGEN_DIR]/frameworks`directory.
- *${pd}* - Current project folder as file path. Usually the current folder selected in the Project View.
- *${oxygenInstallDir}* - Oxygen XML Author installation folder as file path.
- *${homeDir}* - The path (as file path) of the user home folder.
- *${pn}* - Current project name.
- *${env(VAR_NAME)}* - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the *${system(var.name)}* editor variable.
- *${system(var.name)}* - Value of the `var.name` Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as `-Dvar.name=var.value`. If you are looking for operating system environment variables, use the *${env(VAR_NAME)}* editor variable instead.
- *${date(pattern)}* - Current date. The allowed patterns are equivalent to the ones in the *Java SimpleDateFormat class*. Example: yyyy-MM-dd;

  📄 **Note:** This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to *http://www.w3.org/TR/xmlschema-2/#date*. For details about xs:datetime, go to *http://www.w3.org/TR/xmlschema-2/#dateTime*.

The arguments of `InsertFragmentOperation` operation

fragment

This argument has a textual value. This value is parsed by Oxygen XML 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.

You can even use namespace prefixes that are not declared in the inserted fragment, if they are declared in the document where the insertion is done. For the sake of clarity, you should always prefix and declare namespaces in the inserted fragment!

If the fragment contains namespace declarations that are identical to those found in the document, the namespace declaration attributes will be removed from elements contained by the inserted fragment.

There are two possible scenarios:

1. **Prefixes that are not bound explicitly**

   For instance, the fragment:

   ```
   <x:item id="dty2"/>
   &ent;
   <x:item id="dty3"/>
   ```

   Can be correctly inserted in the document: ('|' marks the insertion point):

   ```
   <?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.

`goToNextEditablePosition`

> After inserting the fragment, the first editable position is detected and the caret is placed at that location. It handles any in-place editors used to edit attributes. It will be ignored if the fragment specifies a caret position using the caret editor variable. The possible values of this action are **true** and **false**.

The arguments of `SurroundWithFragmentOperation`

The Author operation `SurroundWithFragmentOperation` has only one argument:

- fragment -

  The XML fragment that will surround the selection. For example let's consider the fragment:

  ```
  <F>
      <A></A>
      <B>
        <C></C>
      </B>
  </F>
  ```

  and the document:

  ```
  <doc>
    <X></X>
    <Y></Y>
    <Z></Z>
  <doc>
  ```

  Considering the selected content to be surrounded is the sequence of elements X and Y, then the result is:

  ```
  <doc>
     <F>
        <A>
           <X></X>
           <Y></Y>
        </A>
        <B>
           <C></C>
        </B>
     </F>
    <Z></Z>
  <doc>
  ```

  Because the element A was the first leaf in the fragment, it received the selected content. The fragment was then inserted in the place of the selection.

How to Add a Custom Operation to an Existing Document Type

This task explains how to add a custom Author operation to an existing document type.

1. Setup an Author sample project following *this set of instructions*. The framework project is **oxygen-sample-framework**.
2. A number of classes in the *simple.documentation.framework.operations* package implement the *ro.sync.ecss.extensions.api.AuthorOperation* interface. Depending on your use-case, modify one of these classes.
3. Pack the operation class inside a Java *jar* library.
4. Copy the *jar* library to the `[OXYGEN_DIR]/frameworks/[FRAMEWORK_DIR]` directory.
5. *Open the Preferences dialog* and go to **Document Type Association** page and edit the document type (you need write access to the `[OXYGEN_DIR]`).
   a) In the **Classpath** tab, add a new entry like: `${framework}/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_DIR]/frameworks/[FRAMEWORK_DIR]` directory.

Java API - Extending Author Functionality through Java

Oxygen XML Author 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 node set.

The following sections contain the Java programming interface (API) available to the developers. You will need the *Oxygen SDK* available *on the Oxygen XML Author 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 Author 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 Author 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 Author. Otherwise the classes may not be loaded by the Java virtual machine. For example if you run Oxygen XML Author XML Author 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. Setup an Author sample project following *this set of instructions*. The framework project is **oxygen-sample-framework**.
2. Modify 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 Author 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 Author 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.getWorkspaceAccess().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.getEditorAccess().getCaretOffset();
          authorAccess.getDocumentController().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-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen XML Author 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 Preferences dialog* and go to **Document Type Association**, 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 Author `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 **${framework}/toolbarImage.png**.

- Set **Menu icon** to **${framework}/menuImage.png**.
- Set **Shortcut key** to **Ctrl (Meta on Mac OS)+Shift+i**.

9. Now let's set up the operation. You want to add images only if the current element is a `section`, `book` or `article`.

- Set the value of **XPath expression** to

```
local-name()='section' or local-name()='book'
 or local-name()='article'
```

- Set the **Invoke operation** field to `simple.documentation.framework.InsertImageOperation`.
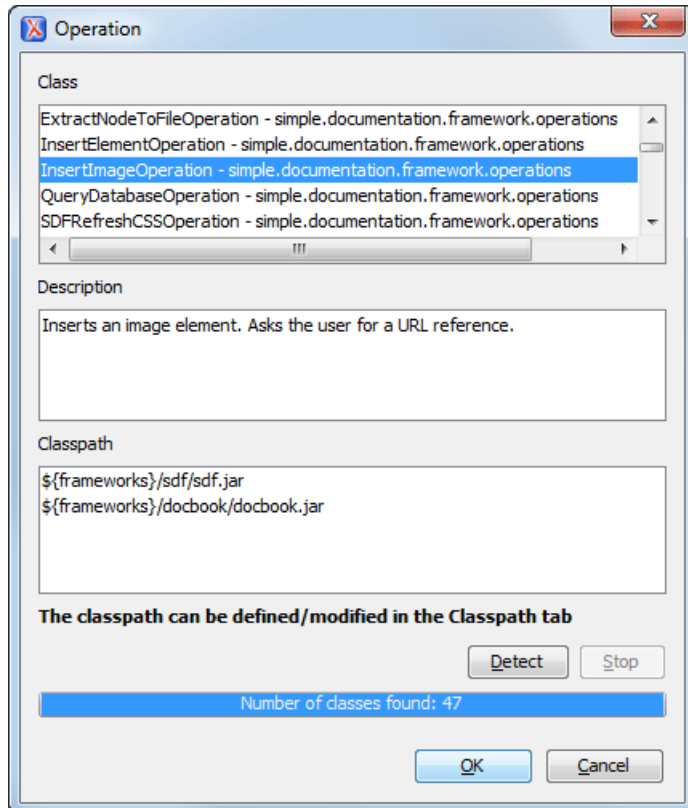
Figure 147: Selecting the Operation

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. Setup an Author sample project following *this set of instructions*. The framework project is **oxygen-sample-framework**.

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.getDocumentController().insertXMLFragment(
     getFragment(jdbcDriver, connection, user, password, sql),
     caretPosition);
  } catch (SQLException e) {
   throw new AuthorOperationException(
     "The operation failed due to the following database error: "
     + e.getMessage(), e);
  } catch (ClassNotFoundException e) {
   throw new AuthorOperationException(
     "The JDBC database driver was not found. Tried to load ' "
     + jdbcDriver + "'", e);
  }
 }
```

6. The `getFragment` method loads the JDBC driver, connects to the database and extracts the data. The result is a `table` element from the `http://www.oxygenxml.com/sample/documentation` namespace. The `header` element contains the names of the SQL columns. All the text from the XML fragment is escaped. This means that the '<' and '&' characters are replaced with the '<' and '&' character entities to ensure the fragment is well-formed.

```
private String getFragment(
   String jdbcDriver,
   String connectionURL,
   String user,
   String password,
   String sql) throws
    SQLException,
    ClassNotFoundException {

      Properties pr = new Properties();
      pr.put("characterEncoding", "UTF8");
      pr.put("useUnicode", "TRUE");
      pr.put("user", user);
      pr.put("password", password);

      // Loads the database driver.
      Class.forName(jdbcDriver);
      // Opens the connection
      Connection connection =
         DriverManager.getConnection(connectionURL, pr);
      java.sql.Statement statement =
         connection.createStatement();
      ResultSet resultSet =
         statement.executeQuery(sql);

      StringBuffer fragmentBuffer = new StringBuffer();
      fragmentBuffer.append(
        "<table xmlns=" +
        "'http://www.oxygenxml.com/sample/documentation'>");

      //
      // Creates the table header.
      //
      fragmentBuffer.append("<header>");
      ResultSetMetaData metaData = resultSet.getMetaData();
      int columnCount = metaData.getColumnCount();
      for (int i = 1; i <= columnCount; i++) {
          fragmentBuffer.append("<td>");
          fragmentBuffer.append(
            xmlEscape(metaData.getColumnName(i)));
          fragmentBuffer.append("</td>");
      }
      fragmentBuffer.append("</header>");

      //
      // Creates the table content.
      //
      while (resultSet.next()) {
          fragmentBuffer.append("<tr>");
          for (int i = 1; i <= columnCount; i++) {
              fragmentBuffer.append("<td>");
              fragmentBuffer.append(
                xmlEscape(resultSet.getObject(i)));
              fragmentBuffer.append("</td>");
          }
          fragmentBuffer.append("</tr>");
      }

      fragmentBuffer.append("</table>");

      // Cleanup
      resultSet.close();
      statement.close();
      connection.close();
      return fragmentBuffer.toString();
}
```

> 📄 **Note:** The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen XML Author 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 **${framework}/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 (Command+Shift+C on OS X)**.

11. The action will work only if the current element is a **section**. Set up the operation as follows:

- Set **XPath expression** to:

```
local-name()='section'
```

- Use the Java operation defined earlier to set the **Invoke operation** field. Press the **Choose** button, then select `simple.documentation.framework.QueryDatabaseOperation`. Once selected, the list of arguments is displayed. In the figure below the first argument, *jdbc_driver*, represents the class name of the MySQL JDBC driver. The connection string has the URL syntax : *jdbc://<database_host>:<database_port>/<database_name>*.

  The SQL expression used in the example follows, but it can be any valid SELECT expression which can be applied to the database:

```
SELECT userID, email FROM users
```

12. Add the action to the toolbar, using the **Toolbar** panel.
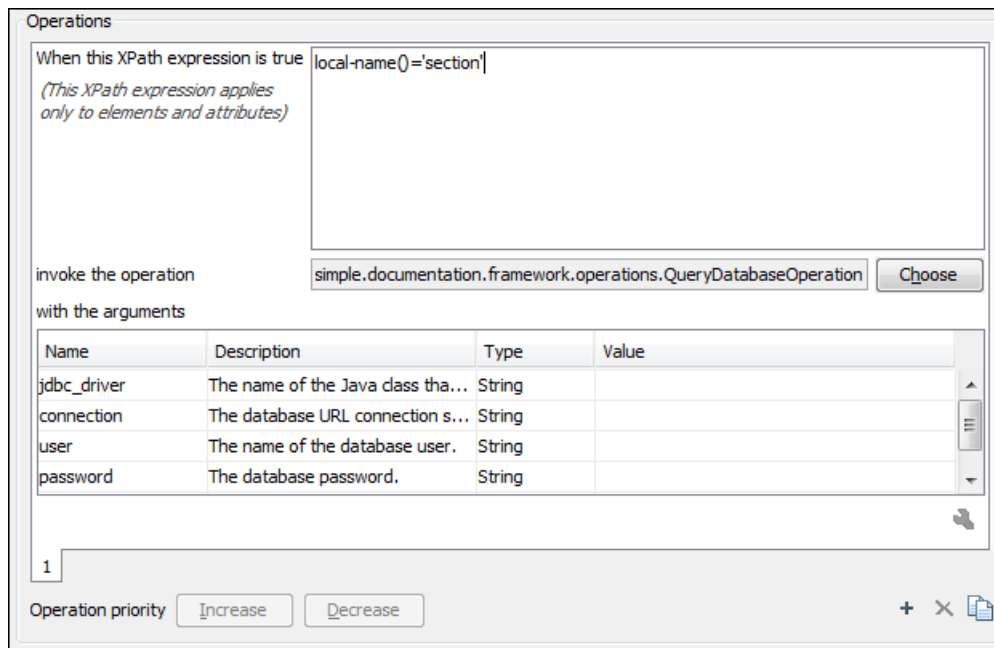
Figure 148: Java Operation Arguments Setup

To test the action you can open the *sdf_sample.xml* sample place the caret inside a `section` between two para elements for instance. Press the 🗄 **Create Report** button from the toolbar. You can see below the toolbar with the action button and sample table inserted by the **Clients Report** action.
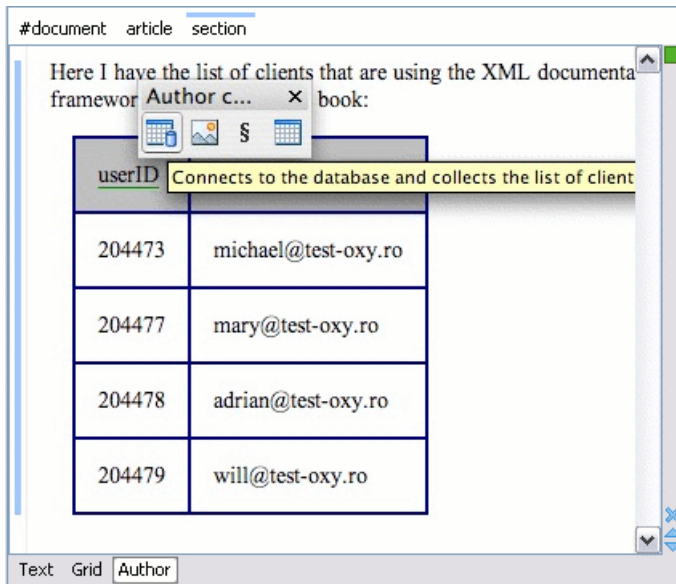
Figure 149: Table Content Extracted from the Database

Editing attributes in-place using form controls

To edit attributes in the **Author** mode, use the *Attributes View*.

The *oxy_editor* CSS extension function allows you to edit attribute and element text values directly in the Author mode using form-based controls. Various implementations are available out of the box: *combo boxes*, *checkboxes*, *text fields*, *pop-ups*, *buttons* which invoke custom Author actions or *URL choosers*. You can also implement custom editors for your specific needs.

As a working example, the bundled samples project contains a file called personal.xml which allows editing attributes in-place using some of these default implementations.

Localizing Frameworks

Oxygen XML Author supports framework localization (translating framework actions, buttons, and menu entries to different languages). This lets you develop and distribute a framework to users that speak different languages without changing the distributed framework. Changing the language used in Oxygen XML Author in the Global preferences page is enough to set the right language for each framework.

To localize the content of a framework, create a translation.xml file which contains all the translation (key, value) mappings. The translation.xml has the following format:

```
<translation>
    <languageList>
        <language description="English" lang="en_US"/>
        <language description="German" lang="de_DE"/>
        <language description="French" lang="fr_FR"/>
    </languageList>
    <key value="list">
        <comment>List menu item name.</comment>
        <val lang="en_US">List</val>
        <val lang="de_DE">Liste</val>
        <val lang="fr_FR">Liste</val>
    </key>
.....................
</translation>
```

Oxygen XML Author matches the GUI language with the language set in the translation.xml file. In case this language is not found, the first available language declared in the languagelist tag for the corresponding framework is used.

Add the directory where this file is located to the **Classpath** list corresponding to the edited document type.

After you create this file, you are able to use the keys defined in it to customize the name and description of:

- framework actions
- menu entries
- contextual menus
- toolbar
- static CSS content

For example, if you want to localize the bold action *open the **Preferences** dialog* and go to **Document Type Association**. Open the **Document type** dialog, go to **Author** > **Actions**, and rename the bold action to `${i18n(translation_key)}`. Actions with a name format different than `${i18n(translation_key)}` are not localized. `Translation_key` corresponds to the key from the translation.xml file.

Now open the `translation.xml` file and edit the translation entry if it exists or create one if it does not exist. This example presents an entry in the translation.xml file:

```
<key value="translation_key">
        <comment>Bold action name.</comment>
        <val lang="en_US">Bold</val>
        <val lang="de_DE">Bold</val>
        <val lang="fr_FR">Bold</val>
    </key>
```

To use a description from the `translation.xml` file in the Java code used by your custom framework, use the new `ro.sync.ecss.extensions.api.AuthorAccess.getAuthorResourceBundle()` API method to request for a certain key the associated value. In this way all the dialogs that you present from your custom operations can have labels translated in different languages.

You can also refer a key directly in the CSS content:

```
title:before{
    content:"${i18n(title.key)} : ";
}
```

📄 **Note:** You can enter any language you want in the `languagelist` tag and any number of keys.

> The `translation.xml` file for the DocBook framework is located here:`[OXYGEN_DIR]/frameworks/docbook/i18n/translation.xml`. In the **Classpath** list corresponding to the DocBook document type the following entry was added: `${framework}/i18n/`.
>
> To see how the DocBook actions are defined to use these keys for their name and description, *open the **Preferences** dialog* and go to **Document Type Association** > **Author** > **Actions**, you can If you look in the Java class `ro.sync.ecss.extensions.docbook.table.SADocbookTableCustomizerDialog` available in the oxygen-sample-framework module of the *Oxygen SDK* Maven archetype, you can see how the new `ro.sync.ecss.extensions.api.AuthorResourceBundle` API is used to retrieve localized descriptions for different keys.

How to Pack and Deploy an Add-on

Packing a Plugin or Framework as an Add-on

This procedure is suitable for developers who want a better control over the add-on package or those who want to automate some of the steps:

1. Pack the plugin or framework as a `ZIP` file or a *Java Archive (JAR)*. Please note that you should pack the entire root directory not just its contents.
2. Digitally sign the package. Please note that you can perform this step only if you have created a *JAR* at the previous step. You will need a certificate signed by a trusted authority. To sign the jar you can either use the `jarsigner`

command line tool inside Oracle's Java Development Kit. ([JDK_DIR]/bin/jarsigner.exe) or, if you are working with *Apache Ant*, you can use the signjar task (which is just a front for the jarsigner command line tool).

> **Note:** The benefit of having a signed add-on is that the user can verify the integrity of the add-on issuer. If you don't have such a certificate you can generate one yourself using the keytool command line tool. Please note that this approach is mostly recommended for tests since anyone can create a self signed certificate.

3. Create a descriptor file. You can use a template that Oxygen XML Author provides. To use this template, go to **File** > **New** and select the **Oxygen add-ons update site** template. Once deployed, this descriptor file is referred as *update site*.

Alternatively, you can use the **Add-ons Packager** plugin by following this procedure:

1. Install the **Add-ons Packager** plugin from *http://www.oxygenxml.com/InstData/Addons/optional/updateSite.xml* as described in the *Installing Add-ons procedure*.
2. Restart Oxygen XML Author. If the add-on is correctly installed, the **Add-ons packager** toolbar action is available.
3. Invoke the **Add-ons packager** toolbar action and input the required information in the displayed dialog box.
4. Press **OK** to complete the packaging process.

Deploying an Add-on

To deploy an add-on, copy the ZIP/*JAR* file and the descriptor file to an HTTP server. The URL to this location serves as the *Update Site URL*.

Creating the Basic Association

Let us go through an example of creating a document type and editing an XML document of this type. We will call our document type **Simple Documentation Framework**.

First Step - XML Schema

Our documentation framework will be very simple. The documents will be either articles or books, both composed of sections. The sections may contain titles, paragraphs, figures, tables and other sections. To complete the picture, each section will include a def element from another namespace.

The first schema file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.oxygenxml.com/sample/documentation"
    xmlns:doc="http://www.oxygenxml.com/sample/documentation"
    xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts"
    elementFormDefault="qualified">

    <xs:import namespace=
    "http://www.oxygenxml.com/sample/documentation/abstracts"
     schemaLocation=
    "abs.xsd"/>
```

The namespace of the documents will be http://www.oxygenxml.com/sample/documentation. The namespace of the def element is http://www.oxygenxml.com/sample/documentation/abstracts.

Now let's define the structure of the sections. They all start with a title, then have the optional def element then either a sequence of other sections, or a mixture of paragraphs, images and tables.

```
<xs:element name="book" type="doc:sectionType"/>
<xs:element name="article" type="doc:sectionType"/>
<xs:element name="section" type="doc:sectionType"/>

<xs:complexType name="sectionType">
    <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element ref="abs:def" minOccurs="0"/>
        <xs:choice>
            <xs:sequence>
                <xs:element ref="doc:section" maxOccurs="unbounded"/>
```

```
            </xs:sequence>
            <xs:choice maxOccurs="unbounded">
                <xs:element ref="doc:para"/>
                <xs:element ref="doc:image"/>
                <xs:element ref="doc:table"/>
            </xs:choice>
        </xs:choice>
    </xs:sequence>
</xs:complexType>
```

The paragraph contains text and other styling markup, such as bold (b) and italic (i) elements.

```
<xs:element name="para" type="doc:paragraphType"/>

<xs:complexType name="paragraphType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="b"/>
        <xs:element name="i"/>
    </xs:choice>
</xs:complexType>
```

The image element has an attribute with a reference to the file containing image data.

```
<xs:element name="image">
    <xs:complexType>
        <xs:attribute name="href" type="xs:anyURI" use="required"/>
    </xs:complexType>
</xs:element>
```

The table contains a header row and then a sequence of rows (tr elements) each of them containing the cells. Each cell has the same content as the paragraphs.

```
 <xs:element name="table">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="header">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td" maxOccurs="unbounded"
                            type="doc:paragraphType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="tr" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td" type="doc:tdType"
                            maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>


<xs:complexType name="tdType">
    <xs:complexContent>
        <xs:extension base="doc:paragraphType">
            <xs:attribute name="row_span" type="xs:integer"/>
            <xs:attribute name="column_span" type="xs:integer"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

The def element is defined as a text only element in the imported schema abs.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=
     "http://www.oxygenxml.com/sample/documentation/abstracts">
    <xs:element name="def" type="xs:string"/>
</xs:schema>
```

Now the XML data structure will be styled.

Schema Settings

In *the dialog for editing the document type properties*, in the bottom section there are a series of tabs. The first one refers to the schema that is used for validation of the documents that match the defined **Association Rules**.

> ⚠ **Important:**
>
> If the document refers a schema, using for instance a `DOCTYPE` declaration or a `xsi:schemaLocation` attribute, the schema from the document type association will not be used when validating.

Schema Type

Select from the combo box the value **XML Schema**.

Schema URI

Enter the value `${frameworks}/sdf/schema/sdf.xsd`. We should use the *${frameworks}* editor variable in the schema URI path instead of a full path in order to be valid for different Oxygen XML Author 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 Author Author change the style of the parent to `block` display.

Styling the `section` Element

The title of any section must be bold and smaller than the title of the parent section. To create this effect a sequence of CSS rules must be created. The `*` operator matches any element, it can be used to match titles having progressive depths in the document.

```
title{
    font-size: 2.4em;
    font-weight:bold;
}
* * title{
```

```
    font-size: 2.0em;
}
* * * title{
    font-size: 1.6em;
}
* * * * title{
    font-size: 1.2em;
}
```

It's useful to have before the title a constant text, indicating that it refers to a section. This text can include also the current section number. The :before and :after pseudo elements will be used, plus the CSS counters.

First declare a counter named sect for each book or article. The counter is set to zero at the beginning of each such element:

```
book,
article{
    counter-reset:sect;
}
```

The sect counter is incremented with each section, that is a direct child of a book or an article element.

```
book > section,
article > section{
    counter-increment:sect;
}
```

The "static" text that will prefix the section title is composed of the constant "Section ", followed by the decimal value of the sect counter and a dot.

```
book > section > title:before,
article > section > title:before{
    content: "Section " counter(sect) ". ";
}
```

To make the documents easy to read, you add a margin to the sections. In this way the higher nesting level, the larger the left side indent. The margin is expressed relatively to the parent bounds:

```
section{
    margin-left:1em;
    margin-top:1em;
}
```
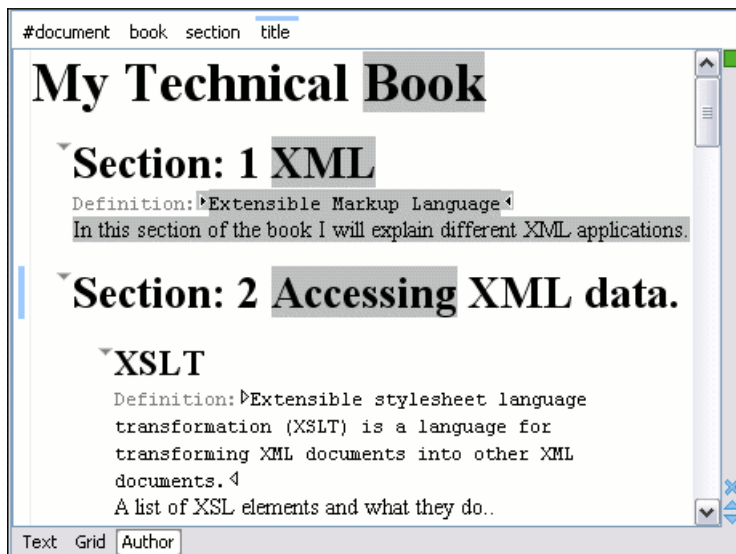

Figure 150: A sample of nested sections and their titles.

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 Author 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 Author.

```
image{
    display:block;
    content: attr(href, url);
    margin-left:2em;
}
```

Our `image` element has the required attribute `href` of type `xs:anyURI`. The `href` attribute contains an image location so the rendered content is obtained by using the function:

```
attr(href, url)
```

The first argument is the name of the attribute pointing to the image file. The second argument of the `attr` function specifies the type of the content. If the type has the `url` value, then Oxygen XML Author identifies the content as being an image. If the type is missing, then the content will be the text representing the attribute value.

Oxygen XML Author Author handles both absolute and relative specified URLs. If the image has an *absolute* URL location (for example: "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 (for example: "images/my_screenshot.jpg") then the location is obtained by adding this value to the location of the edited XML document.

An image can also be referenced by the name of a DTD entity which specifies the location of the image file. For example if the document declares an entity **graphic** which points to a JPEG image file:

```
<!ENTITY graphic SYSTEM "depo/keyboard_shortcut.jpg" NDATA JPEG>
```

and the image is referenced in the XML document by specifying the name of the entity as the value of an attribute:

```
<mediaobject>
    <imageobject>
        <imagedata entityref="graphic" scale="50"/>
    </imageobject>
</mediaobject>
```

The CSS should use the functions `url`, `attr` and `unparsed-entity-uri` for displaying the image in the Author mode:

```
imagedata[entityref]{
    content: url(unparsed-entity-uri(attr(entityref)));
}
```

To take into account the value of the `width` attribute of the `imagedata` and use it for resizing the image, the CSS can define the following rule:

```
imagedata[width]{
  width:attr(width, length);
}
```
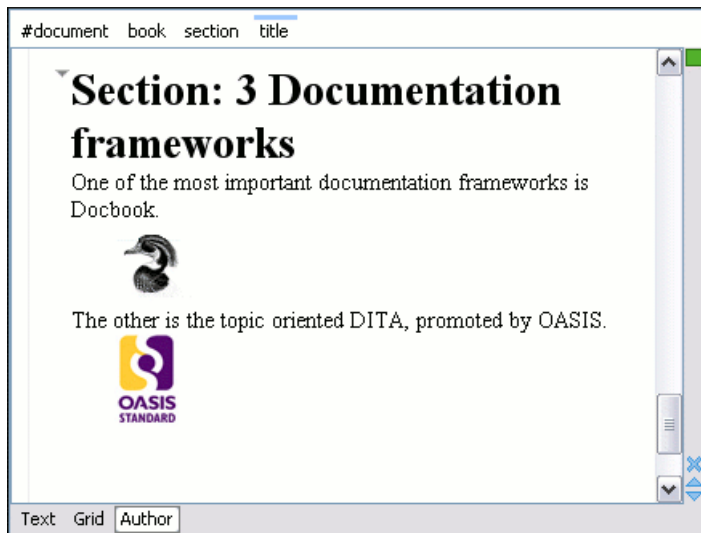

Figure 151: Samples of images in Author

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 Author 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 Author should load the CSS from the document type association and create a layout similar to this:
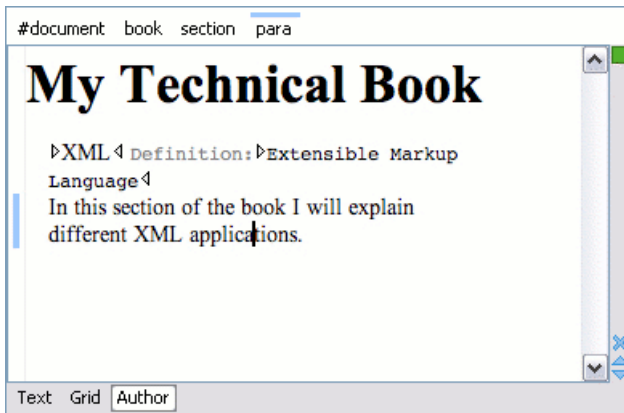
Organizing the Framework Files

First, create a new folder called `sdf` (from "Simple Documentation Framework") in `[OXYGEN_DIR]/frameworks`. This folder will be used to store all files related to the documentation framework. The following folder structure will be created:

```
oxygen
  frameworks
    sdf
      schema
      css
```

The `frameworks` directory is the container where all the Oxygen XML Author framework customizations are located. Each subdirectory contains files related to a specific type of XML documents: schemas, catalogs, stylesheets, CSS stylesheets, etc. Distributing a framework means delivering a framework directory.

It is assumed that you have the right to create files and folder inside the Oxygen XML Author installation directory. If you do not have this right, you will have to install another copy of the program in a folder you have access to, the home directory for instance, or your desktop. You can download the "all platforms" distribution from the oXygen website and extract it in the chosen folder.

To test your framework distribution, copy it in the `frameworks` directory of the newly installed application and start Oxygen XML Author 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 Author `[OXYGEN_DIR]/frameworks` directory. Select the `sdf` directory and make an archive from it. Move it to another Oxygen XML Author installation (eventually on another computer). Extract it in the `[OXYGEN_DIR]/frameworks` directory. Start Oxygen XML Author 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 Author 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 Author will not be able to update the Document Type Association options automatically on the deployed computers.

Configuring New File Templates

You will create a set of document templates that the content authors will use as starting points for creating *Simple Document Framework* books and articles.

Each Document Type Association can point to a directory, usually named `templates`, containing the file templates. All files found here are considered templates for the respective document type. The template name is taken from the file name, and the template type is detected from the file extension.

1. Go to the `[OXYGEN_DIR]\frameworks\sdf` directory and create a directory named `templates`. The directory tree of the documentation framework now is:

```
oxygen
    frameworks
        sdf
            schema
            css
            templates
```

2. In the `templates` directory create two files: a file for the *book* template and another one for the *article* template.

    The `Book.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://www.oxygenxml.com/sample/documentation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">
    <title>Book Template Title</title>
    <section>
        <title>Section Title</title>
        <abs:def/>
        <para>This content is copyrighted:</para>
        <table>
            <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.

    > 📝 **Note:** You should avoid using the `${cfd}`,`${cf}`,`${cfu}`, and `${cfdu}` editor variables when you save your documents in a data base.

3. Open the **Document Type** dialog for the **SDF** framework and click the **Templates** tab. In the **Templates directory** text field, introduce the `${frameworkDir}/templates` path. As you have already seen before, it is recommended that all the file references made from a Document Type Association to be relative to the *${frameworkDir}* directory. Binding a Document Type Association to an absolute file (e. g.: `C:\some_dir\templates`) makes the association difficult to share between users.

4. To test the templates settings, go to **File** > **New** to display the **New** dialog. The names of the two templates are prefixed with the name of the Document Type Association (**SDF** in this case). Selecting one of them should create a new XML file with the content specified in the template file.

Editor Variables

An editor variable is a shorthand notation for context-dependent information, like a file or folder path, a time-stamp, or a date. It is used in the definition of a command (for example the input URL of a transformation, the output file path of a transformation, the command line of an external tool) to make a command or a parameter generic and reusable with other input files. When the same command is applied to different files, the notation is expanded at the execution of the command so that the same command has different effects depending on the actual file.

You can use the following editor variables in Oxygen XML Author commands of external engines or other external tools, in transformation scenarios, validation scenarios, and Author operations:

- *${oxygenHome}* - Oxygen XML Author installation folder as URL.
- *${oxygenInstallDir}* - Oxygen XML Author installation folder as file path.
- *${framework}* - The path (as URL) of the current framework, as part of the `[OXYGEN_DIR]/frameworks` directory.
- *${framework(fr_name)}* - The path (as URL) of the `fr_name` framework.
- *${frameworkDir(fr_name)}* - The path (as file path) of the `fr_name` framework.

  > **Note:** Because multiple frameworks might have the same name (although it is not recommended), for both *${framework(fr_name)}* and *${frameworkDir(fr_name)}* editor variables Oxygen XML Author employs the following algorithm when searching for a given framework name:
  >
  > - all frameworks are sorted, from high to low, according to their ***Priority*** setting from the framework configuration
  > - next, if the two or more frameworks have the same name and priority, a further sorting based on the **Storage** setting is made, in the exact following order:
  >
  >   - frameworks stored in the internal Oxygen XML Author options
  >   - additional frameworks added in the ***Locations*** *preferences page*
  >   - frameworks installed using the add-ons support
  >   - frameworks found in the *main frameworks location* (**Default** or **Custom**)

- *${frameworks}* - The path (as URL) of the `[OXYGEN_DIR]` directory.
- *${frameworkDir}* - The path (as file path) of the current framework, as part of the `[OXYGEN_DIR]/frameworks` directory.
- *${frameworksDir}* - The path (as file path) of the `[OXYGEN_DIR]/frameworks`directory.
- *${home}* - The path (as URL) of the user home folder.
- *${homeDir}* - The path (as file path) of the user home folder.
- *${pdu}* - Current project folder as URL. Usually the current folder selected in the Project View.
- *${pd}* - Current project folder as file path. Usually the current folder selected in the Project View.
- *${pn}* - Current project name.
- *${cfdu}* - Current file folder as URL, that is the path of the current edited document up to the name of the parent folder, represented as a URL.
- *${cfd}* - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- *${cfn}* - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- *${cfne}* - Current file name with extension. The current file is the one currently opened and selected.
- *${cf}* - Current file as file path, that is the absolute file path of the current edited document.
- *${cfu}* - The path of the current file as a URL. The current file is the one currently opened and selected.
- *${af}* - The local file path of the ZIP archive that includes the current edited document.
- *${afu}* - The URL path of the ZIP archive that includes the current edited document.
- *${afd}* - The local directory path of the ZIP archive that includes the current edited document.
- *${afdu}* - The URL path of the directory of the ZIP archive that includes the current edited document.

- *${afn}* - The file name (without parent directory and without file extension) of the zip archive that includes the current edited file.
- *${afne}* - The file name (with file extension, for example `.zip` or `.epub`, but without parent directory) of the zip archive that includes the current edited file.
- *${currentFileURL}* - Current file as URL, that is the absolute file path of the current edited document represented as URL.
- *${ps}* - Path separator, that is the separator which can be used on the current platform (Windows, OS X, Linux) between library files specified in the class path.
- *${timeStamp}* - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- *${caret}* - The position where the caret is inserted. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- *${selection}* - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- *${id}* - Application-level unique identifier;A short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
- *${uuid}* - Universally unique identifier;An unique sequence of 32 hexadecimal digits generated by the Java *UUID* class.
- *${env(VAR_NAME)}* - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the *${system(var.name)}* editor variable.
- *${system(var.name)}* - Value of the `var.name` Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as `-Dvar.name=var.value`. If you are looking for operating system environment variables, use the *${env(VAR_NAME)}* editor variable instead.
- *${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}* - To prompt for values at runtime, use the *ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value'')* editor variable. You can set the following parameters:
    - `'message'` - The displayed message. Note the quotes that enclose the message.
    - `type` - Optional parameter. Can have one of the following values:
        - `url` - Input is considered an URL. Oxygen XML Author checks that the URL is valid before passing it to the transformation.
        - `password` - Input characters are hidden.
        - `generic` - The input is treated as generic text that requires no special handling.
        - `relative_url` - Input is considered an URL. Oxygen XML Author tries to make the URL relative to that of the document you are editing.

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

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

    Examples:

    - `${ask('message')}` - Only the message displayed for the user is specified.
    - `${ask('message', generic, 'default')}` - `'message'` is displayed, the type is not specified (the default is string), the default value is `'default'`.
    - `${ask('message', password)}` - `'message'` is displayed, the characters typed are masked with a circle symbol.
    - `${ask('message', password, 'default')}` - same as before, the default value is `'default'`.
    - `${ask('message', url)}` - `'message'` is displayed, the parameter type is URL.
    - `${ask('message', url, 'default')}` - same as before, the default value is `'default'`.

- *${date(pattern)}* - Current date. The allowed patterns are equivalent to the ones in the *Java SimpleDateFormat class*. Example: yyyy-MM-dd;

  > **Note:** This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to *http://www.w3.org/TR/xmlschema-2/#date*. For details about xs:datetime, go to *http://www.w3.org/TR/xmlschema-2/#dateTime*.

- *${dbgXML}* - The local file path to the XML document which is current selected in the Debugger source combo box (for tools started from the XSLT/XQuery Debugger).
- *${dbgXSL}* - The local file path to the XSL/XQuery document which is current selected in the Debugger stylesheet combo box (for tools started from the XSLT/XQuery Debugger).
- *${tsf}* - The transformation result file path. If the current opened file has an associated scenario which specifies a transformation output file, this variable expands to it.
- *${dsu}* - The path of the detected schema as an URL for the current validated XML document.
- *${ds}* - The path of the detected schema as a local file path for the current validated XML document.
- *${cp}* - Current page number. Used to display the current page number on each printed page in the **Editor / Print** Preferences page.
- *${tp}* - Total number of pages in the document. Used to display the total number of pages on each printed page in the **Editor / Print** Preferences page.
- *${xpath_eval(expression)}* - Evaluates an XPath 3.0 expression. Depending on the context, the expression can be:

  - *static*, when executed in a non-XML context. For example, you can use such static expressions to perform string operations on other editor variables for composing the name of the output file in a transformation scenario's **Output** tab.

    Example:

    ```
    ${xpath_eval(upper-case(substring('${cfn}', 1, 4)))}
    ```

  - *dynamic*, when executed in an XML context. For example, you can use such dynamic expression in a code template or as a value of an author operation's parameter.

    Example:

    ```
    ${ask('Set new ID attribute', generic, '${xpath_eval(@id)}')}
    ```

- *${i18n(key)}* - Editor variable used only at document type/framework level to allow translating names and descriptions of Author actions in multiple actions. More details here:*Localizing Frameworks* on page 348

Custom Editor Variables

An editor variable can be created by the user and included in any user defined expression where a built-in editor variable is also allowed. For example a custom editor variable may be necessary for configuring the command line of an external tool, the working directory of a custom validator, the command line of a custom XSLT engine, a custom FO processor, etc. All the custom editor variables are listed together with the built-in editor variables, for example when editing the working folder or the command line of an external tool or of a custom validator, the working directory, etc.

Creating a custom editor variable is very simple: just specify the name that will be used in user defined expressions, the value that will replace the variable name at runtime and a textual description for the user of that variable.

You can configure the custom editor variables in the **Preferences** page.

Create Your Own Stylesheet Templates

Oxygen XML Author allows you to create your own stylesheets templates and place them in the templates directory:

- Customize the stylesheet (add namespaces etc.) that you want to become a template and save it to a file with an appropriate name.
- Copy the file to the `templates` directory in the Oxygen XML Author installation directory.
- Open Oxygen XML Author and go to **File** > **New** to see your custom template.

Configuring XML Catalogs

In the XML sample file for **SDF** you did not use a 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_DIR]/frameworks/sdf` directory. The content of the file should be:

   ```
   <?xml version="1.0"?>
   <catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
       <uri name="http://www.oxygenxml.com/SDF/abs.xsd"
               uri="schema/abs.xsd"/>
       <uri name="http://www.oxygenxml.com/SDF/abs.xsd"
                uri="schema/abs.xsd"/>
   </catalog>
   ```

2. Add catalog files to your Document Type Association using the Catalogs tab from the Document Type dialog.

To test the catalog settings, restart Oxygen XML Author 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 Preferences dialog* and go to **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 `${framework}/xsl/sdf.xsl`.
   • Set the **Transformer** to *Saxon 9B*.
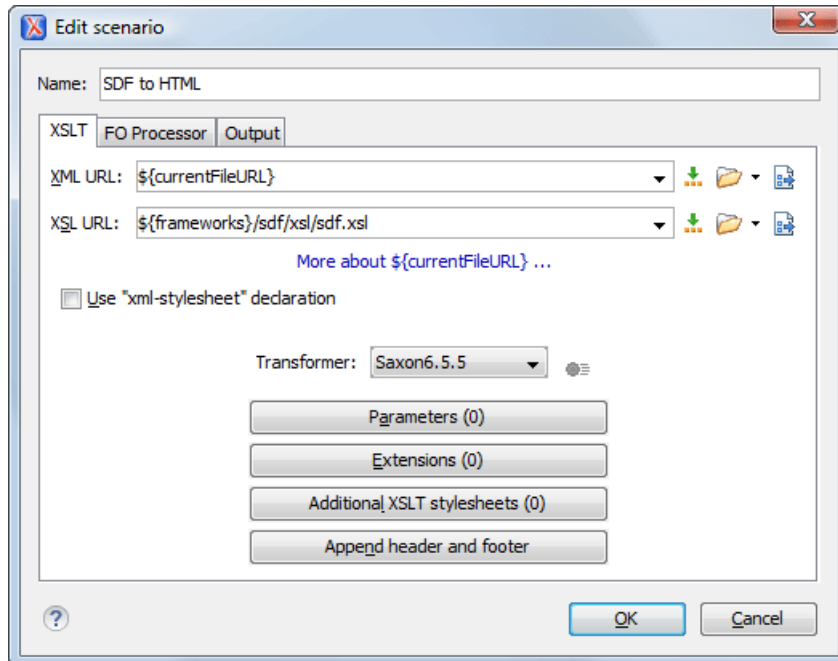
Figure 152: Configuring a transformation scenario

4. Change to the **Output** tab. Configure the fields as follows:

   • Set the **Save as** field to `${cfd}/${cfn}.html`. This means the transformation output file will have the name of the XML file and the *html* extension and will be stored in the same folder.
   • Enable the **Open in Browser/System Application** option.

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

   • Enable the **Saved file** option.

Now the scenario is listed in the **Transformation** tab:
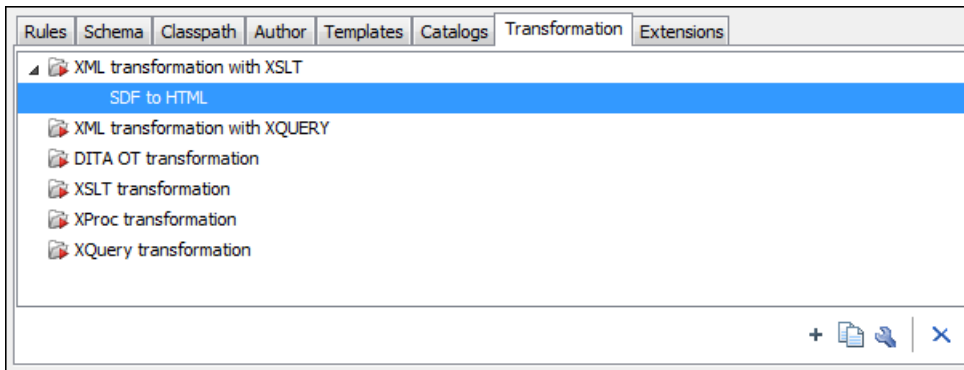
Figure 153: The transformation tab

To test the transformation scenario you just created, open the **SDF** XML sample from the *Example Files Listings*. Click

the ▶ **Apply Transformation Scenario(s)** button to display the **Configure Transformation Scenario(s)** dialog. Its scenario list contains the scenario you defined earlier *SDF to HTML*. Click it then choose **Transform now**. The HTML file should be saved in the same folder as the XML file and displayed in the browser.
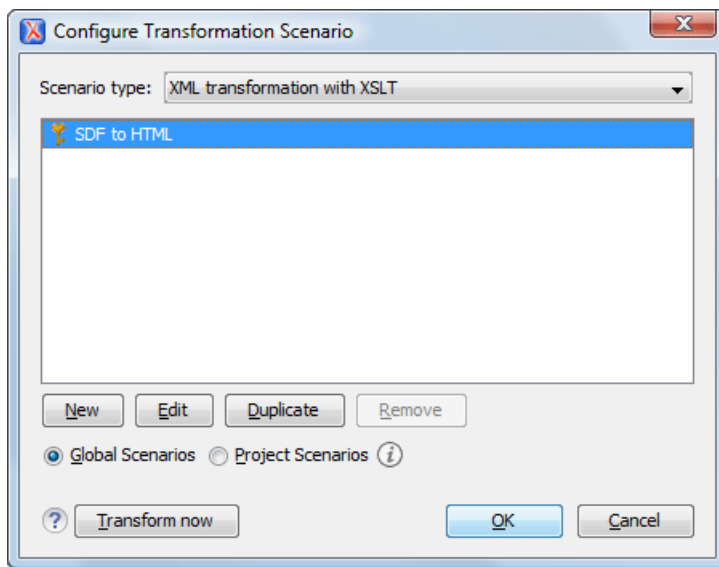
Figure 154: Selecting the predefined scenario

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 Preferences dialog* and go to **Document Type Associations**. Open the **Document Type** dialog for the **SDF** framework, then choose the **Validation** tab. This tab holds a list of document types for which you can define validation scenarios. To set one of the validation scenarios as default for a specific document type, select it and press
   ✅ / ▭ **Toggle default**.
2. Press the **New** button to add a new scenario.
3. Press the **Add** button to add a new validation unit with default settings.
   The dialog that lists all validation units of the scenario is opened.
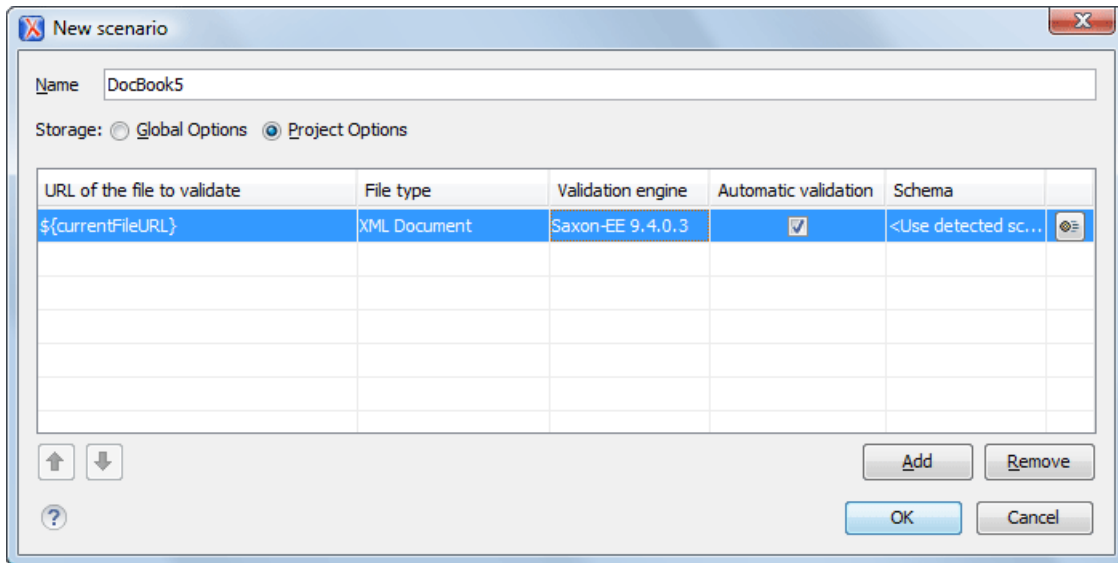
Figure 155: Add / Edit a Validation Unit

The table holds the following information:

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

4. Edit the URL of the main validation module.

Specify the URL of the main module:

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

```
${Desktop} - My Desktop
${start-dir} - Start directory of custom validator
${standard-params} - List of standard params for command line
${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 156: 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.

> 📄 **Note:** It is possible for a plugin to share the same classes with a framework. For further details, go to *How to Share the Classloader Between a Framework and a Plugin*.

Configuring an Extensions Bundle

Starting with Oxygen XML Author 10.3 version a single bundle was introduced acting as a provider for all other extensions. The individual extensions can still be set and if present they take precedence over the single provider, but this practice is being discouraged and the single provider should be used instead. To set individual extensions, *open the Preferences dialog* and go to **Document Type Association**, double-click a document type and go to the extension tab.

The extensions bundle is represented by the *ro.sync.ecss.extensions.api.ExtensionsBundle* class. The provided implementation of the ExtensionsBundle is instantiated when the rules of the Document Type Association defined for the custom framework match a document opened in the editor. 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 Author website*. Also it is available in the *Oxygen SDK Maven Project*.

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_DIR]/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 XML Author, 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 *Implementing an Author Schema Aware Editing Handler* on page 370 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 Author Eclipse plugin and standalone application. The Text mode corresponding listener is available only for Oxygen XML Author 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. An XML vocabulary can contain links to different areas of a document. In case the document contains elements defined as link you can choose to present a more relevant text description for each link. To do this an implementation of the `ro.sync.ecss.extensions.api.link.LinkTextResolver` interface should be returned by the `createLinkTextResolver` method. This implementation is used each time *the oxy_link-text() function* is encountered in the CSS styles associated with an element.

```
public LinkTextResolver createLinkTextResolver() {
  return new DitaLinkTextResolver();
}
```

Oxygen XML Author offers built in implementations for DITA and DocBook:
`ro.sync.ecss.extensions.dita.link.DitaLinkTextResolver`
`ro.sync.ecss.extensions.docbook.link.DocbookLinkTextResolver`

12. Pack the compiled class into a jar file.

13. Copy the jar file into the `frameworks/sdf` directory.

14. Add the jar file to the Author class path.

15. Register the Java class by clicking on the **Extensions** tab. Press the **Choose** button and select from the displayed dialog the name of the class: `SDFExtensionsBundle`.

> 📋 **Note:** The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen XML Author 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.

Customizing Smart Paste Support

The *smart paste* functionalist preserves certain style and structure information when pasting to certain *document types* from certain common applications.

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 Author website*. Also it is available in the *Author SDK Maven Project*.

```
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorExtensionStateListener;

public class SDFAuthorExtensionStateListener implements
  AuthorExtensionStateListener {
  private AuthorListener sdfAuthorDocumentListener;
  private AuthorMouseListener sdfMouseListener;
  private AuthorCaretListener sdfCaretListener;
  private OptionListener sdfOptionListener;
```

The **activation** event received by this listener when the rules of the Document Type Association match a document opened in the Author editor mode, should be used to perform custom initializations and to register listeners like *ro.sync.ecss.extensions.api.AuthorListener*, *ro.sync.ecss.extensions.api.AuthorMouseListener* or *ro.sync.ecss.extensions.api.AuthorCaretListener*.

```
public void activated(AuthorAccess authorAccess) {
  // Get the value of the option.
  String option = authorAccess.getOptionsStorage().getOption(
          "sdf.custom.option.key", "");
  // Use the option for some initializations...

  // Add an option listener.
  authorAccess.getOptionsStorage().addOptionListener(sdfOptionListener);

  // Add author document listeners.
  sdfAuthorDocumentListener = new SDFAuthorListener();
  authorAccess.getDocumentController().addAuthorListener(
          sdfAuthorDocumentListener);

  // Add mouse listener.
  sdfMouseListener = new SDFAuthorMouseListener();
  authorAccess.getEditorAccess().addAuthorMouseListener(sdfMouseListener);

  // Add caret listener.
  sdfCaretListener = new SDFAuthorCaretListener();
  authorAccess.getEditorAccess().addAuthorCaretListener(sdfCaretListener);

  // Other custom initializations...

}
```

The authorAccess parameter received by the `activated` method can be used to gain access to Author specific actions and informations related to components like the editor, document, workspace, tables, or the change tracking manager.

If options specific to the custom developed Author extension need to be stored or retrieved, a reference to the *ro.sync.ecss.extensions.api.OptionsStorage* can be obtained by calling the `getOptionsStorage` method from the author access. The same object can be used to register *ro.sync.ecss.extensions.api.OptionListener* listeners. An option listener is registered in relation with an option **key** and will be notified about the value changes of that option.

An `AuthorListener` can be used if events related to the Author document modifications are of interest. The listener can be added to the *ro.sync.ecss.extensions.api.AuthorDocumentController*. A reference to the document controller is returned by the `getDocumentController` method from the author access. The document controller can also be used to perform operations involving document modifications.

To provide access to Author editor component related functionality and information, the author access has a reference to the *ro.sync.ecss.extensions.api.access.AuthorEditorAccess* that can be obtained when calling the `getEditorAccess` method. At this level `AuthorMouseListener` and `AuthorCaretListener` can be added which will be notified about mouse and caret events occurring in the Author editor mode.

The **deactivation** event is received when another framework is activated for the same document, the user switches to another editor mode or the editor is closed. The `deactivate` method is typically used to unregister the listeners previously added on the `activate` method and to perform other actions. For example, options related to the deactivated author extension can be saved at this point.

```
public void deactivated(AuthorAccess authorAccess) {
  // Store the option.
```

```
      authorAccess.getOptionsStorage().setOption(
                 "sdf.custom.option.key", optionValue);

      // Remove the option listener.
      authorAccess.getOptionsStorage().removeOptionListener(sdfOptionListener);

      // Remove document listeners.
      authorAccess.getDocumentController().removeAuthorListener(
                 sdfAuthorDocumentListener);

      // Remove mouse listener.
      authorAccess.getEditorAccess().removeAuthorMouseListener(sdfMouseListener);

      // Remove caret listener.
      authorAccess.getEditorAccess().removeAuthorCaretListener(sdfCaretListener);

      // Other actions...

   }
```

Implementing an Author Schema Aware Editing Handler

To implement your own handler for actions like typing, deleting, or pasting, provide an implementation of
ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler. For this handler to be called,
the *Schema Aware Editing option* must be set to **On**,or **Custom**. The handler can either resolve a specific case, let the
default implementation take place, or reject the edit entirely by throwing an InvalidEditException.

> 📄 **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Author website*. Also it is available in the *Oxygen SDK Maven Project*.

```
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-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen
> XML Author 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 Author website*. Also it is available in the *Oxygen SDK Maven Project*.

```
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 Author 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-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen XML Author 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 Author website*. Also it is available in the *Oxygen SDK Maven Project*.

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-sample-framework** project.

Configuring a custom Drag and Drop listener

Sometimes it is useful to perform various operations when certain objects are dropped from outside sources in the editing area. You can choose from three interfaces to implement depending on whether you are using the framework with the Eclipse plugin or the standalone version of the application or if you want to add the handler for the Text or Author modes.

> **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Author website*. Also it is available in the *Oxygen SDK Maven Project*.

Table 7: Interfaces for the Drag and Drop listener

Interface	Description
ro.sync.exml.editor.xmleditor.pageauthor.AuthorDnDListener	Receives callbacks from the standalone application for Drag And Drop in **Author** mode.
com.oxygenxml.editor.editors.author.AuthorDnDListener	Receives callbacks from the Eclipse plugin for Drag And Drop in **Author** mode.
com.oxygenxml.editor.editors.TextDnDListener	Receives callbacks from the Eclipse plugin for Drag And Drop in Text mode.

Configuring a References Resolver

You need to provide a handler for resolving references and obtain the content they refer. In our case the element which has references is **ref** and the attribute indicating the referred resource is **location**. You will have to implement a Java extension class for obtaining the referred resources.

> 📄 **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Author website*. Also it is available in the *Oxygen SDK Maven Project*.

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.getUtilAccess().correctURL(attrStringVal));

          InputSource inputSource = entityResolver.resolveEntity(null,
              absoluteUrl.toString());
          if(inputSource == null) {
            inputSource = new InputSource(absoluteUrl.toString());
          }

          XMLReader xmlReader = authorAccess.newNonValidatingXMLReader();
          xmlReader.setEntityResolver(entityResolver);

          saxSource = new SAXSource(xmlReader, inputSource);
        } catch (MalformedURLException e) {
          logger.error(e, e);
        } catch (SAXException e) {
          logger.error(e, e);
        } catch (IOException e) {
          logger.error(e, e);
        }
      }
    }
  }

  return saxSource;
}
```

5. The method `getReferenceUniqueID` should return an unique identifier for the node reference. The unique identifier is used to avoid resolving the references recursively. The method takes as argument an `AuthorNode` that represents the node with the reference. In the implementation the unique identifier is the value of the *location* attribute from the *ref* element.

```
public String getReferenceUniqueID(AuthorNode node) {
  String id = null;
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        id = attrValue.getValue();
      }
    }
  }
  return id;
}
```

6. The method `getReferenceSystemID`should return the *systemID* of the referred content. It takes as arguments an `AuthorNode` that represents the node with the reference and the `AuthorAccess` with access methods to the Author data model. In the implementation you use the value of the *location* attribute from the *ref* element and resolve it relatively to the XML base URL of the node.

```
public String getReferenceSystemID(AuthorNode node,
                                   AuthorAccess authorAccess) {
  String systemID = null;
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
```

```
      if (attrValue != null) {
        String attrStringVal = attrValue.getValue();
        try {
          URL absoluteUrl = new URL(node.getXMLBaseURL(),
              authorAccess.getUtilAccess().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-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen XML Author website*.

In the listing below, the XML document contains the **ref** element:

```
<ref location="referred.xml">Reference</ref>
```

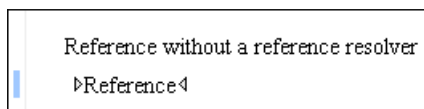When no reference resolver is specified, the reference has the following layout:

Figure 157: Reference with no specified reference resolver

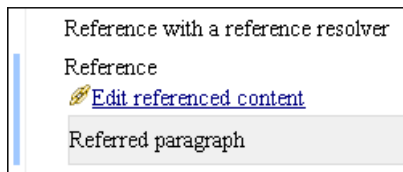When the above implementation is configured, the reference has the expected layout:

Figure 158: 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 Author 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 Author website*. Also it is available in the *Oxygen SDK Maven Project*.

For instance in our simple document example the filter can change the value of the KEY_FONT property for the table element:

```
package simple.documentation.framework;

import ro.sync.ecss.css.Styles;
import ro.sync.ecss.extensions.api.StylesFilter;
import ro.sync.ecss.extensions.api.node.AuthorNode;
import ro.sync.exml.view.graphics.Font;

public class SDFStylesFilter implements StylesFilter {
```

```
     public Styles filter(Styles styles, AuthorNode authorNode) {
         if (AuthorNode.NODE_TYPE_ELEMENT == authorNode.getType()
           && "table".equals(authorNode.getName())) {
           styles.setProperty(Styles.KEY_FONT, new Font(null, Font.BOLD, 12));
         }
         return styles;
     }
}
```

Configuring tables

There are standard CSS properties used to indicate what elements are tables, table rows and table cells. What CSS is missing is the possibility to indicate the cell spanning, row separators or the column widths. Oxygen XML Author Author offers support for adding extensions to solve these problems. This will be presented in the next chapters.

The table in this example is a simple one. The header must be formatted in a different way than the ordinary rows, so it will have a background color.

```
table{
    display:table;
    border:1px solid navy;
    margin:1em;
    max-width:1000px;
    min-width:150px;
}

table[width]{
  width:attr(width, length);
}

tr, header{
    display:table-row;
}

header{
    background-color: silver;
    color:inherit
}

td{
  display:table-cell;
  border:1px solid navy;
  padding:1em;
}
```

Because in the *schema* the td tag has the attributes **row_span** and **column_span** that are not automatically recognized by Oxygen XML Author Author, a Java extension will be implemented which will provide information about the cell spanning. See the section *Configuring a Table Cell Span Provider*.

The column widths are specified by the attributes **width** of the elements customcol that are not automatically recognized by Oxygen XML Author Author. It is necessary to implement a Java extension which will provide information about the column widths. See the section *Configuring a Table Column Width Provider*.

The table from our example does not make use of the attributes colsep and rowsep (which are automatically recognized) but we still want the rows to be separated by horizontal lines. It is necessary to implement a Java extension which will provide information about the row and column separators. See the section *Configuring a Table Cell Row And Column Separator Provider* on page 385.

Configuring a Table Column Width Provider

In the sample documentation framework the table element as well as the table columns can have specified widths. In order for these widths to be considered by Author we need to provide the means to determine them. As explained in the *Configuring tables* on page 379, if you use the table element attribute **width** Oxygen XML Author 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 Author website*. Also it is available in the *Oxygen SDK Maven Project*.

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-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen XML Author website*.

In the listing below, the XML document contains the table element:

```
<table width="300">
    <customcol width="50.0px"/>
    <customcol width="1*"/>
    <customcol width="2*"/>
    <customcol width="20%"/>
    <header>
        <td>C1</td>
        <td>C2</td>
        <td>C3</td>
        <td>C4</td>
    </header>
    <tr>
        <td>cs=1, rs=1</td>
        <td>cs=1, rs=1</td>
        <td row_span="2">cs=1, rs=2</td>
        <td row_span="3">cs=1, rs=3</td>
    </tr>
    <tr>
        <td>cs=1, rs=1</td>
        <td>cs=1, rs=1</td>
    </tr>
    <tr>
        <td column_span="3">cs=3, rs=1</td>
    </tr>
</table>
```

When no table column width provider is specified, the table has the following layout:

Figure 159: Table layout when no column width provider is specified

When the above implementation is configured, the table has the correct layout:

Figure 160: Columns with custom widths

Configuring a Table Cell Span Provider

In the sample documentation framework the `table` element can have cells that span over multiple columns and rows. As explained in *Configuring tables* on page 379, you need to indicate Oxygen XML Author a method to determine the cell spanning. If you use the cell element attributes `rowspan` and `colspan` or `rows` and `cols`, Oxygen XML Author 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 Author website*. Also it is available in the *Oxygen SDK Maven Project*.

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-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen XML Author website*.

6. In the listing below, the XML document contains the table element:

```
<table>
    <header>
        <td>C1</td>
        <td>C2</td>
        <td>C3</td>
        <td>C4</td>
    </header>
    <tr>
        <td>cs=1, rs=1</td>
        <td column_span="2" row_span="2">cs=2, rs=2</td>
        <td row_span="3">cs=1, rs=3</td>
    </tr>
    <tr>
        <td>cs=1, rs=1</td>
    </tr>
    <tr>
        <td column_span="3">cs=3, rs=1</td>
    </tr>
</table>
```

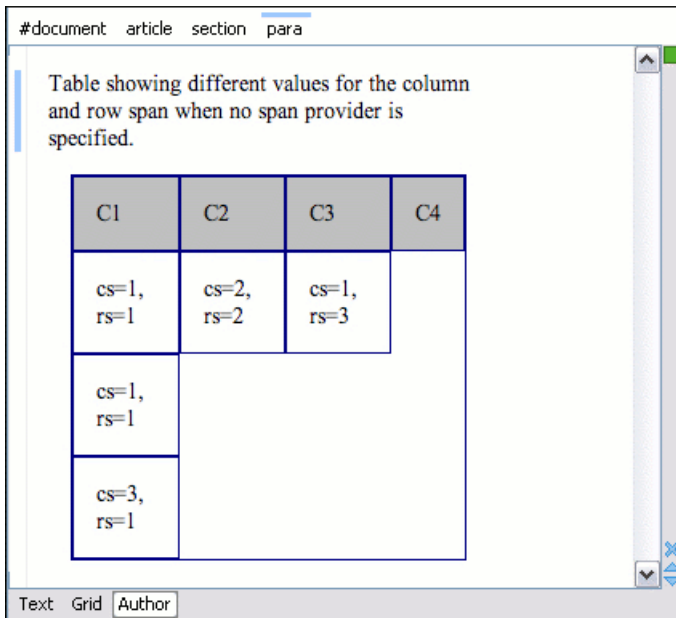When no table cell span provider is specified, the table has the following layout:

Figure 161: Table layout when no cell span provider is specified

When the above implementation is configured, the table has the correct layout:
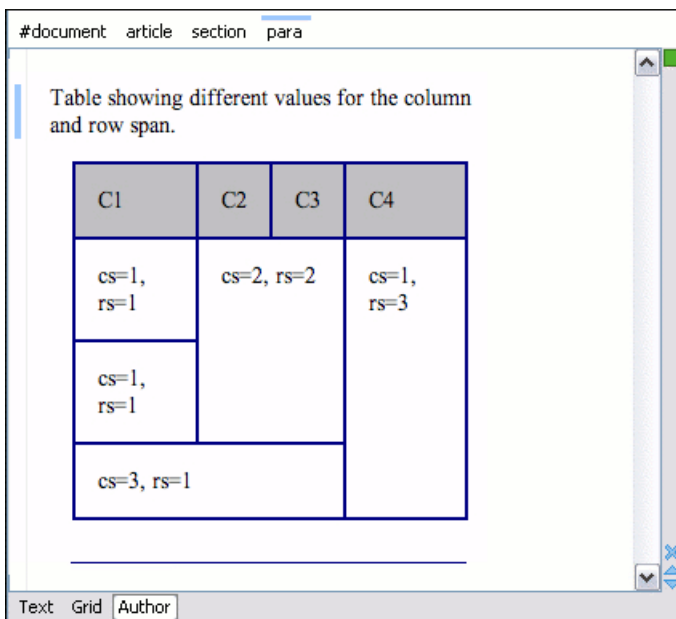

Figure 162: Cells spanning multiple rows and columns.

Configuring a Table Cell Row And Column Separator Provider

In the sample documentation framework the `table` element has separators between rows. As explained in *Configuring tables* on page 379 section which describes the CSS properties needed for defining a table, you need to indicate Oxygen XML Author a method to determine the way rows and columns are separated. If you use the `rowsep` and `colsep` cell element attributes, or your table is conforming to the CALS table model, Oxygen XML Author can determine the cell separators. In the example there are no attributes defining the separators but we still want the rows to be separated. You will need to implement a Java extension.

> **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Author website*. Also it is available in the *Oxygen SDK Maven Project*.

1. Create the class `simple.documentation.framework.TableCellSepProvider`. This class must implement the *`ro.sync.ecss.extensions.api.AuthorTableCellSepProvider`* interface.

```
import ro.sync.ecss.extensions.api.AuthorTableCellSepProvider;
import ro.sync.ecss.extensions.api.node.AuthorElement;

public class TableCellSepProvider implements AuthorTableCellSepProvider{
```

2. The `init` method is taking as argument the *`ro.sync.ecss.extensions.api.node.AuthorElement`* that represents the XML `table` element. In our case the separator information is implicit, it does not depend on the current table, so you leave this method empty. However there are cases like the table CALS model when the cell separators are specified in the `table` element - in that case you should initialize your provider based on the given argument.

```
public void init(AuthorElement table) {
}
```

3. The `getColSep` method is taking as argument the table cell. The table layout engine will ask this `AuthorTableCellSepProvider` implementation if there is a column separator for each XML element from the table that was marked as cell in the CSS using the property `display:table-cell`. In our case we choose to return **false** since we do not need column separators.

```
/**
 * @return false - No column separator at the right of the cell.
 */
@Override
public boolean getColSep(AuthorElement cellElement, int columnIndex) {
  return false;
}
```

4. The row separators are determined in a similar manner. This time the method returns **true**, forcing a separator between the rows.

```
/**
 * @return true - A row separator below each cell.
 */
@Override
public boolean getRowSep(AuthorElement cellElement, int columnIndex) {
  return true;
}
```

> **Note:** The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen XML Author website*.

5. In the listing below, the XML document contains the table element:

```
<table>
    <header>
      <td>H1</td>
      <td>H2</td>
      <td>H3</td>
      <td>H4</td>
    </header>
    <tr>
      <td>C11</td>
      <td>C12</td>
      <td>C13</td>
      <td>C14</td>
    </tr>
    <tr>
      <td>C21</td>
      <td>C22</td>
      <td>C23</td>
      <td>C24</td>
    </tr>
    <tr>
      <td>C31</td>
      <td>C32</td>
      <td>C33</td>
      <td>C34</td>
    </tr>
  </table>
```

When the borders for the `td` element are removed from the CSS, the row separators become visible:

H1	H2	H3	H4
C11	C12	C13	C14
C21	C22	C23	C24
C31	C32	C33	C34

Figure 163: Row separators provided by the Java implementation.

Configuring an Unique Attributes Recognizer

The *ro.sync.ecss.extensions.api.UniqueAttributesRecognizer* interface can be implemented if you want to provide for your framework the following features:

> **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Author website*. Also it is available in the *Oxygen SDK Maven Project*.

* **Automatic ID generation** - You can automatically generate unique IDs for newly inserted elements. Implementations are already available for the DITA and DocBook frameworks. The following methods can be implemented to accomplish this: `assignUniqueIDs(int startOffset, int endOffset)`, `isAutoIDGenerationActive()`
* **Avoiding copying unique attributes when "Split" is called inside an element** - You can split the current block element by pressing the "Enter" key and then choosing "Split". This is a very useful way to create new paragraphs, for example. All attributes are by default copied on the new element but if those attributes are IDs you sometimes want to avoid creating validation errors in the editor. Implementing the following method, you can decide whether an attribute should be copied or not during the split: `boolean copyAttributeOnSplit(String attrQName, AuthorElement element)`

  > **Tip:**
  >
  > The *ro.sync.ecss.extensions.commons.id.DefaultUniqueAttributesRecognizer* class is an implementation of the interface which can be extended by your customization to provide easy assignation of IDs in your framework. You can also check out the DITA and DocBook implementations of `ro.sync.ecss.extensions.api.UniqueAttributesRecognizer` to see how they were implemented and connected to the extensions bundle.

Configuring an XML Node Renderer Customizer

You can use this API extension to customize the way an XML node is rendered in the **Author Outline** view, **Author** breadcrumb navigation bar, **Text** mode **Outline** view, content completion assistant window or **DITA Maps Manager** view.

> **Note:** Oxygen XML Author uses XMLNodeRendererCustomizer implementations for the following frameworks: DITA, DITAMap, DocBook 4, DocBook 5, TEI P4, TEI P5, XHTML, XSLT, and XML Schema.

> **Note:** The Javadoc documentation of the Author API used in the example files is *available on the Oxygen XML Author website*. Also it is available in the *Oxygen SDK Maven Project*.

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.

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 `[OXYGEN_DIR]/frameworks/dita/css_classed` where the default stylesheet called `dita.css` is located. 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, *open the **Preferences** dialog* and go to **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 `${framework}/css_classed/dita.css` to `${framework}/css_classed/my_dita.css`.
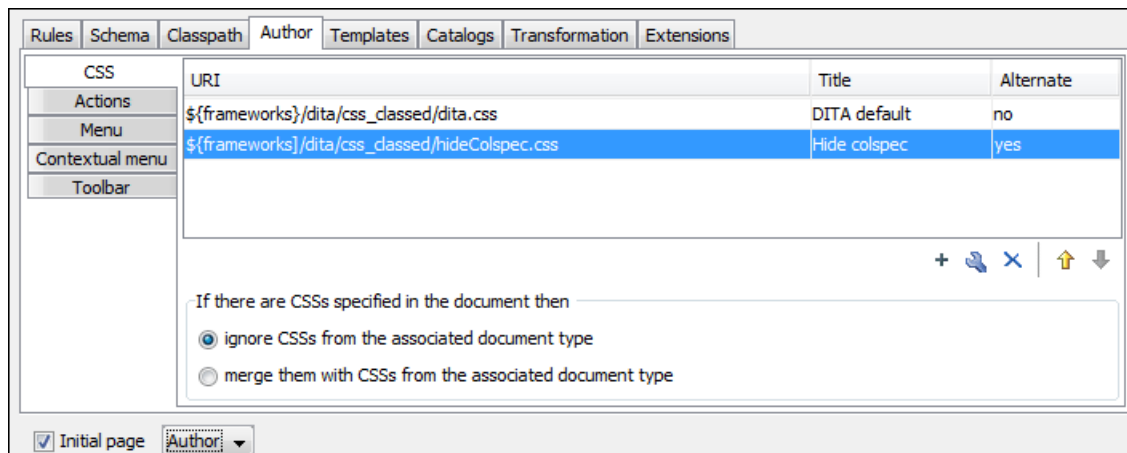

Figure 164: Set the location of the default CSS stylesheet

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 XML Author allows you to share the customizations for a specific XML type by creating your own *Document Type* in the **Document Type Association** preferences page.

A document type can be shared between authors as follows:

- Save it externally in a separate framework folder in the `[OXYGEN_DIR]/frameworks` directory.

  ⚠ **Important:** For this approach to work, have the application installed to a folder with full write access.

Please follow these steps:

1. Go to [OXYGEN_DIR]/frameworks and create a directory for your new framework (name it for example custom_framework). This directory will contain resources for your framework (CSS files, new file templates, schemas used for validation, catalogs). See the **Docbook** framework structure from the [OXYGEN_DIR]/frameworks/docbook as an example.

2. Create your custom document type and save it externally, in the custom_framework directory.

3. Configure the custom document type according to your needs, take special care to make all file references relative to the [OXYGEN_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_DIR]/frameworks/custom_framework/custom.framework after the Preferences are saved.

5. Then, to share the new framework directory with other users, have them copy it to their [OXYGEN_DIR]/frameworks directory. The new document type will be available in the list of Document Types when Oxygen XML Author starts.

> 📝 **Note:** In case you have a frameworks directory stored on your local drive, you can also go to the **Document Type Association** > **Locations** preferences page and add your frameworks directory in the **Additional frameworks directories** list.

- Save the document type at project level in the **Document Type Association** preferences page.

  Please see the following steps:

  1. On your local drive, create a directory with full write access, containing the Oxygen XML Author project file and associated document type resources (CSS files, new file templates, schemas used for validation, catalogs).

  2. Start Oxygen XML Author, go to the ***Project view*** and create a project. Save it in the newly created directory.

  3. In the **Document Type Association** preferences page, select **Project Options** at the bottom of the page.

  4. Create your custom document type using the default **internal** storage for it. It will actually be saved in the previously chosen Oxygen XML Author 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. When they open the customized project file in the ***Project view***, the new document type becomes available in the list of Document Types.

- Deploy your document type configuration *as an add-on*.

Adding Custom Persistent Highlights

The Author API allows you to create or remove custom persistent highlights, set their properties, and customize their appearance. They get serialized in the XML document as processing instructions, having the following format:

```
<?oxy_custom_start prop1="val1"....?> xml content <?oxy_custom_end?>
```

The functionality is available in the AuthorPersistentHighlighter class, accessible through AuthorEditorAccess#getPersistentHighlighter() method. For more information, see JavaDoc online at: *http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/index.html*

CSS Support in Author

Author editing mode supports most CSS 2.1 selectors, a lot of CSS 2.1 properties and a few CSS 3 selectors. Also some custom functions and properties that extend the W3C CSS specification and are useful for URL and string manipulation are available to the developer who creates an Author editing framework.

Handling CSS Imports

When a CSS document contains imports to other CSS documents, the references are also passed through the XML catalog URI mappings in order to determine an indirect CSS referenced location.

> You can have a CSS import like:
>
> ```
> @import "http://host/path/to/location/custom.css";
> ```
>
> and then add to the *XML / XML Catalog* preferences page your own XML catalog file which maps the location to a custom CSS:
>
> ```
> <uri name="http://host/path/to/location/custom.css" uri="path/to/custom.css"/>
> ```

In addition to this, you can also add in your XML Catalog file the following mapping:

```
<uri name="http://www.oxygenxml.com/extensions/author/css/userCustom.css" uri="path/to/custom.css"/>
```

This extra mapped CSS location will be parsed every time the application processes the CSS stylesheets used to render the opened XML document in the visual **Author** editing mode. In this way your custom CSS can be used without the need to modify all other CSS stylesheets contributed in the document type configuration.

Media Type `oxygen`

The CSS stylesheets can specify how a document is presented on different media: on the screen, on paper, on speech synthesizer, etc. You can specify that some of the selectors from your CSS should be taken into account only in the **Author** mode and ignored in the rest. This can be accomplished by using the oxygen media type.

For instance using the following CSS:

```
b{
 font-weight:bold;
 display:inline;
}

@media oxygen{
 b{
  text-decoration:underline;
 }
}
```

would make a text bold if the document was opened in a web browser that does not recognize @media oxygen and bold and underlined in Oxygen XML Author Author.

You can use this media type to group specific Oxygen XML Author CSS selectors and have them ignored when opening the documents with other viewers.

Standard W3C CSS Supported Features

Oxygen XML Author supports most of the CSS Level 3 selectors and most of the CSS Level 2.1 properties

Supported CSS Selectors

Expression	Name	CSS Level	Description / Example
*	Universal selector	CSS Level 2	Matches any element
E	Type selector	CSS Level 2	Matches any E element (i. e. an element with the local name E)
E F	Descendant selector	CSS Level 2	Matches any F element that is a descendant of an E element.

Expression	Name	CSS Level	Description / Example
`E > F`	Child selectors	CSS Level 2	Matches any `F` element that is a child of an element `E`.
`E:lang(c)`	Language pseudo-class	CSS Level 2	Matches element of type `E` if it is in (human) language `c` (the document language specifies how language is determined).
`E + F`	Adjacent selector	CSS Level 2	Matches any `F` element immediately preceded by a sibling element `E`.
`E ~ F`	General sibling selector	CSS Level 3	Matches any `F` element preceded by a sibling element `E`.
`E[foo]`	Attribute selector	CSS Level 2	Matches any `E` element with the `"foo"` attribute set (whatever the value).
`E[foo="warning"]`	Attribute selector with value	CSS Level 2	Matches any `E` element whose `"foo"` attribute value is exactly equal to `"warning"`.
`E[foo~="warning"]`	Attribute selector containing value	CSS Level 2	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 containing hyphen separated values	CSS Level 2	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	CSS Level 2	The `':before'` and `':after'` pseudo-elements can be used to insert generated content before or after an element's content.
`E:first-child`	The first-child pseudo-class	CSS Level 2	Matches element `E` when `E` is the first child of its parent.
`E:not(s)`	Negation pseudo-class	CSS Level 2	An E element that does not match simple selector s.
`E:hover`	The hover pseudo-class	CSS Level 2	The `:hover` pseudo-class applies while the user designates an element with a pointing device, but does not necessarily activate it. When moving the pointing device over an element, all the parent elements up to the root are taken into account.
`E:focus`	The focus pseudo-class	CSS Level 2	The `:focus` pseudo-class applies while an element has the focus (accepts keyboard input).
`E#myid`	The ID selector	CSS Level 2	Matches any `E` element with ID equal to "myid". ⚠️ **Important:** Limitation: In Oxygen XML Author the match is performed taking into account

Expression	Name	CSS Level	Description / Example
			only the attributes with the exact name: "id".
`E[att^="val"]`	Substring matching attribute selector	CSS Level 3	An E element whose `att` attribute value begins exactly with the string `val`.
`E[att$="val"]`	Substring matching attribute selector	CSS Level 3	An E element whose `att` attribute value ends exactly with the string `val`.
`E[att*="val"]`	Substring matching attribute selector	CSS Level 3	An E element whose `att` attribute value contains the substring `val`.
`E:root`	Root pseudo-class	CSS Level 3	Matches the root element of the document. In HTML, the root element is always the HTML element.
`E:empty`	Empty pseudo-class	CSS Level 3	An E element which has no text or child elements.
`E:nth-child(n)`	The nth-child pseudo-class	CSS Level 3	An E element, the nth child of its parent.
`E:nth-last-child(n)`	The nth-last-child pseudo-class	CSS Level 3	An E element, the nth child of its parent, counting from the last one.
`E:nth-of-type(n)`	The nth-of-type pseudo-class	CSS Level 3	An E element, the nth sibling of its type.
`E:nth-last-of-type(n)`	The nth-last-of-type pseudo-class	CSS Level 3	An E element, the nth sibling of its type, counting from the last one.
`E:last-child`	The last-child pseudo-class	CSS Level 3	An E element, last child of its parent.
`E:first-of-type`	The first-of-type pseudo-class	CSS Level 3	An E element, first sibling of its type.
`E:last-of-type`	The last-of-type pseudo-class	CSS Level 3	An E element, last sibling of its type.
`E:only-child`	The only-child pseudo-class	CSS Level 3	An E element, only child of its parent.
`E:only-of-type`	The only-of-type pseudo-class	CSS Level 3	An E element, only sibling of its type.
`ns|E`	Element namespace selector	CSS Level 3	An element that has the local name E and the namespace given by the prefix `ns`. The namespace prefix can be bound to an URI by the at-rule: `@namespace ns "http://some_namespace_uri";` See *Namespace Selector* on page 392.
`E!>F`	The subject selector	CSS Level 4 (experimental)	An element that has the local name E and has a child F. See *Subject Selector* on page 394.

Namespace Selector

In the CSS 2.1 standard the element selectors are ignoring the namespaces of the elements they are matching. Only the local name of the elements are considered in the selector matching process.

Oxygen XML Author 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.

Defining prefixed namespaces combined with pseudo-elements

To match the `def` element its namespace will be declared, bind it to the *abs* prefix, and then write a CSS rule:

```
@namespace abs "http://www.oxygenxml.com/sample/documentation/abstracts";
```

Then:

abs|def

represents the name "def" in the `http://www.oxygenxml.com/sample/documentation/abstracts` namespace.

abs|def:before

represents the :before pseudo-element of the "def" element from the `http://www.oxygenxml.com/sample/documentation/abstracts` namespace.

Subject Selector

Oxygen XML Author Author supports the subject selector described in CSS Level 4 (currently a working draft at W3C *http://dev.w3.org/csswg/selectors4/*). This selector matches a structure of the document, but unlike a compound selector, the styling properties are applied to the subject element (the one marked with "!") instead of the last element from the path.

The subject of the selector can be explicitly identified by appending an exclamation mark (!) to one of the compound selectors in a selector. Although the element structure that the selector represents is the same with or without the exclamation mark, indicating the subject in this way can change which compound selector represents the subject in that structure.

```
table! > caption {
  border: 1px solid red;
}
```

A border will be drawn to the table elements that contain a caption as direct child.

This is different from:

```
table > caption {
  border: 1px solid red;
}
```

which draws a border around the caption.

⚠ **Important:** As a limitation of the current implementation the general descendant selectors are taken into account as direct child selectors. For example the two CSS selectors are considered equivalent:

```
a! b c
```

and:

```
a! > b > c
```

Supported CSS Properties

Oxygen XML Author validates all CSS 2.1 properties, but does not render in Author mode *aural* and *paged* categories properties, as well as some of the values of the *visual* category, listed below under the **Ignored Values** column. For the Oxygen XML Author specific (extension) CSS properties, go to *Oxygen XML Author CSS Extensions* on page 402.

Name	Rendered Values	Ignored Values
`'background-attachment'`		ALL
`'background-color'`	`<color> | inherit`	`transparent`
`'background-image'`	`<uri> | none | inherit`	
`'background-position'`	`top | right | bottom | left | center`	`<percentage> | <length>`
`'background-repeat'`	`repeat | repeat-x | repeat-y | no-repeat | inherit`	
`'background'`		ALL
`'border-collapse'`		ALL
`'border-color'`	`<color> | inherit`	`transparent`
`'border-spacing'`		ALL
`'border-style'`	`<border-style> | inherit`	
`'border-top' 'border-right' 'border-bottom' 'border-left'`	`[ <border-width> || <border-style> || 'border-top-color' ] | inherit`	
`'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> ) |`	`no-open-quote | no-close-quote`

Name	Rendered Values	Ignored Values
	open-quote | close-quote]+ | inherit	
'counter-increment'	[<identifier> <integer> ?]+ | none | inherit	
'counter-reset'	[<identifier> <integer> ?]+ | none | inherit	
'cursor'		ALL
'direction'	ltr| rtl | inherit	
'display'	inline | block | list-item | table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit	run-in | inline-block | inline-table - considered block
'empty-cells'	show | hide | inherit	
'float'		ALL
'font-family'	[[<family-name> | <generic-family>] [, <family-name> | <generic-family>]*] | inherit	
'font-size'	<absolute-size> | <relative-size> | <length> | <percentage> | inherit	
'font-style'	normal | italic | oblique | inherit	
'font-variant'		ALL
'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	

Name	Rendered Values	Ignored Values
`'list-style-image'`		ALL
`'list-style-position'`		ALL
`'list-style-type'`	`disc	circle	square	decimal	lower-roman	upper-roman	lower-latin	upper-latin	lower-alpha	upper-alpha	-oxy-lower-cyrillic-ru	-oxy-lower-cyrillic-uk	-oxy-upper-cyrillic-ru	-oxy-upper-cyrillic-uk	box	diamond	check	hyphen	none	inherit`	`lower-greek	armenian	georgian`
`'list-style'`	`['list-style-type']	inherit`	`'list-style-position'		'list-style-image'`
`'margin-right' 'margin-left'`	`<margin-width>	inherit	auto`	
`'margin-top' 'margin-bottom'`	`<margin-width>	inherit`	
`'margin'`	`<margin-width>	inherit	auto`	
`'max-height'`		ALL
`'max-width'`	`<length>	<percentage>	none	inherit` - supported for inline, block-level, and replaced elements, e.g. images, tables, table cells.	
`'min-height'`		ALL
`'min-width'`	`<length>	<percentage>	inherit` - supported for inline, block-level, and replaced elements, e. g. images, tables, table cells.	
`'outline-color'`		ALL
`'outline-style'`		ALL
`'outline-width'`		ALL
`'outline'`		ALL
`'overflow'`		ALL
`'padding-top' 'padding-right' 'padding-bottom' 'padding-left'`	`<padding-width>	inherit`	
`'padding'`	`<padding-width>	inherit`	
`'position'`		ALL
`'quotes'`		ALL

Name	Rendered Values	Ignored Values
`'right'`		ALL
`'table-layout'`	`auto`	`fixed	inherit`
`'text-align'`	`left	right	center	inherit`	`justify`
`'text-decoration'`	`none	[underline		overline		line-through]	inherit`	`blink`
`'text-decoration-style'`	`solid	double	dotted	dashed	wavy	initial	inherit`	
`'text-indent'`		ALL
`'text-transform'`	ALL	
`'top'`		ALL
`'unicode-bidi'`	`bidi-override| normal| embed| inherit`	
`'vertical-align'`	`baseline	sub	super	top	text-top	middle	bottom	text-bottom	inherit`	`<percentage>	<length>`
`'visibility'`	`visible	hidden	inherit	-oxy-collapse-text`	`collapse`
`'white-space'`	`normal	pre	nowrap	pre-wrap	pre-line`	
`'width'`	`<length>	<percentage>	auto	inherit` - supported for inline, block-level, and replaced elements, e.g. images, tables, table cells.	
`'word-spacing'`		ALL
`'z-index'`		ALL

Transparent Colors

CSS3 supports RGBA colors. The RGBA declaration allows you to set opacity (via the Alpha channel) as part of the color value. A value of `0` corresponds to a completely transparent color, while a value of `1` corresponds to a completely opaque color. To specify a value, you can use either a *real* number between `0` and `1`, or a percent.

RGBA color

```
personnel:before {
    display:block;
    padding: 1em;
    font-size: 1.8em;
    content: "Employees";
    font-weight: bold;
    color:#EEEEEE;
    background-color: rgba(50, 50, 50, 0.6);
}
```

The `attr()` Function: Properties Values Collected from the Edited Document.

In CSS Level 2.1 you may collect attribute values and use them as content *only* for the pseudo-elements. For instance the :before pseudo-element can be used to insert some content before an element. This is valid in CSS 2.1:

```
title:before{
  content: "Title id=(" attr(id) ")";
}
```

If the `title` element from the XML document is:

```
<title id="title12">My title.</title>
```

Then the title will be displayed as:

```
                    Title id=(title12) My title.
```

In Oxygen XML Author 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 Author 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 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 Author 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:
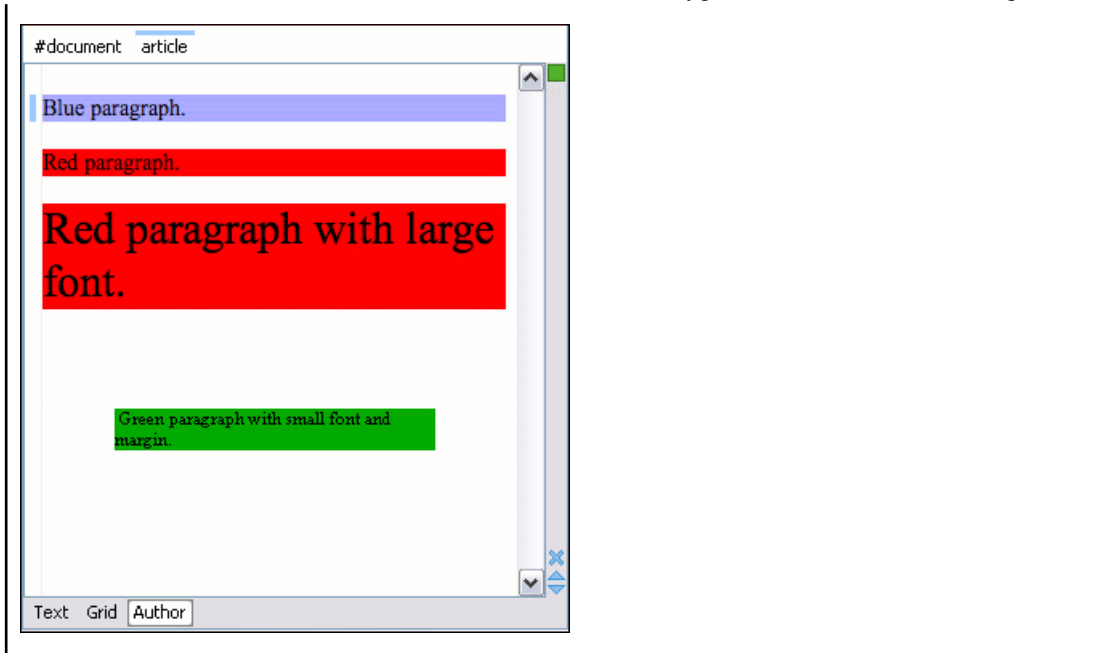
Supported CSS At-rules

Oxygen XML Author supports some of the at-rules specified by CSS Level 2.1 and 3.

The @font-face at-rule

Oxygen XML Author allows you to use custom fonts in the **Author** mode by specifying them in the CSS using the `@font-face` media type. Only the `src` and `font-family` CSS properties can be used for this media type.

```
@font-face{
    font-family:"Baroque Script";
    /*The location of the loaded TTF font must be relative to the CSS*/
    src:url("BaroqueScript.ttf");
}
```

The @media at-rule

Oxygen XML Author supports several media types, allowing you to set different styles for presenting a document on different media (on the screen, on paper and so on). The following media types are supported:

- `screen` - the styles marked with this media type are used only for rendering a document in the **Author** mode
- `all` - the styles marked with this media type are used for rendering a document in the **Author** mode and also for printing the document
- `oxygen` - the styles marked with this media type are used only for rendering a document in the **Author** mode

    Note: This is an Oxygen XML Author specific media

- `oxygen-high-contrast-black` - the styles marked with this media type are used only for rendering a document in the **Author** mode, on a Windows black background High Contrast Theme

    Note: This is an Oxygen XML Author specific media

- `oxygen-high-contrast-white` - the styles marked with this media type are used only for rendering a document in the **Author** mode, on a Windows white background High Contrast Theme

    Note: This is an Oxygen XML Author specific media

- `print` - the styles marked with this media type are used only for printing a document

Oxygen XML Author CSS Extensions

CSS stylesheets provide support mainly for displaying documents. When editing documents some non-standard, Oxygen XML Author CSS extensions are useful, for example:

- property for marking foldable elements in large files
- enforcing a display mode for the XML tags regardless of the current mode selected by the author user
- construct an URL from a relative path location
- string processing functions

Additional CSS Selectors

Oxygen XML Author Author provides support for selecting additional types of nodes. These custom selectors apply to: *document*, *doctype sections*, *processing-instructions*, *comments*, *CDATA sections*, *reference sections*, and *entities*. *Processing-instructions* are not displayed by default. To display them, *open the **Preferences** dialog* ,go to **Editor** > **Author**, and select **Show processing instructions**.

> 📝 **Note:** The custom selectors are presented in the default CSS for **Author** mode and all of their properties are marked with an *!important* flag. For this reason, you have to set the *!important* flag on each property of the custom selectors from your CSS to be applicable.

For the custom selectors to work in your CSS stylesheets, declare the Author extensions namespace at the beginning of the stylesheet documents:

```
@namespace oxy url('http://www.oxygenxml.com/extensions/author');
```

- The *oxy|document* selector matches the entire document:

```
oxy|document {
    display:block !important;
}
```

- The following example changes the rendering of doctype sections:

```
oxy|doctype {
    display:block !important;
    color:blue !important;
    background-color:transparent !important;
}
```

- To match the processing instructions, you can use the *oxy|processing-instruction* selector:

```
oxy|processing-instruction {
    display:block !important;
    color:purple !important;
    background-color:transparent !important;
}
```

A processing instruction usually has a target and one or more pseudo attributes:

```
<?target_name data="b"?>
```

You can match a processing instruction with a particular target from the CSS using the construct:

```
oxy|processing-instruction[target_name]
```

You can also match the processing instructions having a certain target and pseudo attribute value like:

```
oxy|processing-instruction[target_name][data="b"]
```

- The XML comments display in Author mode can be changed using the *oxy|comment* selector:

```
oxy|comment {
    display:block !important;
    color:green !important;
    background-color:transparent !important;
}
```

- The *oxy|cdata* selector matches CDATA sections:

```
oxy|cdata{
    display:block !important;
    color:gray !important;
    background-color:transparent !important;
}
```

- The *oxy|entity* selector matches the entities content:

```
oxy|entity {
    display:morph !important;
    editable:false !important;
    color:orange !important;
    background-color:transparent !important;
}
```

- The *references to entities*, *XInclude*, and *DITA conrefs* and *conkeyrefs* are expanded by default in Author mode and the referenced content is displayed. The referenced resources are displayed inside the element or entity that refers to 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 the **Author** mode, content is highlighted when parts of text contain:

  - *comments*.
  - changes and *Track Changes* was active when the content was modified.

  If this content is referred, the **Author** mode does not display the highlighted areas in the new context. If you want to mark the existence of this comments and changes you can use the *oxy|reference[comments]*, *oxy|reference[changeTracking]*, and *oxy|reference[changeTracking][comments]* selectors.

  > 📋 **Note:** Two artificial attributes (*comments* and *changeTracking*) are set on the reference node, containing information about the number of comments and track changes in the content.

  - The following example represents the customization of the reference fragments that contain comments:

    ```
    oxy|reference[comments]:before {
      content: "Comments: " attr(comments) !important;
    }
    ```

  - To match reference fragments based on the fact that they contain change tracking inside, use the *oxy|reference[changeTracking]* selector.

    ```
    oxy|reference[changeTracking]:before {
      content: "Change tracking: " attr(changeTracking) !important;
    }
    ```

  - Here is an example of how you can set a custom color to the reference containing both track changes and comments:

    ```
    oxy|reference[changeTracking][comments]:before {
      content: "Change tracking: " attr(changeTracking) " and comments: " attr(comments) !important;
    }
    ```

A sample document rendered using these rules:

```
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.
```

Additional CSS Properties

Oxygen XML Author Author offers an extension of the standard CSS properties suited for content editing.

Folding Elements: `-oxy-foldable`, `-oxy-not-foldable-child` and `-oxy-folded` properties

Oxygen XML Author 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 Author 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:

- **Close Other Folds Ctrl+NumPad +/ (Command+NumPad+/ on OS X)** - Folds all the elements except the current element.
- **Collapse Child Folds (Ctrl+Decimal)** - Folds the elements indented with one level inside the current element.
- **Expand Child Folds (Ctrl+Equals)**- Unfolds all child elements of the currently selected element.
- **Expand All (Ctrl+NumPad+*)** - Unfolds all elements in the current document.
- **Toggle Fold** - 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 element that is kept visible. It accepts as value an element name or a list of comma separated element names. The first child element from the XML document that appears in the list of element names will be identified as the not foldable child and displayed. If the element is

marked as *foldable* (`-oxy-foldable:true;`) but it doesn't have the property `-oxy-not-foldable-child` or none of the specified non-foldable children exists, then the element is still foldable. In this case the element kept visible when folded will be the `before` pseudo-element.

📄 **Note:** Deprecated properties `foldable`, `not-foldable-child`, and `folded` are also supported.

Folding DocBook Elements

All the elements below can have a `title` child element and are considered to be logical sections. You mark them as being *foldable* leaving the `title` element visible.

```
set,
book,
part,
reference,
chapter,
preface,
article,
sect1,
sect2,
sect3,
sect4,
section,
appendix,
figure,
example,
table {
    -oxy-foldable:true;
    -oxy-not-foldable-child: title;
}
```

Placeholders for empty elements: `-oxy-show-placeholder` and `-oxy-placeholder-content` properties

Oxygen XML Author 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 in 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 Author allows you to specify that an element has an `-oxy-morph` display type (deprecated `morph` property is also supported), meaning that the element is inline if all its children are inline.

Let's suppose we have a **wrapper** XML element allowing users to set a number of attributes on all sub-elements. This element should have an inline or block behavior depending on the behavior of its child elements:

```
wrapper{
 display:-oxy-morph;
}
```

The `whitespace` property: `-oxy-trim-when-ws-only` value

Oxygen XML Author Author allows you to set the `whitespace` property to `-oxy-trim-when-ws-only`, meaning that the leading and trailing whitespaces are removed.

The `visibility` property: -oxy-collapse-text

Oxygen XML Author Author allows you to set the value of the `visibility` property to `-oxy-collapse-text`, meaning that the text content of that element is not rendered. If an element is marked as `-oxy-collapse-text` you are not able to position the caret inside it and edit it. The purpose of `-oxy-collapse-text` is to make the text value of an element editable only through a form control.

> The text value of an XML element will be edited using a text field form control. In this case, we want the text content not to be directly present in the Author visual editing mode:
>
> ```
> title{
>  content: oxy_textfield(edit, '#text', columns, 40);
>  visibility:-oxy-collapse-text;
> }
> ```

Cyrillic Counters: `list-style-type` values `-oxy-lower-cyrillic`

Oxygen XML Author Author allows you to set the value of the `list-style-type` property to `-oxy-lower-cyrillic-ru`, `-oxy-lower-cyrillic-uk`, `-oxy-upper-cyrillic-ru` or `-oxy-upper-cyrillic-uk`, meaning that you can have Russian and Ukrainian counters.

> Counting list items with Cyrillic symbols:
>
> ```
> li{
>    display:list-item;
>    list-style-type:-oxy-lower-cyrillic-ru;
> }
> ```

The link property: link

Oxygen XML Author Author allows you to declare some elements to be *links*. This is especially useful when working with many documents which refer each other. The links allow for an easy way to get from one document to another. Clicking on the link marker will open the referred resource in an editor.

To define the element which should be considered a link, you must use the property `link` on the before or after pseudo element. The value of the property indicates the location of the linked resource. Since links are usually indicated by the value of an attribute in most cases it will have a value similar to `attr(href)`

> **DocBook Link Elements**
>
> All the elements below are defined to be links on the before pseudo element and their value is defined by the value of an attribute.
>
> ```
> *[href]:before{
>     link:attr(href);
>     content: "Click " attr(href) " for opening" ;
> }
>
> ulink[url]:before{
>     link:attr(url);
>     content: "Click to open: " attr(url);
> }
>
> olink[targetdoc]:before{
>     -oxy-link: attr(targetdoc);
>     content: "Click to open: " attr(targetdoc);
> }
> ```

Display Tag Markers: -oxy-display-tags

Oxygen XML Author Author allows you to choose whether tag markers of an element should never be presented or the current display mode should be respected. This is especially useful when working with :before and :after pseudo-elements in which case the element range is already visually defined so the tag markers are redundant.

The property is named -oxy-display-tags, with the following possible values:

- *none* - Tags markers must not be presented regardless of the current *Display mode.*.

- *default* - The tag markers will be created depending on the current *Display mode.*.

- *inherit* - The value of the property is inherited from an ancestor element.

```
-oxy-display-tags
    Value: none | default | inherit
    Initial: default
    Applies to: all nodes(comments, elements, CDATA, etc)
    Inherited: false
    Media: all
```

DocBook Para elements

In this example the **para** element from DocBook is using an :before and :after element so you don't want its tag markers to be visible.

```
para:before{
    content: "{";
}

para:after{
    content: "}";
}

para{
    -oxy-display-tags: none;
    display:block;
    margin: 0.5em 0;
}
```

Append Content Properties: `-oxy-append-content` and `-oxy-prepend-content`

The `-oxy-append-content` Property

This property appends the specified content to the content generated by other matching CSS rules of lesser specificity. Unlike the content property, where only the value from the rule with the greatest specificity is taken into account, the -oxy-append-conent property adds content to that generated by the lesser specificity rules into a new compound content.

`-oxy-append-content` Example

```
element:before{
    content: "Hello";
}
element:before{
    -oxy-append-content: " World!";
}
```

The content shown before the element will be Hello World!.

The `-oxy-prepend-content` Property

Prepends the specified content to the content generated by other matching CSS rules of lesser specificity. Unlike the content property, where only the value from the rule with the greatest specificity is taken into account, the -oxy-prepend-conent prepends content to that generated by the lesser specificity rules into a new compound content.

-oxy-prepend-content Example

```
element:before{
    content: "Hello!";
}
element:before{
    -oxy-prepend-content: "said: ";
}
element:before{
    -oxy-prepend-content: "I ";
}
```

The content shown before the element will be I said: Hello!.

Custom colors for element tags: -oxy-tags-color and -oxy-tags-background-color

By default Oxygen XML Author does not display element tags. You can use this button ᴾ▷◁ from the **Author** tool bar to control the amount of *displayed markup.*

To configure the default background and foreground colors of the tags, go to **Editor** > **Edit modes** > **Author**. The -oxy-tags-background-color and -oxy-tags-color properties allow you to control the background and foreground colors for any particular XML element.

```
para {
    -oxy-tags-color:white;
    -oxy-tags-background-color:green;
}
title {
    -oxy-tags-color:yellow;
    -oxy-tags-background-color:black;
}
```

Custom CSS Functions

The visual Author editing mode supports also a wide range of custom CSS extension functions.

The oxy_local-name() Function

The oxy_local-name() function evaluates the local name of the current node. It does not have any arguments.

To insert as static text content before each element its local name, use this CSS selector:

```
*:before{
  content: oxy_local-name() ": ";
}
```

The oxy_name() Function

The oxy_name() function evaluates the qualified name of the current node. It does not have any arguments.

To insert as static text content before each element its qualified name, use this CSS selector:

```
*:before{
  content: oxy_name() ": ";
}
```

The oxy_url() Function

The oxy_url() function extends the standard CSS **url()** function, by allowing you to specify additional relative path components (parameters **loc_1** to **loc_n**). Oxygen XML Author uses all these parameters to construct an absolute location.

oxy_url(location , loc_1 , loc_2)

location

    The location as string. If not absolute, will be solved relative to the CSS file URL.

loc_1 ... loc_n

Relative location path components as string. (optional)

> The following function:
>
> ```
> oxy_url('http://www.oxygenxml.com/css/test.css', '../dir1/', 'dir2/dir3/',
> '../../dir4/dir5/test.xml')
> ```
>
> returns
>
> ```
> 'http://www.oxygenxml.com/dir1/dir4/dir5/test.xml'
> ```
>
> As a concrete example if you have image references but you want to see in the visual Author editing mode thumbnail images which reside in the same folder:
>
> ```
> image[href]{
>  content:oxy_url(oxy_base-uri(), oxy_replace(attr(href), '.jpeg', 'Thumbnail.jpeg'));
> }
> ```

The `oxy_base-uri()` Function

The `oxy_base-uri()` function evaluates the base URL in the context of the current node. It does not have any arguments and takes into account the `xml:base` context of the current node. See the *XML Base specification* for more details.

> If you have image references but you want to see in the visual Author editing mode thumbnail images which reside in the same folder:
>
> ```
> image[href]{
>  content:oxy_url(oxy_base-uri(), oxy_replace(attr(href), '.jpeg', 'Thumbnail.jpeg'));
> }
> ```

The `oxy_parent-url()` Function

The `oxy_parent-url()` function evaluates the parent URL of an URL received as string.

`oxy_parent-url ( URL )`

URL

The URL as string.

The `oxy_capitalize()` Function

This function capitalizes the first letter of the text received as argument.

`oxy_capitalize ( text )`

text

The text for which the first letter will be capitalized.

> To insert as static text content before each element its capitalized qualified name, use this CSS selector:
>
> ```
> *:before{
>   content: oxy_capitalize(oxy_name()) ": ";
> }
> ```

The `oxy_uppercase()` Function

The `oxy_uppercase()` function transforms to upper case the text received as argument.

`oxy_uppercase ( text )`

text

The text to be capitalized.

> To insert as static text content before each element its upper-cased qualified name, use this CSS selector:
>
> ```
> *:before{
>   content: oxy_uppercase(oxy_name()) ": ";
> }
> ```

The `oxy_lowercase()` Function

The `oxy_lowercase()` function transforms to lower case the text received as argument.

```
oxy_lowercase( text )
```

text

    The text to be lower cased.

> To insert as static text content before each element its lower-cased qualified name, use this CSS selector:
>
> ```
> *:before{
>   content: oxy_lowercase(oxy_name()) ": ";
> }
> ```

The `oxy_concat()` Function

The `oxy_concat()` function concatenates the received string arguments.

```
oxy_concat( str_1 , str_2 )
```

str_1 ... str_n

    The string arguments to be concatenated.

> If an XML element has an attribute called **padding-left**:
>
> ```
>  <p padding-left="20">....
> ```
>
> and you want to add a padding before it with that specific amount specified in the attribute value:
>
> ```
> *[padding-left]{
>  padding-left:oxy_concat(attr(padding-left), "px");
> }
> ```

The `oxy_replace()` Function

The `oxy_replace()` function has two signatures:

- `oxy_replace( text , target , replacement )`

  This function replaces each substring of the text that matches the literal target string with the specified literal replacement string.

  text

  The text in which the replace will occur.

  target

  The target string to be replaced.

  replacement

  The string replacement.

- `oxy_replace( text , target , replacement , isRegExp )`

  This function replaces each substring of the text that matches the target string with the specified replacement string.

text

The text in which the replace will occur.

target

The target string to be replaced.

replacement

The string replacement.

isRegExp

If *true* the target and replacement arguments are considered regular expressions, if *false* they are considered literal strings.

If you have image references but you want to see in the visual Author editing mode thumbnail images which reside in the same folder:

```
image[href]{
 content:oxy_url(oxy_base-uri(), oxy_replace(attr(href), '.jpeg', 'Thumbnail.jpeg'));
}
```

The `oxy_unparsed-entity-uri()` Function

The `oxy_unparsed-entity-uri()` function returns the URI value of an unparsed entity name.

```
oxy_unparsed-entity-uri( unparsedEntityName )
```

unparsedEntityName

The name of an unparsed entity defined in the DTD.

This function can be useful to display images which are referred with unparsed entity names.

CSS for displaying the image in Author for an `imagedata` with `entityref` to an unparsed entity

```
imagedata[entityref]{
content: oxy_url(oxy_unparsed-entity-uri(attr(entityref)));
}
```

The `oxy_attributes()` Function

The `oxy_attributes()` function concatenates the attributes for an element and returns the serialization.

```
oxy_attributes()
```

---

oxy_attributes()

For the following XML fragment:`<element att1="x" xmlns:a="2" x="""/>` the CSS selector

```
element{
  content:oxy_attributes();
}
```

will display`att1="x" xmlns:a="2" x="""`.

The `oxy_substring()` Function

The `oxy_substring()` function has two signatures:

- `oxy_substring( text , startOffset )`

  Returns a new string that is a substring of the original **text** string. It begins with the character at the specified index and extends to the end of **text** string.

text

The original string.

startOffset

The beginning index, inclusive

- `substring ( text , startOffset , endOffset )`

  Returns a new string that is a substring of the original **text** string. The substring begins at the specified **startOffset** and extends to the character at index **endOffset** - 1.

  text

  The original string.

  startOffset

  The beginning index, inclusive

  endOffset

  The ending index, exclusive.

  `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'`.

  If we want to display only part of an attribute's value, the part which comes before an **Appendix** string:

  ```
  image[longdesc]{
   content: oxy_substring(attr(longdesc), 0, oxy_indexof(attr(longdesc), "Appendix"));
  }
  ```

The `oxy_getSomeText(text, length)` Function

The `oxy_getSomeText(text, length)` function allows you to truncate a long string and to set a maximum number of displayed characters.

The following properties are supported:

- **text** - displays the actual text
- **length** - sets the maximum number of characters that are displayed
- **endsWithPoints** - specifies whether the truncated text ends with ellipsis

  ---

  If an attribute value is very large we can trim its content before it is displayed as static content:

  ```
  *[longdesc]:before{
    content: oxy_getSomeText(attr(longdesc), 200);
  }
  ```

The `oxy_indexof()` Function

The `oxy_indexof()` 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.
```

If we want to display only part of an attribute's value, the part which comes before an **Appendix** string:

```
image[longdesc]{
  content: oxy_substring(attr(longdesc), 0, oxy_indexof(attr(longdesc), "Appendix"));
}
```

The `oxy_lastindexof()` Function

The oxy_lastindexof() 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.
```

If we want to display only part of an attribute's value, the part which comes before an **Appendix** string:

```
image[longdesc]{
  content: oxy_substring(attr(longdesc), 0, oxy_lastindexof(attr(longdesc), "Appendix"));
}
```

The `oxy_xpath()` Function

The oxy_xpath() function has the signature:

• oxy_xpath (XPathExpression [, processChangeMarkers , value])

Evaluates the given XPath expression using Saxon 9 and returns the result. The parameters of the function are: the XPath expression, followed by an optional `processChangeMarkers` parameter and then by its value, which can be either `true` or `false` (default value). When you set the parameter to `true`, the function returns the result text with all the change markers accepted (delete changes are removed and insert changes are preserved).

> 📑 **Note:** The entities and `xi:include` sections are ignored when the XPath expressions are evaluated.

expression

An XPath 2.0 expression to be evaluated.

The following example counts the number of words from a paragraph (including tracked changes) and displays the result in front of it:

```
para:before{
   content:
     concat("|Number of words:",
            oxy_xpath(
                "count(tokenize(normalize-space(string-join(text(), ''')), ' '))",
                processChageMarkers,
                true),
            "| ");
}
```

The `oxy_editor()` Function

The `oxy_editor` function allows you to edit attributes or simple element values in the **Author** mode using standard form controls like combo boxes, text fields, pop-ups, URL choosers or to implement your own custom form controls and renderers.

The `oxy_editor()` function can appear in the `content` property of a CSS rule. The function's arguments are *property name - property value* pairs:

```
myElement {
    content: oxy_editor(
        type, combo,
        edit, "@my_attr"
    )
}
```

The form control allows you to edit Processing Instructions (PIs), the value of an attribute, or the text content of an element. This is specified using the `edit` property. This property can have the following values:

- **@attribute_name** - specifies that the presented/edited value is the value of an attribute;
- **#text** - specifies that the presented/edited value is the simple text value of an element. This text can contain built-in character entities.

  > 📑 **Note:** You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the oxy_editor function specifies.

You can use a number of built-in form controls through the `type` property. The following values are recognized:

- *text* - a text field with optional content completion capabilities is used to present and edit a value;
- *combo* - a combo-box is used to present and edit a value;
- *check* - a single check box or multiple check boxes are used to present and edit a value;
- *popupSelection* - a pop-up with single/multiple selection is used as form control;
- *button* - a button that invokes an author action is used as form control;
- *urlChooser* - a text field with a browse button is used as form control.
- *datePicker* - a text field with a calendar browser button is used as form control.

To watch our video demonstration about form controls, go to *http://oxygenxml.com/demo/Form_Controls.html*.

The Text Field Form Control

A text field with optional content completion capabilities is used to present and edit the value of an attribute or an element. It can be added using the generic function `oxy_editor()`. Alternatively, you can use the dedicated function available for each built-in form control `oxy_textfield`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Text form control, its value has to be `text`.
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). This property can have the following values:

    - **@attribute_name** - The name of the attribute who's value is being edited. If the attribute is in a namespace the value of the property must be a QName and the CSS must have a namespace declaration for the prefix:

      ```
      @namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
      myElement {
          content: oxy_editor(
              type, combo,
              edit,  "@fc:my_attr"
          )
      }
      ```

    - **#text** - specifies that the presented/edited value is the simple text value of an element.

      > 📝 **Note:** You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the oxy_editor function specifies

- `columns` - controls the width of the form control. The unit size is the width of the **w** character.
- `width` - specifies the width of the content area using relative (em, ex), absolute (in, cm, mm, pt, pc, px) and percentage length units (followed by the `%` character). The `width` property takes precedence over the `columns` property, if the two are used together.
- `fontInherit` - this value specifies whether the form control inherits its font from its parent element. The values of this property can be `true`, or `false`.To make the `pop-up` form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- `visible` - specifies whether the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible).
- `values` - specifies the values that populate the content completion list of proposals. In case these values are not specified, they are collected from the associated schema.
- `tooltips` - associates tooltips to each value in the `values` property. The value of this property are a list of tooltips separated by commas. In case you want the tooltip to display a comma, use the `${comma}` variable.
- `tooltip` - specifies a tooltip for the form control itself. This tooltip is displayed when you hover the form control using your cursor.
- `color` - specifies the foreground color of the form control. In case the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.

Text Field Form Control

```
element {
    content: "Label: "
        oxy_editor(
            type, text,
            edit, "@my_attr",
            values, "value1, value2"
            columns, 40);
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_textfield` dedicated function.

```
element {
    content: "Label: "
        oxy_textfield(
            edit, "@my_attr",
            values, "value1, value2"
```

```
        columns, 40);
}
```

The Combo Box Form Control

A combo box is used to present and edit the value of an attribute or an element. It can be added using the generic function `oxy_editor()`. Alternatively, you can use the dedicated function available for each built-in form control `oxy_combobox`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Combo box form control, its value has to be `combo`.
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). This property can have the following values:

  - **@attribute_name** - The name of the attribute who's value is being edited. If the attribute is in a namespace the value of the property must be a QName and the CSS must have a namespace declaration for the prefix:

    ```
    @namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
    myElement {
        content: oxy_editor(
            type, combo,
            edit,  "@fc:my_attr"
        )
    }
    ```

  - **#text** - specifies that the presented/edited value is the simple text value of an element.

    > 📄 **Note:** You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the oxy_editor function specifies

- `columns` - controls the width of the form control. The unit size is the width of the **w** character.
- `width` - specifies the width of the content area using relative (em, ex), absolute (in, cm, mm, pt, pc, px) and percentage length units (followed by the `%` character). The `width` property takes precedence over the `columns` property, if the two are used together.
- `visible` - specifies whether the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible).
- `editable` - this property accepts the **true** and **false** values. The **true** value generates an editable combo-box that allows you to insert other values than the proposed ones. The **false** value generates a combo-box that only accepts the proposed values.
- `tooltips` - associates tooltips to each value in the `values` property. The value of this property are a list of tooltips separated by commas. In case you want the tooltip to display a comma, use the `${comma}` variable.
- `values` - specifies the values that populate the content completion list of proposals. In case these values are not specified, they are collected from the associated schema.
- `fontInherit` - this value specifies whether the form control inherits its font from its parent element. The values of this property can be `true`, or `false`. To make the Combo-box form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- `labels` - this property must have the same number of items as the `values` property. Each item provides a literal description of the items listed in the `values` property.

  > 📄 **Note:** This property is available only for read-only Combo boxes (Combo boxes that have the `editable` property set to `false`).

- `color` - specifies the foreground color of the form control. In case the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.

Combo Box Form Control

```
comboBox:before {
    content: "A combo box that edits an attribute value. The possible values are provided from
CSS:"
        oxy_editor(
            type, combo,
```

```
                    edit, "@attribute",
                    editable, true,
                    values, "value1, value2, value3",
                    labels, "Value no1, Value no2, Value no3");
      }
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_combobox` dedicated function.

```
comboBox:before {
    content: "A combo box that edits an attribute value. The possible values are provided from
 CSS:"
        oxy_combobox(
            edit, "@attribute",
            editable, true,
            values, "value1, value2, value3",
            labels, "Value no1, Value no2, Value no3");
}
```

The Check Box Form Control

A single check-box or multiple check-boxes are used to present and edit the value on an attribute or element. It can be added using the generic function `oxy_editor()`. Alternatively, you can use the dedicated function available for each built-in form control `oxy_checkbox`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Check Box form control, its value has to be `check`;
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). This property can have the following values:
    - **@attribute_name** - The name of the attribute who's value is being edited. If the attribute is in a namespace the value of the property must be a QName and the CSS must have a namespace declaration for the prefix:

    ```
    @namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
    myElement {
        content: oxy_editor(
            type, combo,
            edit,  "@fc:my_attr"
        )
    }
    ```

    - **#text** - specifies that the presented/edited value is the simple text value of an element.

        > 📄 **Note:** You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the oxy_editor function specifies

- `resultSeparator` - in case multiple check-boxes are used, the separator is used to compose the final result;
- `tooltips` - associates tooltips to each value in the `values` property. The value of this property are a list of tooltips separated by commas. In case you want the tooltip to display a comma, use the `${comma}` variable;
- `visible` - specifies whether the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible);
- `values` - specifies the values that are committed when the check-boxes are selected. In case these values are not specified in the CSS, they are collected from the associated XML Schema;
- `fontInherit` - this value specifies whether the form control inherits its font from its parent element. The values of this property can be `true`, or `false`. To make the Check box form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- `uncheckedValues` - specifies the values that are committed when the check-boxes are not selected;
- `labels` - this property must have the same number of items as the `values` property. Each item provides a literal description of the items listed in the `values` property. In case this property is not specified, the `values` property is used as label;
- `columns` - controls the width of the form control. The unit size is the width of the **w** character;
- `color` - specifies the foreground color of the form control. In case the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.

Single Check-box Form Control

```
checkBox[attribute]:before {
    content: "A check box editor that edits a two valued attribute (On/Off).
            The values are specified in the CSS:"
oxy_editor(
    type, check,
        edit, "@attribute",
        values, "On",
        uncheckedValues, "Off",
        labels, "On/Off");
}
```

Multiple Check-boxes Form Control

```
multipleCheckBox[attribute]:before {
    content: "Multiple checkboxes editor that edits an attribute value.
            Depending whether the check-box is selected a different value is committed:"
        oxy_editor(
        type, check,
        edit, "@attribute",
        values, "true, yes, on",
        uncheckedValues, "false, no, off",
        resultSeparator, ",",
        labels, "Present, Working, Started");
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_checkbox` dedicated function.

```
multipleCheckBox[attribute]:before {
    content: "Multiple checkboxes editor that edits an attribute value.
            Depending whether the check-box is selected a different value is committed:"
        oxy_checkbox(
        edit, "@attribute",
        values, "true, yes, on",
        uncheckedValues, "false, no, off",
        resultSeparator, ",",
        labels, "Present, Working, Started");
}
```

The Pop-up Form Control

A pop-up with single or multiple selection is used as a form control. It can be added using the generic function `oxy_editor()`. Alternatively, you can use the dedicated function available for each built-in form control `oxy_popup`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Pop-up form control, its value has to be `popupSelection`.
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). This property can have the following values:

    - **@attribute_name** - The name of the attribute who's value is being edited. If the attribute is in a namespace the value of the property must be a QName and the CSS must have a namespace declaration for the prefix:

    ```
    @namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
    myElement {
        content: oxy_editor(
            type, combo,
            edit,  "@fc:my_attr"
        )
    }
    ```

    - **#text** - specifies that the presented/edited value is the simple text value of an element.

        📄 **Note:** You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the oxy_editor function specifies

- `rows` - this property specifies the number of rows that the form control presents.

    📄 **Note:** In case the value of the **rows** property is not specifies, the default value of *12* is used.

- `color` - specifies the foreground color of the form control. In case the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.

  📋 **Note:** This property is used for rendering in the **Author** mode.

- `visible` - specifies whether the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible).
- `tooltips` - associates tooltips to each value in the `values` property. The value of this property are a list of tooltips separated by commas. In case you want the tooltip to display a comma, use the `${comma}` variable.
- `values` - specifies the values that are committed when the check-boxes are selected. In case these values are not specified in the CSS, they are collected from the associated XML Schema.
- `resultSeparator` - in case multiple check-boxes are used, the separator is used to compose the final result.

  📋 **Note:** The value of the `resultSeparator` property cannot exceed one character.

- `selectionMode` - specifies whether the form control allows the selection of a single value or of multiple values. The predefined values of this property are `single` and `multiple`.
- `labels` - specifies the label associated with each entry used for presentation. In case this property is not specified, the **values** property is used as label;
- `columns` - controls the width of the form control. The unit size is the width of the **w** character. This property is used for the visual representation of the form control.
- `width` - specifies the width of the content area using relative (em, ex), absolute (in, cm, mm, pt, pc, px) and percentage length units (followed by the `%` character). The `width` property takes precedence over the `columns` property, if the two are used together.
- `rendererSort` - allows you to sort the values rendered on the pop-up form control label. The possible values of this property are `ascending` and `descending`.
- `editorSort` - allows you to sort the values rendered on the pop-up window. The possible values of this property are `ascending` and `descending`.
- `rendererSeparator` - defines a separator used when multiple values are rendered.
- `fontInherit` - this value specifies whether the form control inherits its font from its parent element. The values of this property can be `true`, or `false`. To make the Pop-up form control inherit its font from its parent element, set the `fontInherit` property to `true`.

  ⓘ **Tip:** In the below example, the value of the **fontInherit** property is **true**, meaning that the pop-up form control inherits the font size of `30px` from the **font-size** property.

Pop-up Form Control

```
popupWithMultipleSelection:before {
    content: " This editor edits an attribute value. The possible values are     specified
    inside the CSS: "
        oxy_editor(
            type, popupSelection,
            edit, "@attribute",
            values, "value1, value2, value3, value4, value5",
            labels, "Value no1, Value no2, Value no3, Value no4, Value no5",
            resultSeparator, "|",
            columns, 10,
            selectionMode, "multiple",
            fontInherit, true);
    font-size:30px;
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_popup` dedicated function.

```
popupWithMultipleSelection:before {
    content: " This editor edits an attribute value. The possible values are     specified
    inside the CSS: "
        oxy_popup(
            edit, "@attribute",
            values, "value1, value2, value3, value4, value5",
```

```
            labels, "Value no1, Value no2, Value no3, Value no4, Value no5",
            resultSeparator, "|",
            columns, 10,
            selectionMode, "multiple",
            fontInherit, true);
        font-size:30px;
}
```

The Button Form Control

This form control contributes a button that invokes a *custom Author action* (defined in the associated Document Type) using its defined ID. It can be added using the generic function oxy_editor(). Alternatively, you can use the dedicated function available for each built-in form control oxy_button. The following properties are supported:

- type - this property specifies the built-in form control you are using. Only needed if you use oxy_editor generic function to add the form control. For the Button form control the value of the type property is button;
- actionContext - specifies the context in which the action associated with the form control is executed. Its possible values are element and caret. If you select the element value, the context is the element that holds the form control. If you select the caret value, the action is invoked at the caret location. In case the caret is not inside the element that holds the form control, the element value is selected automatically;
- fontInherit - this value specifies whether the form control inherits its font from its parent element. The values of this property can be true, or false. To make the button form control inherit its font from its parent element, set the **fontInherit** property to **true**;
- color - specifies the foreground color of the form control. In case the value of the color property is inherit, the form control has the same color as the element in which it is inserted;
- actionID - the ID of the action specified in *Author actions*, that is invoked when you click the button;

> 📄 **Note:** The element that contains the Button form control represents the context where the action is invoked.

- visible - specifies whether the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible);
- transparent - flattens the aspect of the button form control, removing its border and background.
- showText - specifies if the action text should be displayed on the button form control. If this property is missing then the button displays only the icon if it is available, or the text if the icon is not available. The values of this property can be true or false.

```
element {
  content: oxy_button(actionID, 'remove.attribute', showText, true);
}
```

- showIcon - specifies if the action icon should be displayed on the button form control. If this property is missing then the button displays only the icon if it is available, or the text if the icon is not available. The values of this property can be true or false.

```
element {
  content: oxy_button(actionID, 'remove.attribute', showIcon, true);
}
```

- enableInReadOnlyContext - set this property to true on *button form controls* or *groups of buttons form controls* to enable them. This property is useful in *read-only* areas (having the -oxy-editable property set to false) if , for example, you want to execute an action that does not modify the context.

Button Form Control

```
button:before {
    content: "Label:"
        oxy_editor(
            type, button,
            /* This action is declared in the document type associated with the XML document.
 */
            actionID, "insert.popupWithMultipleSelection");
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_button` dedicated function.

```
button:before {
    content: "Label:"
        oxy_button(
            /* This action is declared in the document type associated with the XML document.
 */
            actionID, "insert.popupWithMultipleSelection");
}
```

The Button Group Form Control

A pop-up menu is shown, which can invoke one of the several custom Author actions (defined in the associated Document Type) specified by their ID. It can be added using the generic function `oxy_editor()`. Alternatively, you can use the dedicated function available for each built-in form control `oxy_buttonGroup`. This type of form control supports the following properties:

- `actionIDs` - comma separated IDs of the actions to be displayed in the pop-up menu;
- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control
- `label` - specifies the label to be displayed on the button;
- `icon` - the path to the icon to be displayed on the button;
- `actionContext` - specifies the context in which the action associated with the form control is executed. Its possible values are `element` and `caret`. If you select the `element` value, the context is the element that holds the form control. If you select the `caret` value, the action is invoked at the caret location. In case the caret is not inside the element that holds the form control, the `element` value is selected automatically;
- `visible` - specifies whether the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible);
- `actionStyle` - specifies what to display for an action in the pop-up menu. The values of this property can be text, and icon, or both;
- `tooltips` - associates tooltips to each value in the `values` property. The value of this property are a list of tooltips separated by commas. In case you want the tooltip to display a comma, use the `${comma}` variable;
- `transparent` - makes the button transparent without any borders or background colors. The values of this property can be `true` or `false`;
- `fontInherit` - this value specifies whether the form control inherits its font from its parent element. The values of this property can be `true`, or `false`. To make the form control inherit its font from its parent element, set the **fontInherit** property to `true`.
- `enableInReadOnlyContext` - set this property to `true` on *button form controls* or *groups of buttons form controls* to enable them. This property is useful in *read-only* areas (having the `-oxy-editable` property set to `false`) if , for example, you want to execute an action that does not modify the context.

The Button Group Form Control

```
buttongroup:before {
    content:
        oxy_label(text, "Button Group:", width, 150px, text-align, left)
        oxy_buttonGroup(
            label, 'A group of actions',
            /* The action IDs are declared in the document type  associated with the XML
document. */
            actionIDs, "insert.popupWithMultipleSelection,insert.popupWithSingleSelection",
            actionStyle, "both");
}
```

The Text Area Form Control

A text area with optional syntax highlight capabilities is used to present and edit the value of an attribute or an element. It can be added using the generic function `oxy_editor()`. Alternatively, you can use the dedicated function available for each built-in form control `oxy_textArea`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Text Area form control, its value has to be `textArea`.
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). This property can have the following values:

  - **@attribute_name** - The name of the attribute who's value is being edited. If the attribute is in a namespace the value of the property must be a QName and the CSS must have a namespace declaration for the prefix:

    ```
    @namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
    myElement {
        content: oxy_editor(
            type, combo,
            edit,  "@fc:my_attr"
        )
    }
    ```

  - **#text** - specifies that the presented/edited value is the simple text value of an element.

    📝 **Note:** You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the oxy_editor function specifies

- `columns` - controls the width of the form control. The unit size is the width of the **w** character.
- `width` - specifies the width of the content area using relative (`em`, `ex`), absolute (`in`, `cm`, `mm`, `pt`, `pc`, `px`) and percentage length units (followed by the `%` character). The `width` property takes precedence over the `columns` property, if the two are used together.
- `fontInherit` - this value specifies whether the form control inherits its font from its parent element. The values of this property can be `true`, or `false`.
- `visible` - specifies whether the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible).
- `rows` - this property specifies the number of rows that the form control presents. In case the form control has more lines, you are able to scroll and see them all.
- `contentType` - specifies the type of content for which the form control offers syntax highlight. The following values are supported: `text/css; text/shell; text/cc; text/xquery; text/xml; text/python; text/xsd; text/c; text/xpath; text/javascript; text/xsl; text/wsdl; text/html; text/xproc; text/properties; text/sql; text/rng; text/sch; text/json; text/perl; text/php; text/java; text/batch; text/rnc; text/dtd; text/nvdl; text/plain;`
- `indentOnTab` - specifies the behaviour of the **Tab** key. If the value of this property is set to **true**, the **Tab** key inserts characters. If it is set to **false**, **Tab** is used for navigation, jumping to the next editable position in the document.

The `white-space` CSS property influences the value that you edit, as well as the from control size:

- `pre` - the white spaces and new lines of the value are preserved and edited. If the `rows` and `columns` properties are not specifies, the Text Area form control calculates its size on its own so that all the text is visible;
- `pre-wrap` - the long lines are wrapped to avoid horizontal scrolling;

  📝 **Note:** The `rows` and `columns` properties have to be specified. In case these are not specified, the form control considers the value to be `pre`.

- `normal` - the white spaces and new lines are normalized.

  > The following example presents a text area with CSS syntax highlight which calculates its own dimension, and a second one with XML syntax highlight with defined dimension.
  >
  > ```
  > textArea {
  >     visibility: -oxy-collapse-text;
  >     white-space: pre;
  > }
  >
  > textArea[language="CSS"]:before {
  >     content: oxy_textArea(
  >         edit, '#text',
  > ```

```
        contentType, 'text/css');
}

textArea[language="XML"]:before {
    content: oxy_textArea(
        edit, '#text',
        contentType, 'text/xml',
        rows, 10,
        columns, 30);
}
```

The URL Chooser Form Control

A field that allows you to select local and remote resources is used as a form control. The inserted reference will be made relative to the current opened editor's URL. It can be added using the generic function oxy_editor(). Alternatively, you can use the dedicated function available for each built-in form control oxy_urlChooser. This type of editor supports the following properties:

- type - this property specifies the built-in form control you are using. Only needed if you use oxy_editor generic function to add the form control. For the URL Chooser editor, its value has to be urlChooser.
- edit - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). This property can have the following values:

    - **@attribute_name** - The name of the attribute who's value is being edited. If the attribute is in a namespace the value of the property must be a QName and the CSS must have a namespace declaration for the prefix:

    ```
    @namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
    myElement {
        content: oxy_editor(
            type, combo,
            edit,  "@fc:my_attr"
        )
    }
    ```

    - **#text** - specifies that the presented/edited value is the simple text value of an element.

        📄 **Note:** You can set the value of the visibility property to -oxy-collapse-text to render the text only in the form control that the oxy_editor function specifies

- columns - controls the width of the form control. The unit size is the width of the **w** character.
- width - specifies the width of the content area using relative (em, ex), absolute (in, cm, mm, pt, pc, px) and percentage length units (followed by the % character). The width property takes precedence over the columns property, if the two are used together.
- color - specifies the foreground color of the form control. In case the value of the color property is inherit, the form control has the same color as the element in which it is inserted.
- visible - specifies whether the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible).
- fontInherit - this value specifies whether the form control inherits its font from its parent element. The values of this property can be true, or false.

> **URL Chooser Form Control**
>
> ```
> urlChooser[file]:before {
>     content: "An URL chooser editor that allows browsing for a URL. The selected URL is made
> relative to the currently edited file:"
>         oxy_editor(
>         type, urlChooser,
>         edit, "@file",
>         columns 25);
> }
> ```
>
> The oxy_editor function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the oxy_urlChooser dedicated function.
>
> ```
> urlChooser[file]:before {
>     content: "An URL chooser editor that allows browsing for a URL. The selected URL is made
> relative to the currently edited file:"
> ```

```
        oxy_urlChooser(
        edit, "@file",
        columns 25);
}
```

The Date Picker Form Control

A text field with a calendar browser is used as a form control. The browse button shows a date chooser allowing you to easily choose a certain date. It can be added using the generic function `oxy_editor()`. Alternatively, you can use the dedicated function available for each built-in form control `oxy_datePicker`. This type of form control supports the following properties:

- `type` - this property specifies the built-in form control you are using. Only needed if you use `oxy_editor` generic function to add the form control. For the Date picker form control, its value has to be `datePicker`.
- `edit` - lets you edit the value of an attribute, the text content of an element or Processing Instructions (PIs). This property can have the following values:

  - **@attribute_name** - The name of the attribute who's value is being edited. If the attribute is in a namespace the value of the property must be a QName and the CSS must have a namespace declaration for the prefix:

    ```
    @namespace fc "http://www.oxygenxml.com/ns/samples/form-controls";
    myElement {
        content: oxy_editor(
            type, combo,
            edit,  "@fc:my_attr"
        )
    }
    ```

  - **#text** - specifies that the presented/edited value is the simple text value of an element.

    > 📝 **Note:** You can set the value of the `visibility` property to `-oxy-collapse-text` to render the text only in the form control that the oxy_editor function specifies

- `columns` - controls the width of the form control. The unit size is the width of the **w** character.
- `width` - specifies the width of the content area using relative (em, ex), absolute (in, cm, mm, pt, pc, px) and percentage length units (followed by the `%` character). The `width` property takes precedence over the `columns` property, if the two are used together.
- `color` - specifies the foreground color of the form control. In case the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.
- `format` - this property specifies the format of the inserted date. The pattern value must be a valid Java date (or date-time) format. If missing, the type of the date is determined from the associated schema.
- `visible` - specifies whether the form control is visible. The possible values of this property are **true** (the form control is visible) and **false** (the form control is not visible).
- `validateInput` - specifies if the form control is validated. In case you introduce a date that does not respect the format, the `datePicker` form control is rendered with red foreground. By default, the input is validated. To disable the validation, set this property to `false`.

Date Picker Form Control

```
date {
    content:
        oxy_label(text, "Date time attribute with format defined in CSS: ", width, 300px)
        oxy_editor(
            type, datePicker,
            columns, 16,
            edit, "@attribute",
            format, "yyyy-MM-dd");
}
```

The `oxy_editor` function acts as a proxy that allows you to insert any of the supported form controls. Alternatively, you can use the `oxy_datePicker` dedicated function.

```
date {
    content:
        oxy_label(text, "Date time attribute with format defined in CSS: ", width, 300px)
```

```
            oxy_datePicker(
                columns, 16,
                edit, "@attribute",
                format, "yyyy-MM-dd");
}
```

Editing Processing Instructions Using Form Controls

Oxygen XML Author allows you to edit *processing instructions*, *comments*, and *CDATA* using the built-in editors.

> **Note:** You can edit both the content and the attribute value from a *processing instruction*.

Editing an Attribute from a Processing Instruction

PI content

```
<?pi_target attr="val"?>
```

CSS

```
oxy|processing-instruction:before {
    display:inline;
    content:
        "EDIT attribute: " oxy_textfield(edit, '@attr', columns, 15);
    visibility:visible;
}
oxy|processing-instruction{
    visibility:-oxy-collapse-text;
}
```

Implementing Custom Form Controls

In case the built-in form controls are not enough, you can implement custom form controls in Java and specify them using the following properties:

- **rendererClassName** - the name of the class that draws the edited value. It must be an implementation of `ro.sync.ecss.extensions.api.editor.InplaceRenderer`. The renderer has to be a SWING implementation and can be used both in the standalone and Eclipse distributions;
- **swingEditorClassName** - you can use this property for the standalone (**Swing**-based) distribution to specify the name of the class used for editing. It is a **Swing** implementation of `ro.sync.ecss.extensions.api.editor.InplaceEditor`;
- **swtEditorClassName** - you can use this property for the Eclipse plug-in distribution to specify the name of the class used for editing. It is a **SWT** implementation of the `ro.sync.ecss.extensions.api.editor.InplaceEditor`;
- **classpath** - you can use this property to specify the location of the classes used for a custom form control. The value of the **classpath** property is an enumeration of URLs separated by comma;
- **edit** - in case your form control edits the value of an attribute, or the text value of an element, you can use the **@attribute_name** and **#text** predefined values and oxygen will perform the commit logic by itself. You can use the **custom** value to perform the commit logic yourself.

Custom Form Control Implementation

Sample Java code for a custom combo box form control implementation which inserts an XML element in the content when the editing stops:

```
public class ComboBoxEditor  extends AbstractInplaceEditor {
  /**
   * @see ro.sync.ecss.extensions.api.editor.InplaceEditor#stopEditing()
   */
  @Override
  public void stopEditing() {
    Runnable customCommit =  new Runnable() {
      @Override
```

```
        public void run() {
         AuthorDocumentController documentController =
context.getAuthorAccess().getDocumentController();
            documentController.insertXMLFragment( "<custom/>", offset);
        }
      };
      EditingEvent event =  new EditingEvent(customCommit,  true);
      fireEditingStopped(event);
    }
```

If the custom form control is intended to work in the Oxygen XML Author standalone distribution, the declaration of **swtEditorClassName** is not required. The *renderer* (the class that draws the value) and the *editor* (the class that edits the value) have different properties because you can present a value in one way and edit it in another way.

The custom form controls can use any of the predefined properties of the oxy_editor function, as well as specified custom properties. This is an example of how to specify a custom form control:

```
myElement {
    content: oxy_editor(
        rendererClassName, "com.custom.editors.CustomRenderer",
        swingEditorClassName, "com.custom.editors.SwingCustomEditor",
        swtEditorClassName, "com.custom.editors.SwtCustomEditor",
        edit, "@my_attr"
        customProperty1, "customValue1",
        customProperty2, "customValue2"
    )
}
```

> 📄 **Note:** Add these custom **Java** implementations in the *classpath* of the document type associated with the document you are editing. To get you started the Java sources for the SimpleURLChooserEditor are available in the Oxygen SDK.

The oxy_editor function can receive other functions as parameters for obtaining complex behaviors.

> The following example shows how the combo box editor can obtain its values from the current XML file by calling the oxy_xpath function:
>
> ```
> link:before{
>         content: "Managed by:"
>           oxy_editor(
>               type, combo,
>               edit, "@manager",
>               values, oxy_xpath('string-join(//@id , ",") '));
> ```

The `oxy_label()` Function

The oxy_label() function can be used in conjunction with the CSS content property to change the style of generated text. The arguments of the function are *property name - property value* pairs. The following properties are supported:

- text - this property specifies the built-in form control you are using
- width - specifies the horizontal space reserved for the content. The value of this property has the same format as the value of the CSS width property. In case this value is not specified, the text is wrapped
- color - specifies the foreground color of the form control. In case the value of the color property is inherit, the form control has the same color as the element in which it is inserted
- background-color - specifies the background color of the form control. In case the value of the background-color property is inherit, the form control has the same color as the element in which it is inserted
- styles - specifies styles for the form control. The values of this property are a set of CSS properties:

  - font-weight, font-size, font-style, font
  - text-align, text-decoration
  - width

• `color`, `background-color`

```
element{
    content: oxy_label(text, "Label Text", styles,
        "font-size:2em;color:red;");
}
```

If the text from an `oxy_label()` function contains new lines, for example `oxy_label(text, 'LINE1\A LINE2', width, 100px)`, the text is split in two. Each of the two new lines has the specified width of 100 pixels.

📄 **Note:** The text is split after `\A`, which represents a new line character.

You can use the *`oxy_editor()`* and `oxy_label()` functions together to create a form control based layout.

> Let's say we want to edit two attributes on a single element using form controls on separate lines:
>
> ```
> person:before {
>   content: "Name:*" oxy_textfield(edit, '@name', columns, 20) "\A Address:" oxy_textfield(edit,
>   '@address', columns, 20)
> }
> ```
>
> We can use `oxy_label()` if we want only the **Name** label to be bold and also to properly align the two controls:
>
> ```
> person:before {
>   content: oxy_label(text, "Name:*", styles, "font-weight:bold;width:200px") oxy_textfield(edit,
>   '@name', columns, 20) "\A "
>             oxy_label(text, "Address:", styles, "width:200px") oxy_textfield(edit, '@address',
>   columns, 20)
> }
> ```

The `oxy_link-text()` Function

You can use the `oxy_link-text()` function on the CSS `content` property to obtain a text description from the source of a reference. By default, the `oxy_link-text()` function resolves DITA and DocBook references. For further details about how you can also extend this functionality to other frameworks, go to *Configuring an Extensions Bundle*.

DITA Support

For DITA, the `oxy_link-text()` function resolves the `xref` element and the elements that have a `keyref` attribute. The text description is the same as the one presented in the final output for those elements. If you use this function for a `topicref` element that has the `navtitle` and `locktitle` attributes set, the function returns the value of the `navtitle` attribute.

DocBook Support

For DocBook, the `oxy_link-text()` function resolves the `xref` element that defines a link in the same document. The text description is the same as the one presented in the final output for those elements.

For the following XML and associated CSS fragments the `oxy_link-text()` function is resolved to the value of the `xreflabel` attribute.

```
<para><code id="para.id" xreflabel="The reference label">my code</code></para>
<para><xref linkend="para.id"/></para>
```

```
xref {
    content: oxy_link-text();
}
```

The `oxy_unescapeURLValue(string)` Function

The `oxy_unescapeURLValue()` function returns the unescaped value of an URL-like string given as a parameter. For example if the value contains `%20` it will be converted to a simple space character.

```
oxy_unescapeURLValue("http://www.example.com/a%20simple%20example.html")
returns the http://www.example.com/a simple example.html value.
```

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.

> If we have an image with **width** and **height** specified on it we can compute the number of pixels on it:
>
> ```
> image:before{
>  content: "Number of pixels: " oxy_multiply(attr(width), attr(height), "px");
> }
> ```

Custom CSS Pseudo-classes

You can set your custom CSS pseudo-classes on the nodes from the *AuthorDocument* model. These are similar to the normal XML attributes, with the important difference that they are not serialized, and by changing them the document does not create undo and redo edits - the document is considered unmodified. You can use custom pseudo-classes for changing the style of an element (and its children) without altering the document.

In Oxygen XML Author they are used to hide/show the `colspec` elements from CALS tables. To take a look at the implementation, see:

1. `[OXYGEN_DIR]/frameworks/docbook/css/cals_table.css` (Search for `-oxy-visible-colspecs`)
2. The definition of action `table.toggle.colspec` from the DocBook 4 framework makes use of the pre-defined *TogglePseudoClassOperation* Author operation.

Here are some examples:

> **Controlling the visibility of a section using a pseudo-class**
>
> You can use a non standard (custom) pseudo-class to impose a style change on a specific element. For instance you can have CSS styles matching the custom pseudo-class `access-control-user`, like the one below:
>
> ```
> section {
>   display:none;
> }
>
> section:access-control-user {
> ```

```
    display:block;
}
```

By setting the pseudo-class `access-control-user`, the element section will become visible by matching the second CSS selector.

Coloring the elements over which the caret was placed

```
*:caret-visited {
  color:red;
}
```

You could create an *AuthorCaretListener* that sets the `caret-visited` pseudo-class to the element at the caret location. The effect will be that all the elements traversed by the caret become red.

The API you can use from the caret listener:

```
ro.sync.ecss.extensions.api.AuthorDocumentController#setPseudoClass(java.lang.String,
ro.sync.ecss.extensions.api.node.AuthorElement)
ro.sync.ecss.extensions.api.AuthorDocumentController#removePseudoClass(java.lang.String,
ro.sync.ecss.extensions.api.node.AuthorElement)
```

Pre-defined *AuthorOperation*s can be used directly in your framework ("Author/Actions") to work with custom pseudo classes:

1. *TogglePseudoClassOperation*
2. *SetPseudoClassOperation*
3. *RemovePseudoClassOperation*

Built in CSS Stylesheet

When Oxygen XML Author renders content in the **Author** mode, it adds built-in CSS selectors (in addition to the CSS stylesheets linked in the XML or specified in the document type associated to the XML document). These built-in CSS selectors are processed before all other CSS content, but they can be overwritten in case the CSS developer wants to modify a default behavior.

List of CSS Selector Contributed by Oxygen XML Author

```
@namespace oxy "http://www.oxygenxml.com/extensions/author";
@namespace xi "http://www.w3.org/2001/XInclude";
@namespace xlink "http://www.w3.org/1999/xlink";
@namespace svg "http://www.w3.org/2000/svg";
@namespace mml "http://www.w3.org/1998/Math/MathML";

oxy|document {
    display:block !important;
}

oxy|cdata {
    display:-oxy-morph !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|processing-instruction {
    display:block !important;
    color: rgb(139, 38, 201) !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|comment {
    display:-oxy-morph !important;
    color: rgb(0, 100, 0) !important;
    background-color:rgb(255, 255, 210) !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}
```

```
oxy|reference:before,
oxy|entity[href]:before{
  link: attr(href) !important;
  text-decoration: underline !important;
  color: navy !important;

  margin: 2px !important;
  padding: 0px !important;
}

oxy|reference:before {
  display: -oxy-morph !important;
  content: url(../images/editContent.gif) !important;
}

oxy|entity[href]:before{
  display: -oxy-morph !important;
  content: url(../images/editContent.gif) !important;
}

oxy|reference,
oxy|entity {
    -oxy-editable:false !important;
    background-color: rgb(240, 240, 240) !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|reference {
    display:-oxy-morph !important;
    /*EXM-28674 No need to present tags for these artificial references.*/
    -oxy-display-tags: none;
}

oxy|entity {
    display:-oxy-morph !important;
}

oxy|entity[href] {
  border: 1px solid rgb(175, 175, 175) !important;
  padding: 0.2em !important;
}

xi|include {
    display:-oxy-morph !important;
    margin-bottom: 0.5em !important;
    padding: 2px !important;
}
xi|include:before,
xi|include:after{
    display:inline !important;
    background-color:inherit !important;
    color:#444444 !important;
    font-weight:bold !important;
}

xi|include:before {
    content:url(../images/link.gif) attr(href)  !important;
    link: attr(href) !important;
}
xi|include[xpointer]:before {
    content:url(../images/link.gif) attr(href) " " attr(xpointer) !important;
    link: oxy_concat(attr(href), "#", attr(xpointer)) !important;
}

xi|fallback {
    display:-oxy-morph !important;
    margin: 2px !important;
    border: 1px solid #CB0039 !important;
}

xi|fallback:before {
    display:-oxy-morph !important;
    content:"XInclude fallback: " !important;
    color:#CB0039 !important;
}

oxy|doctype {
    display:block !important;
    background-color: transparent !important;
    color:blue !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 2px !important;
}

oxy|error {
    display:-oxy-morph !important;
    -oxy-editable:false !important;
```

```
    white-space:pre !important;
    color: rgb(178, 0, 0) !important;
    font-weight:bold !important;
}

oxy|error:before {
    content:url(../images/ReferenceError.png) !important;
}

*[xlink|href]:before {
    content:url(../images/link.gif);
    link: attr(xlink|href) !important;
}


/*No direct display of the MathML and SVG images.*/
svg|svg{
 display:inline !important;
    white-space: -oxy-trim-when-ws-only;
}
/*EXM-28827 SVG can contain more than one namespace in it*/
svg|svg * {
    display:none !important;
    white-space:normal;
}

mml|math{
 display:inline !important;
    white-space: -oxy-trim-when-ws-only;
}
mml|math mml|*{
    display:none !important;
    white-space: normal;
}

/*Text direction attributes*/
*[dir='rtl'] { direction:rtl; unicode-bidi:embed; }
*[dir='rlo'] { direction:rtl; unicode-bidi:bidi-override; }

*[dir='ltr'] { direction:ltr; unicode-bidi:embed; }
*[dir='lro'] { direction:ltr; unicode-bidi:bidi-override; }
```

> To show all entities in the **Author** mode as transparent, without that grayed-out background, first
> define in your CSS after all imports the namespace:
>
> ```
> @namespace oxy "http://www.oxygenxml.com/extensions/author";
> ```
>
> and then add the following selector:
>
> ```
> oxy|entity {
>     background-color: inherit !important;
> }
> ```

Example Files Listings - The Simple Documentation Framework Files

This section lists the files used in the customization tutorials: the XML Schema, CSS files, XML files, XSLT stylesheets.

XML Schema files

sdf.xsd

This sample file can also be found in the *Oxygen SDK distribution* in the "oxygensdk\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 *Oxygen SDK distribution* in the "oxygensdk\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 *Oxygen SDK distribution* in the oxygensdk\samples\Simple Documentation Framework - SDF\framework\css directory.

```
/* Element from another namespace */
@namespace abs "http://www.oxygenxml.com/sample/documentation/abstracts";

abs|def{
    font-family:monospace;
    font-size:smaller;
}
abs|def:before{
    content:"Definition:";
    color:gray;
}

/* Vertical flow */
book,
section,
para,
title,
image,
ref {
    display:block;
}

/* Horizontal flow */
b,i {
    display:inline;
}

section{
    margin-left:1em;
    margin-top:1em;
}

section{
    -oxy-foldable:true;
    -oxy-not-foldable-child: title;
}

link[href]:before{
    display:inline;
    link:attr(href);
    content: "Click to open: " attr(href);
}

/* Title rendering*/
title{
    font-size: 2.4em;
    font-weight:bold;
}

* * title{
    font-size: 2.0em;
}
* * * title{
    font-size: 1.6em;
}
* * * * title{
    font-size: 1.2em;
}
```

```
book,
article{
    counter-reset:sect;
}
book > section,
article > section{
    counter-increment:sect;
}
book > section > title:before,
article > section > title:before{
    content: "Section: " counter(sect) " ";
}

/* Inlines rendering*/
b {
    font-weight:bold;
}

i {
    font-style:italic;
}

/*Table rendering */
table{
    display:table;
    border:1px solid navy;
    margin:1em;
    max-width:1000px;
    min-width:150px;
}

table[width]{
  width:attr(width, length);
}

tr, header{
    display:table-row;
}

header{
    background-color: silver;
    color:inherit
}

td{
  display:table-cell;
  border:1px solid navy;
  padding:1em;
}

image{
    display:block;
    content: attr(href, url);
    margin-left:2em;
}
```

XML Files

sdf_sample.xml

This sample file can also be found in the *Oxygen SDK distribution* in the `"oxygensdk\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 *Oxygen SDK distribution* in the `"oxygensdk\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. Recently (in version 14.2), the component API was extended to also allow multiple edit modes like **Text** and **Grid**. The component can be embedded either in a third-party standalone Java application or customized as a Java Web Applet to provide WYSIWYG-like XML editing directly in your web browser of choice.

The Author Component Startup Project for Java/Swing integrations is available online on the Oxygen XML Author website as a Maven archetype. More information about the setup can be found *here*

Licensing

The licensing terms and conditions for the Author Component are defined in the ***<oXygen/> XML Editor SDK License Agreement***. To obtain the licensing terms and conditions and other licensing information as well, you can also contact our support team at **support@oxygenxml.com**. You may also obtain a free of charge evaluation license key for development purposes. Any deployment of an application developed using the Author Component is also subject to the terms of the SDK agreement.

There are two main categories of Author Component integrations:

1. Integration for internal use.

    You develop an application which embeds the Author Component to be used internally (in your company or by you). You can buy and use *oXygen XML Author standard licenses* (either user-based or floating) to enable the Author Component in your application.

2. Integration for external use.

    Using the Author Component, you create an application that you distribute to other users outside your company (with a CMS for example). In this case you need to contact us to apply for a Value Added Reseller (VAR) partnership.

From a technical point of view, the Author Component provides the Java API to:

*   Inject floating license server details in the Java code. The following link provides details about how to configure a floating license servlet or server:*http://www.oxygenxml.com/license_server.html*.

```
AuthorComponentFactory.getInstance().init(frameworkZips, optionsZipURL, codeBase, appletID,
    //The servlet URL
    "http://www.host.com/servlet",
    //The HTTP credentials user name
    "userName",
    //The HTTP credentials password
    "password");
```

*   Inject the licensing information key (for example the evaluation license key) directly in the component's Java code.

```
AuthorComponentFactory.getInstance().init(
    frameworkZips, optionsZipURL, codeBase, appletID,
    //The license key if it is a fixed license.
    licenseKey);
```

*   Display the license registration dialog to the end user. This is the default behavior in case a `null` license key is set using the API, this transfers the licensing responsibility to the end-user. The user can license an Author component using standard Oxygen XML Author Editor/Author license keys. The license key will be saved to the local user's disk and on subsequent runs the user will not be asked anymore.

```
AuthorComponentFactory.getInstance().init(
    frameworkZips, optionsZipURL, codeBase, appletID,
    //Null license key, will ask the user.
    null);
```

Installation Requirements

Running the Author component as a Java applet requires:

* Oracle (Sun) Java JRE version 1.6 update 10 or newer;
* At least 100 MB disk space and 100MB free memory;
* The applet needs to be signed with a valid certificate and will request full access to the user machine, in order to store customization data (like options and framework files);
* A table of supported browsers can be found here:*Supported browsers and operating systems* on page 442.

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 Author already has frameworks for editing DocBook, DITA, TEI, and so on. Their sources are available in *the Oxygen SDK*. This custom framework is then packed in a zip archive and used to deploy the component.

The following diagram shows the components of a custom framework.
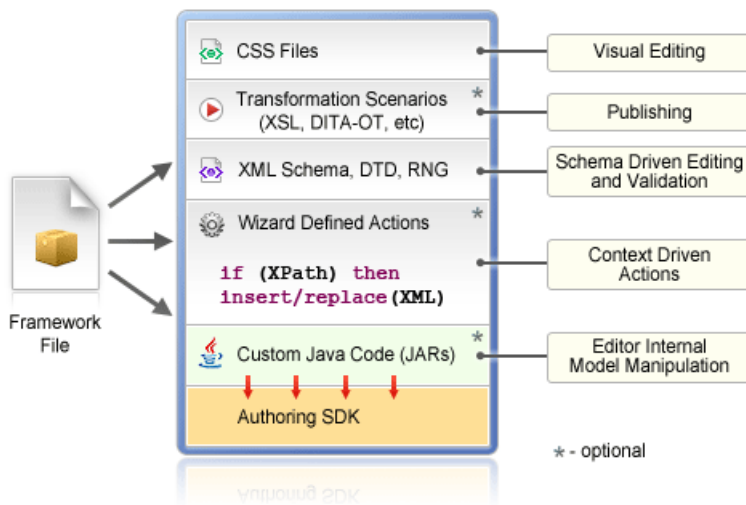
More than one framework can coexist in the same component and can be used at the same time for editing XML documents.
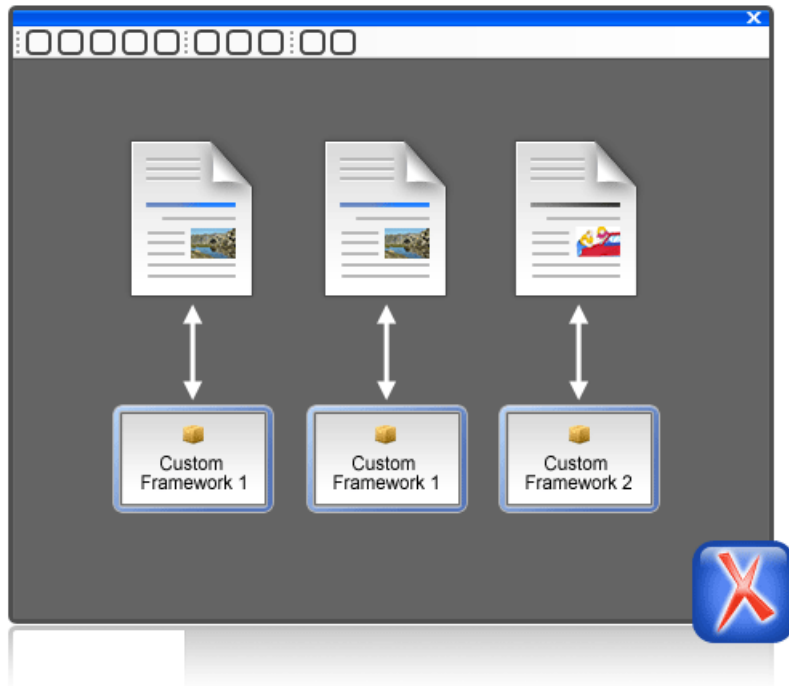
You can add on your custom toolbar all actions available in the standalone Oxygen XML Author application for editing in the **Author** mode. You can also add custom actions defined in the framework customized for each XML type.

The Author component can also provide the *Outline*, *Model*, *Elements* and *Attributes* views which can be added to your own developed containers.

The main entry point for the Author Component Java API is the *AuthorComponentFactory* class.

Example - Customizing the DITA Framework

If you look inside the `bundle-frameworks\oxygen-frameworks` folder distributed with the *Author Component sample project*, it contains a document type framework folder. Customizations which affect the framework/document type configuration for the component should first be done in an Oxygen standalone installation.

An Oxygen standalone installation comes with a `frameworks` folder which contains the `dita` framework located in `[OXYGEN_DIR]\frameworks\dita`. The `dita` framework contains a bundled **DITA-OT** distribution which contains the DTDs used for DITA editing. If your DTD specialization is a DITA OT plugin, it *should be installed* in the `[OXYGEN_DIR]\frameworks\dita\DITA-OT\plugins` folder.

To make changes to the DITA document type configuration, *open the Preferences dialog* **Document Type Association** page. These changes will affect the `[OXYGEN_DIR]\frameworks\dita\dita.framework` configuration file.

After you do this you can re-pack the Author Component following the instructions from the README.html file located in the **oxygen-sample-applet** project. The Author Component Sample Project and the Oxygen standalone installation should be of the same version.

Packing a fixed set of options

The Author Component shares a common internal architecture with the standalone application although it does not have a **Preferences** dialog. But the Author Component Applet can be configured to use a fixed set of user options on startup.

The sample project contains a module called `bundle-options`. The module contains a file called `options.xml` in the `oxygen-options` folder. Such an XML file can be obtained by exporting the options to an XML format from an installation of Oxygen XML Author.

To create an *options file* in the Oxygen XML Author:

- Make sure the options that you want to set are not *stored at project level*.
- Set the values you want to impose as defaults in the *Preferences pages*.
- Select **Options** > **Export Global Options**.

Deployment

The Author Component Java API allows you to use it in your Java application or as a Java applet. The JavaDoc for the API can be found *here*. The sample project found in the `oxygen-sample-applet` module 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:

- Follow the instructions *here* to setup the sample project and look for Java sources of the sample Applet implementation in the sample project **oxygen-sample-applet** module. 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 `web-resources/author-component-dita.html` and `web-resources/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 `target/jnlp/author-component-dita.jnlp` file contains the list of used libraries. This list is automatically generated from the Maven dependencies of the project.
- The sample frameworks and options JAR archives can be found in the `bundle-frameworks` and `bundle-options` modules of the sample project.
- Use the Maven command `mvn package` to pack the component. More information are available *here*. The resulting applet distribution is copied in the `target/jnlp/` directory. From this on, you can copy the applet files on your web server.
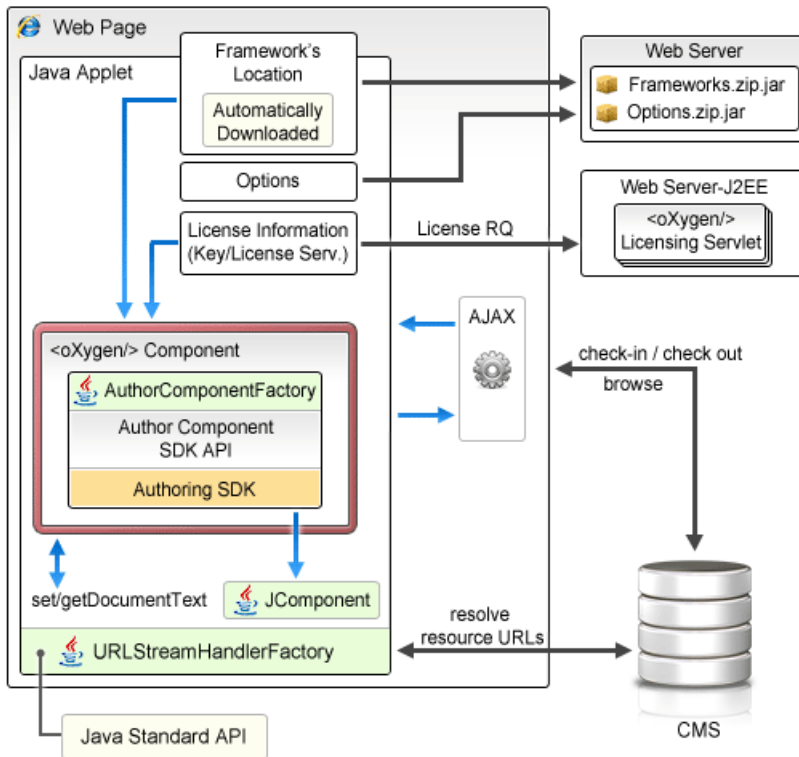
Figure 165: Oxygen XML Author Author Component deployed as a Java applet

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. The jar files are automatically signed during the *package* phase of the Maven build.

Supported browsers and operating systems

The applet was tested for compatibility with the following browsers:

	IE 7	IE 8	IE 9	IE 10	IE 11	Firefox	Safari	Chrome	Opera
Windows XP	Passed	Passed	-	-	-	Passed	-	Passed	Passed
Vista	-	Passed	Passed	Passed	Passed	Passed	-	Passed	Passed
Windows 7	-	-	Passed	Passed	Passed	Passed	-	Passed	Passed
Windows 8	-	-	-	Passed	Passed	Passed	-	Passed	Passed
Mac OS X (10.6 - 10.9)	-	-	-	-	-	Passed	Passed	Failed	Passed
Linux Ubuntu 10	-	-	-	-	-	Passed	-	Failed	Passed

Communication between the Web Page and Java Applet

Using the Java 1.6 *LiveConnect* technology, applets can communicate with JavaScript code which runs in the Web Page. JavaScript code can call an applet's Java methods and from the Java code you can invoke JavaScript code from the web page.

> You are not limited to displaying only Swing dialogs from the applet. From an applet's operations you can invoke JavaScript API which shows a web page and then obtains the data which has been filled by the user.

Troubleshooting

When the applet fails to start:

1. Make sure that your web browser really runs the next generation Java plug-in and not the legacy Java plug-in.

   For Windows and Mac OSX the procedure is straight forward. Some steps are given below for installing the Java plug-in on Linux.

   Manual Installation and Registration of Java Plugin for Linux:
   http://www.oracle.com/technetwork/java/javase/manual-plugin-install-linux-136395.html

2. Refresh the web page.

3. Remove the Java Webstart cache from the local drive and try again.

   - On Windows this folder is located in: `%APPDATA%\LocalLow\Sun\Java\Deployment\cache`;
   - On Mac OSX this folder is located in: `/Users/user_name/Library/Caches/Java/cache`;
   - On Linux this folder is located in: `/home/user/.java/deployment/cache`.

4. Remove the Author Applet Frameworks cache from the local drive and try again:

   - On Windows Vista or 7 this folder is located in:
     `%APPDATA%\Roaming\com.oxygenxml.author.component`;
   - On Windows XP this folder is located in: `%APPDATA%\com.oxygenxml.author.component`;

- On Mac OSX this folder is located in:
  `/Users/user_name/Library/Preferences/com.oxygenxml.author.component`;
- On Linux this folder is located in: `/home/user/.com.oxygenxml.author.component`.

5. Problems sometimes occur after upgrading the web browser and/or the JavaTM runtime. Redeploy the applet on the server by running ANT in your Author Component project. However, doing this does not always fix the problem, which often lies in the web browser and/or in the Java plug-in itself.

6. Sometimes when the HTTP connection is slow on first time uses the JVM would simply shut down while the jars were being pushed to the local cache (i.e., first time uses). This shut down typically occurs while handling `oxygen.jar`. One of the reasons could be that some browsers (Firefox for example) implement some form of "Plugin hang detector" See
https://developer.mozilla.org/en/Plugins/Out_of_process_plugins/The_plugin_hang_detector.

7. If you are running the Applet using Safari on OS X and it has problems writing to disk or fails to start, do the following:

- in Safari, go to **Safari**->**Preferences**->**Security**;
- select **Manage Website Settings**;
- then select Java and for the **oxygenxml.com** entry choose the **Run in Unsafe mode** option.

Enable JavaWebstart logging on your computer to get additional debug information:

1. Open a console and run `javaws -viewer`;
2. In the **Advanced** tab, expand the **Debugging** category and select all boxes.
3. Expand the **Java console** category and choose **Show console**.
4. Save settings.
5. After running the applet, you will find the log files in:

- On Windows this folder is located in: `%APPDATA%\LocalLow\Sun\Java\Deployment\log`;
- On Mac OSX this folder is located in: `/Users/user_name/Library/Caches/Java/log`;
- On Linux this folder is located in: `/home/user/.java/deployment/log`.

Avoiding Resource Caching

A Java plugin installed in a web browser caches access to all HTTP resources that the applet uses. This is useful in order to avoid downloading all the libraries each time the applet is run. However, this may have undesired side-effects when the applet presents resources loaded via HTTP. If such a resource is modified on the server and the browser window is refreshed, you might end-up with the old content of the resource presented in the applet.

To avoid such a behaviour, you need to edit the `ro.sync.ecss.samples.AuthorComponentSampleApplet` class and set a custom `URLStreamHandlerFactory` implementation. A sample usage is already available in the class, but it is commented-out for increased flexibility:

```
//THIS IS THE WAY IN WHICH YOU CAN REGISTER YOUR OWN PROTOCOL HANDLER TO THE JVM.
//THEN YOU CAN OPEN YOUR CUSTOM URLs IN THE APPLET AND THE APPLET WILL USE YOUR HANDLER
URL.setURLStreamHandlerFactory(new URLStreamHandlerFactory() {
 public URLStreamHandler createURLStreamHandler(String protocol) {
  if("http".equals(protocol) || "https".equals(protocol)) {
   return new URLStreamHandler() {
    @Override
    protected URLConnection openConnection(URL u) throws IOException {
     URLConnection connection = new HttpURLConnection(u, null);
     if(!u.toString().endsWith(".jar")) {
      //Do not cache HTTP resources other than JARS
      //By default the Java HTTP connection caches content for
      //all URLs so if one URL is modified and then re-loaded in the
      //applet the applet will show the old content.
      connection.setDefaultUseCaches(false);
     }
     return connection;
    }
   };
  }
  return null;
 }
});
```

Adding MathML support in the Author Component Web Applet

By default the Author Component Web Applet project does not come with the libraries necessary for viewing and editing MathML equations in the Author page. You can view and edit MathML equations either by adding support for *JEuclid* or by adding support for *MathFlow*.

Adding MathML support using JEuclid

By default, the `JEuclid` library is excluded from the *oxygen-sdk* artifact dependencies. To enable it, comment the following lines in the `pom.xml` file:

```
<exclusion>
    <artifactId>jeuclid-core</artifactId>
    <groupId>net.sourceforge.jeuclid</groupId>
</exclusion>
```

To edit specialized DITA Composite with MathML content, include the entire `[OXYGEN_DIR]/frameworks/mathml2` Mathml2 framework directory in the frameworks bundled with the component in the `bundle-frameworks` module. This directory is used to solve references to MathML DTDs.

Adding MathML support using MathFlow

In the `pom.xml` file add dependencies to the additional libraries used by the `MathFlow` library to parse MathML equations:

1. MFComposer.jar
2. MFExtraSymFonts.jar
3. MFSimpleEditor.jar
4. MFStructureEditor.jar
5. MFStyleEditor.jar

You can refer these additional libraries from the `MathFlow` SDK as in the example below:

```
<dependency>
    <groupId>com.dessci</groupId>
    <artifactId>MFComposer</artifactId>
    <version>1.0.0</version>
    <scope>system</scope>
    <systemPath>${MathFlowSDKDir}/lib/MFComposer.jar</systemPath>
</dependency>
```

In addition, you must obtain fixed `MathFlow` license keys for editing and composing MathML equations and register them using these API methods: `AuthorComponentFactory.setMathFlowFixedLicenseKeyForEditor` and `AuthorComponentFactory.setMathFlowFixedLicenseKeyForComposer`.

To edit specialized DITA Composite with MathML content, include the entire `[OXYGEN_DIR]/frameworks/mathml2` Mathml2 framework directory in the frameworks bundled with the component in the `bundle-frameworks` module. This directory is used to solve references to MathML DTDs.

Adding Support to Insert References from a WebDAV Repository

Already defined actions which insert references, like the **Insert Image Reference** action, display an URL chooser which allows you to select the **Browse Data Source Explorer** action. To use an already configured WebDAV connection in the Author Component, follow these steps:

1. Open a standalone Oxygen XML Author 16.1 and configure a WebDAV connection;
2. Pack the *fixed set of options* from the standalone to use them with the Author Component Project;
3. In the Author Component, the defined connection still does not work when expanded because the additional JAR libraries used to browse the WebDAV repository are missing. By default, the *httpclient* dependency of the *oxygen-sdk* artifact is excluded. You can enable it by commenting the following lines:

```
<exclusion>
    <artifactId>httpclient</artifactId>
    <groupId>org.apache.httpcomponents</groupId>
</exclusion>
```

If you want to have a different WebDAV connection URL, user name and password depending on the user who has started the component, you have a more flexible approach using the API:

```
//DBConnectionInfo(String id, String driverName, String url, String user, String passwd, String host, String
 port)
DBConnectionInfo info = new DBConnectionInfo("WEBDAV", "WebDAV FTP", "http://host/webdav-user-root", "userName",
 "password", null, null);
AuthorComponentFactory.getInstance().setObjectProperty("database.stored.sessions1", new DBConnectionInfo[]
{info});
```

Using Plugins with the Author Component

To bundle Workspace Access plugins, that are developed for standalone application with the Author Component, follow these steps:

- The `bundle-plugins` module must contain the additional plugin directories in the `dropins` subdirectory. The content must also contain a `plugin.dtd` file.

    Note:

    Copy the `plugin.dtd` file from an `[OXYGEN_DIR]\plugins` folder.

- In the class which instantiates the `AuthorComponentFactory`, for example the `ro.sync.ecss.samples.AuthorComponentSample` class, call the methods `AuthorComponentFactory.getPluginToolbarCustomizers()`, `AuthorComponentFactory.getPluginViewCustomizers()` and `AuthorComponentFactory.getMenubarCustomizers()`, obtain the customizers which have been added by the plugins and call them to obtain the custom swing components that they contribute. There is a commented-out example for this in the `AuthorComponentSample.reconfigureActionsToolbar()` method for adding the toolbar from the **Acrolinx** plugin.

    Important: As the Author Component is just a subset of the entire application, there is no guarantee that all the functionality of the plugin works.

Sample SharePoint Integration of the Author Component

This section presents the procedure to integrate the Author Component as a Java applet on a SharePoint site.

Author Component

The Author Component was designed as a separate product to provide the functionality of the standard **Author** mode. Recently (in version 14.2), the component API was extended to also allow multiple edit modes like **Text** and **Grid**. The component can be embedded either in a third-party standalone Java application or customized as a Java Web Applet to provide WYSIWYG-like XML editing directly in your web browser of choice.

The Author Component startup project for Java/Swing integrations is available online as a Maven archetype on the **<oXygen/> XML Editor** website. More information about the setup can be found *here*

Microsoft SharePoint®

Microsoft SharePoint® is a Web application platform developed by Microsoft®.

SharePoint comprises a multipurpose set of Web technologies backed by a common technical infrastructure. It provides the benefit of a central location for storing and collaborating on documents, which can significantly reduce emails and duplicated work in an organization. It is also capable of keeping track of the different versions created by different users.

Why Integrate the Author Component with SharePoint

The Author Component can be embedded in a SharePoint site as a Java applet. This is a simple and convenient way for you to retrieve, open, and save XML and XML related documents stored on your company's SharePoint server, directly from your web browser.

For example, let's say that you are working on a team project that uses the DITA framework for writing product documentation. You have the DITA Maps and topics stored on a SharePoint repository. By using a custom defined

action from the contextual menu of a document, you can easily open it in the Author Component applet that is embedded in your SharePoint Documents page.

You can embed the applet either on a site that is located on a standalone SharePoint server, or on your company's Microsoft Office 365 account.

This example can be used as a starting point for other CMS integrations.

Integration Adjustments
Deploying Resources

You are able to embed the Author component in a SharePoint site as a Java Applet, using the new Applet with JNLP Java technology. Sign with a valid certificate the JNLP file and the associated JAR files that the applet needs.

Deploy these resources on a third party server (other than the SharePoint server). The Java applet downloads the resources as needed. If you deploy the JNLP and JAR files on the SharePoint server, the Java Runtime Environment will not be able to access the applet resources because it is not aware of the current authentication tokens from your browser. This causes the Java Class Loader to fail loading classes, making the applet unable to start.

Accessing Documents

One of the main challenges when integrating the Author Component applet in your SharePoint site is to avoid authenticating twice when opening a document resource stored in your SharePoint repository.

You have already signed in when you started the SharePoint session, but the applet is not aware of your current session. In this case every time the applet is accessing a document it will ask you to input your credentials again.

As a possible solution, do not execute HTTP requests directly from the Java code, but forward them to the web browser that hosts the applet, because it is aware of the current user session (authentication cookies).

To open documents stored on your SharePoint repository, register your own protocol handler to the JVM. We implemented a handler for both *http* and *https* protocols that forwards the HTTP requests to a JavaScript `XMLHttpRequest` object. This way, the browser that executes the JavaScript code is responsible for handling the authentication to the SharePoint site.

To install this handler, add the following line to your Java Applet code (in our case, in the `ro.sync.ecss.samples.AuthorComponentSampleApplet` class):

```
URL.setURLStreamHandlerFactory(new ro.sync.net.protocol.http.handlers.CustomURLStreamHandlerFactory(this));
```

To enable JavaScript calls from your Java applet code, set the MAYSCRIPT attribute to `true` in the `<applet>` element embedded in you HTML page:

```
<applet width="100%" height="600"
  code="ro.sync.ecss.samples.AuthorComponentSampleApplet"
  name="authorComponentAppletName" id="authorComponentApplet"
  MAYSCRIPT="true">
    .....
</applet>
```

> **Tip:** In case the applet is not working, or you cannot open documents from your SharePoint repository, enable the debugging tools that come bundled with your Web Browser or the Java Console from your operating system to try to identify the cause of the problem.

Getting Started

To integrate the Author Component as a Java applet with your SharePoint site, you need the author component start-up project.

The project is available as a Maven archetype online. More information about the setup can be found *here*.

An online demo applet is deployed at
http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-dita-requirements.html.

Customize Your Applet

Follow these steps to customize the Author Component Java applet:

1. Follow *this set of instructions* to setup the sample project and look for the Java sources (these can be customized to fit your requirements) of the sample applet implementation;

    📄 **Note:** The Java source files are located in the `src` folder of the `oxygen-sample-applet` module.

2. Look inside `web-resources/sharepoint/author-component-dita.aspx` and the associated `*.js` resources, to see how the applet is embedded in the page and how it can be controlled using JavaScript (to set and get XML content from it).

3. Edit the `default.properties` configuration to specify your custom certificate information, used to sign the applet libraries. Also, specify the code base from where the applet resources will be downloaded;

4. The sample Applet `target/jnlp/author-component-dita.jnlp` file contains the list of used libraries. This list is automatically generated from the Maven dependencies of the project. The sample frameworks and options JAR archives are located in the `bundle-frameworks` and `bundle-options` modules of the sample project..

    📄 **Note:** The JNLP file and the associated resources and libraries must be deployed on a non-SharePoint web server, otherwise the applet will not be loaded.

5. Use the Maven command `mvn package` to pack the component. More information are available *here*. The resulting applet distribution is copied in the `target/jnlp/` directory. From now on, you can copy the applet files on your web server.

Add Resources to Your SharePoint Site

Copy the following resources to a sub-folder (in our example named `author-component`) of the `SitePages` folder from your SharePoint site, where you want to embed the applet:

1. **`author-component-dita.aspx`** - an HTML document containing the Java applet;

    📄 **Note:** It has an `.aspx` extension instead of `.html`. If you use the latter extension, the browser will download the HTML document instead of displaying it.

    📄 **Note:** Edit the **`.aspx`** file and change the value of the applet parameter jnlp_href to the URL of the deployed `author-component-dita.jnlp`. Keep in mind that the JNLP file should be deployed on a third party server. For example:

    ```
    <applet>
        <param name="jnlp_href"
               value="http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-dita.jnlp"/>
        ..........
    </applet>
    ```

2. **`author-component-dita.css`** - contains custom styling rules for the HTML document;

3. **`author-component-dita.js`** - contains JavaScript code, giving access to the Author Component contained by the Java applet;

4. **`connectionUtil.js`** - contains JavaScript utility methods.

    📄 **Note:** Replace the value of the SPRootSiteURL property with the URL of your SharePoint root site, without trailing `'/'`. This is used by the `openListItemInAuthor(itemUrl)` method, to compute the absolute URL of the list item that is to be opened in the Author applet.

Copy Resources Using <oXygen/> XML Editor

You can use **<oXygen/> XML Editor** to copy your resources to the SharePoint server:

1. Configure a new connection to your SharePoint site in the **Data Source Explorer** View.

    📄 **Note:** To watch our video demonstration about connecting to repository located on a SharePoint server, go to *http://www.oxygenxml.com/demo/SharePoint_Support.html*.

2. Browse your new SharePoint connection site and select the `SitePages` folder;
3. Create a folder named `author-component` using the **New Folder** contextual menu action;
4. Upload your resources to this folder using the **Import Files** contextual menu action.
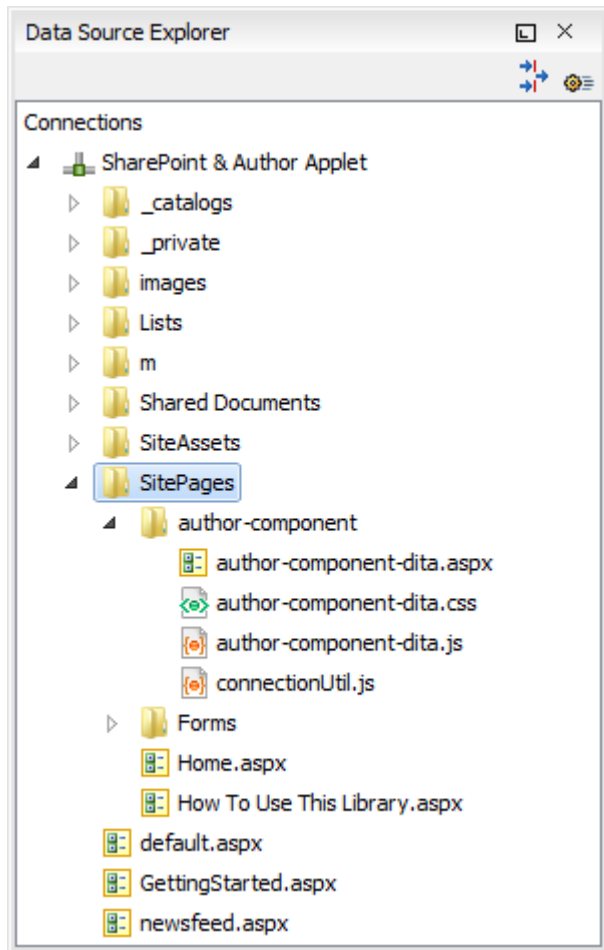
Embed the Java Applet in Your SharePoint Site

To embed the Java Applet in your SharePoint site, edit the page that contains the applet and add a new Script Editor Web Part next to an existing Documents web part.

> **Note:** It is recommended that you deselect the **Enable Java content in the browser** option from the **Java Control Panel** until you finish editing the page. Otherwise, the browser will load the applet for every change that you will make.

Edit the page directly in your browser, following these steps:

1. Navigate to the home page of your SharePoint site where you want to add the Author Component Java applet;
2. Select the **Page** tab from the ribbon located at top of the page and click the **Edit** button;
3. Select the **Insert** tab and click **Web Part**;
4. In the **Categories** panel, select **Media and Content**;
5. In the **Parts** panel, select the **Script Editor** Web Part;
6. Click the **Add** button to insert the selected Web Part to your page content;
7. Select the newly added Web Part;
8. Select the **Web Part** tab and click the **Web Part Properties** button.
9. Click the **Edit Snippet** link under your Web Part;

10. Insert the following HTML snippet to your newly created Web Part:

```
<div>
    <iframe
        id="appletIFrame"
        src="/applet/SitePages/author-component/author-component-dita.aspx"
        width="800px" height="850px">
    </iframe>
    <script type="text/JavaScript">
        function openInAuthor(itemUrl) {
            var appletFrame = document.getElementById("appletIFrame");
            var appletWin = appletFrame.contentWindow;
            appletWin.openListItemInAuthor(itemUrl);
        }
    </script>
</div>
```

The above HTML fragment contains an `IFrame` that points to the page where the Java applet resides. Replace the value of the `src` attribute with the path of the `author-component-dita.aspx` HTML page that you added earlier to the `SitePages` folder;

> **Note:** Use the `iframe` element from the HTML fragment with the expanded form (`<iframe></iframe>`). Otherwise, the Web Part will not display the target page of the frame.

11. Save the changes you made to the page.

> **Note:** Do not forget to select the **Enable Java content in the browser**, to allow the browser to load the Java applet.

Create a SharePoint Custom Action

To open a document from your SharePoint repository in the Author Component applet, add a new custom action to the contextual menu of your Documents Library:

1. Open your SharePoint site in **Microsoft SharePoint Designer®**;

2. Click **Lists and Libraries** in the **Navigation** pane;

3. Open the **Documents** library;

4. Go to the **Custom Actions** panel;

5. Click the **New** button to add a new custom action;

6. Give a name to the action, for example **Open In Oxygen XML Author**;

7. In the **Select the type of action** section, select the **Navigate to URL** option an enter the following text:

```
javascript:openInAuthor("{ItemUrl}")
```

> **Note:** This translates to a call to the `openInAuthor(itemUrl)` JavaScript function defined in the HTML fragment that was embedded in the Script Editor Web Part. The `{ItemUrl}` parameter will be expanded to the URL of the list item that the action is invoked on.

8. Click the **OK** button to save the action.

The Result

The Author Component applet embedded in a SharePoint site:
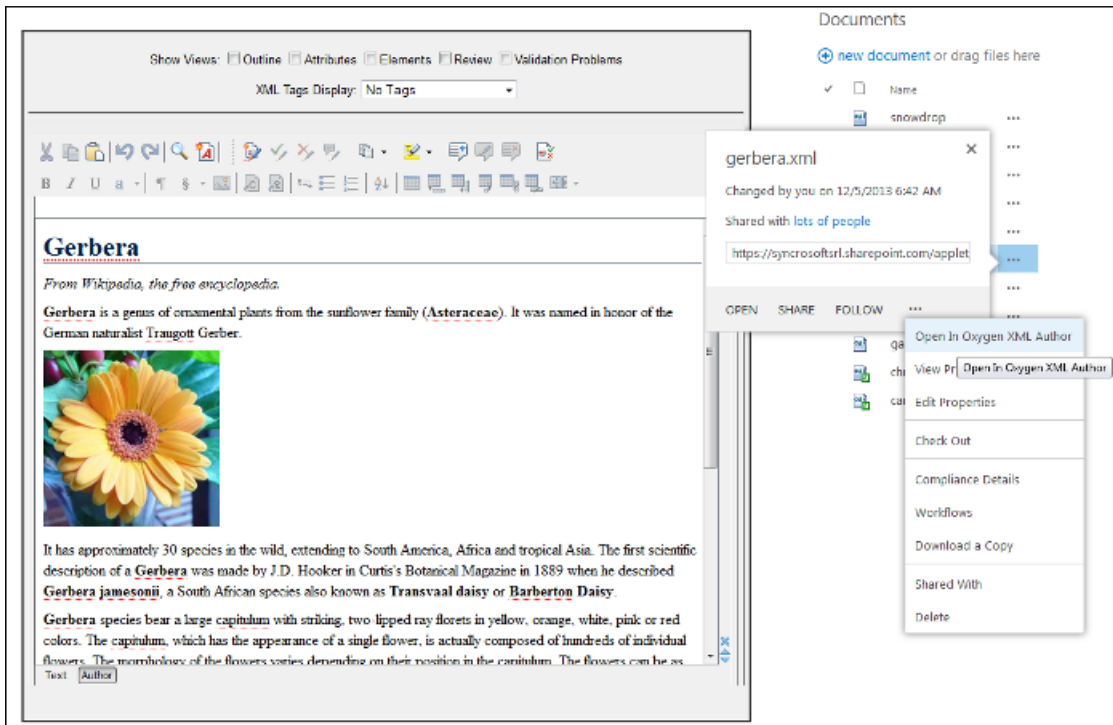
Frequently Asked Questions

Installation and Licensing

1. What hosting options are available for applet delivery and licensing services (i.e., Apache, IIS, etc.)?

   For applet delivery any web server. We currently use Apache to deploy the sample on our site. For the floating license server you would need a J2EE server, like Tomcat if you want to restrict the access to the licenses.

   If you do not need the access restrictions that are possible with a J2EE server you can simplify the deployment of the floating license server by using the standalone version of this server. The standalone license server is a simple Java application that communicates with Author Component by TCP/IP connections.

2. Are there any client requirements beyond the Java VM and (browser) Java Plug-In Technology?

   Oracle (formerly Sun) Java JRE version 1.6 update 10 or newer. At least 200 MB disk space and 200MB free memory would be necessary for the Author Applet component.

3. Are there any other client requirements or concerns that could make deployment troublesome (i.e., browser security settings, client-side firewalls and AV engines, etc.)?

   The applet is signed and will request access to the user machine, in order to store customization data (frameworks). The applet needs to be signed by you with a valid certificate.

4. How sensitive is the applet to the automatic Java VM updates, which are typically on by default (i.e., could automatic updates potentially "break" the run-time)?

   The component should work well with newer Java versions but we cannot guarantee this.

5. How and when are "project" related files deployed to the client (i.e., applet code, DTD, styling files, customizations, etc.)?

   Framework files are downloaded on the first load of the applet. Subsequent loads will re-use the cached customization files and will be much faster.

6. For on-line demo (http://www.oxygenxml.com/demo/AuthorDemoApplet/author-component-dita.html), noted a significant wait during initial startup. Any other mechanisms to enhance startup time?

   See explanation above.

7. Does the XML Author component support multiple documents being open simultaneously? What are the licensing ramifications?

   A single `AuthorComponentFactory` instance can create multiple `EditorComponentProvider` editors which can then be added and managed by the developer who is customizing the component in a Swing `JTabbedPane`. A single license (floating or user-based) is enough for this.

   If you need to run multiple Java Applets or distinct Java processes using the Author component, the current floating license model allows for now only two concurrent components from the same computer when using the license servlet. An additional started component will take an extra license seat.

   Another licensing technique would be to embed the license key in one of the jar libraries used by the applet. But you would need to implement your own way of determining how many users are editing using the Author applet.

8. Is there any internet traffic during an editing session (user actively working on the content, on the client side, in the XML Author component))?

   No.

9. Does Oxygen XML Author work in virtualized environments with terminal services, such as Citrix.

   Oxygen XML Author has been tested in virtualized environments with terminal services, including Citrix, and there have been no problems. We also have several customers that use Oxygen XML Author in Citrix environments and we have not received any problem reports from them either.

   For named licenses, you would normally have to deploy the license for each user or let each user be asked by Oxygen XML Author for the license key and give it himself. Perhaps a better approach would be to create a file named "licensekey.txt"(without the quotes) and in this file paste the nine lines of text of the license key. Copy this file to the Oxygen XML Author installation folder. Any user who runs Oxygen XML Author from this installation will use that license. Please note that this way of deploying the license key does not automatically limit the number of licenses to the specified number, so you should attempt to manually limit(keep evidence of) the number of users that are running Oxygen XML Author to the number of licenses specified in the license key.

Functionality

1. How and when are saves performed back to the hosting server?

   What you can see on our web site is just an example of the Author component (which is a Java Swing component) used in an Applet.

   This applet is just for demonstration purposes. It's source can be at most a starting point for a customization. You should implement, sign and deploy your custom applet implementation.

   The save operation could be implemented either in JavaScript by requesting the XML content from the Applet or in Java directly working with the Author component. You would be responsible to send the content back to the CMS.

2. Is there a particular XML document size (or range) when the Author applet would start to exhibit performance problems?

   The applet has a total amount of used memory specified in the JNLP JavaWebstart configuration file which can be increased if necessary. By default it is 156 Mb. It should work comfortably with documents of 1-3 megabytes.

3. What graphic formats can be directly rendered in the XML Author component?

   GIF, JPEG, PNG, BMP and SVG.

4. Can links be embedded to retrieve (from the server) and "play" other types of digital assets, such as audio or video files?

You could add listeners to intercept clicks and open the clicked links. This would require a good knowledge of the Oxygen 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 XML Author 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 Oxygen SDK*.

More than one framework can coexist in the same Oxygen XML Author 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 expertise). 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 authors.

- *Forcing behaviors (i.e., ensuring change tracking is on and preventing it from being shut down)*

  You could avoid placing the change tracking toolbar actions in the custom applet. You could also use API to turn change tracking ON when the content has been loaded.

- *Preventing access to "privileged" editor processes (i.e., accept/reject changes)*

  You can remove the change tracking actions completely in a custom applet implementation. Including the ones from the contextual menu.

- *Presenting and/or describing XML constructs (i.e., tags) in "plain-English"*

  Using our API you can customize what the Outline or Breadcrumb presents for each XML tag. You can also customize the in-place content completion list.

- *Presenting a small subset of the overall XML tag set (rather than the full tag set) for use by contributors (i.e., allowing an author to only insert Heading, Para and inline emphasis) Could varying "interfaces", with different mixes these capabilities and customizations, be developed and pushed to the user based on a "role" or a similar construct?*

  The API allows for a content completion filter which also affects the *Elements* view.

4. Does the XML Author component's API provide access to the XML document, for manipulation purposes, using common XML syntax such as DOM, XPath, etc.?

   Yes, using the Author API.

5. Can custom dialogs be developed and launched to collect information in a "form" (with scripting behind to push tag the collection information and embed it in the XML document?

   Yes.

6. Can project resources, customizations, etc. be readily shared between the desktop and component versions of your XML Author product line?

   A framework developed for the Desktop Oxygen application can then be bundled with an Author component in a custom applet. For example the Author demo applet from our web site is DITA-aware using the same framework as the Oxygen standalone distribution.

   A custom version of the applet that includes one or more customized frameworks and user options can be built and deployed for non-technical authors by a technical savvy user using a built-in tool of Oxygen. All the authors that load the deployed applet from the same server location will share the same frameworks and options.

   A custom editing solution can deploy one or more frameworks that can be used at the same time.

Creating and Running Automated Tests

If you have developed complex custom plugins and/or document types the best way to test your implementation and insure that further changes will not interfere with the current behavior is to make automated tests for your customization.

An Oxygen XML Author installation standalone (Author or Editor) comes with a main `oxygen.jar` library located in the `[OXYGEN_DIR]`. That JAR library contains a base class for testing developer customizations named `ro.sync.exml.workspace.api.PluginWorkspaceTCBase`.

Please see below some steps in order to develop `JUnit` tests for your customizations using the **Eclipse** workbench:

1. Create a new Eclipse Java project and copy to it the entire contents of the `[OXYGEN_DIR]`.
2. Add to the **Java Build Path**->**Libraries** tab all JAR libraries present in the `[OXYGEN_DIR]/lib` directory. Make sure that the main JAR library `oxygen.jar` or `oxygenAuthor.jar` is the first one in the Java classpath by moving it up in the **Order and Export** tab.

3. Click **Add Library** and add the `JUnit` libraries.

4. Create a new Java class which extends `ro.sync.exml.workspace.api.PluginWorkspaceTCBase`.

5. Pass on to the constructor of the super class the following parameters:

- `File frameworksFolder` The file path to the frameworks directory. It can point to a custom frameworks directory where the custom framework resides.
- `File pluginsFolder` The file path to the plugins directory. It can point to a custom plugins directory where the custom plugins resides.
- `String licenseKey` The license key used to license the test class.

6. Create test methods which use the API in the base class to open XML files and perform different actions on them. Your test class could look something like:

```
public class MyTestClass extends PluginWorkspaceTCBase {

 /**
  * Constructor.
  */
 public MyTestClass() throws Exception {
  super(new File("frameworks"), new File("plugins"),
        "------START-LICENSE-KEY------\n" +
    "\n" +
    "Registration_Name=Developer\n" +
    "\n" +
    "Company=\n" +
    "\n" +
    "Category=Enterprise\n" +
    "\n" +
    "Component=XML-Editor, XSLT-Debugger, Saxon-SA\n" +
    "\n" +
    "Version=14\n" +
    "\n" +
    "Number_of_Licenses=1\n" +
    "\n" +
    "Date=09-04-2012\n" +
    "\n" +
    "Trial=31\n" +
    "\n" +
    "SGN=MCwCFGNoEGJSeiC3XCYIyalvjzHhGhhqAhRNRDpEu8RIWb8icCJO7HqfVP4++A\\=\\\=\n" +
    "\n" +
  "-------END-LICENSE-KEY-------");
 }

  /**
    * <p><b>Description:</b> TC for opening a file and using the bold operation</p>
    * <p><b>Bug ID:</b> EXM-20417</p>
    *
    * @author radu_coravu
    *
    * @throws Exception
    */
   public void testOpenFileAndBoldEXM_20417() throws Exception {
     WSEditor ed = open(new File("D:/projects/eXml/test/authorExtensions/dita/sampleSmall.xml").toURL());
     //Move caret
     moveCaretRelativeTo("Context", 1, false);

     //Insert <b>
     invokeAuthorExtensionActionForID("bold");
     assertEquals("<?xml version=\"1.0\" encoding=\"utf-8\"?>\n" +
       "<!DOCTYPE task PUBLIC \"-//OASIS//DTD DITA Task//EN\"
\"http://docs.oasis-open.org/dita/v1.1/OS/dtd/task.dtd\">\n" +
       "<task id=\"taskId\">\n" +
       "    <title>Task <b>title</b></title>\n" +
       "    <prolog/>\n" +
       "    <taskbody>\n" +
       "        <context>\n" +
       "            <p>Context for the current task</p>\n" +
       "        </context>\n" +
       "        <steps>\n" +
       "            <step>\n" +
       "                <cmd>Task step.</cmd>\n" +
       "            </step>\n" +
       "        </steps>\n" +
       "    </taskbody>\n" +
       "</task>\n" +
       "", getCurrentEditorXMLContent());
   }
}
```

Chapter

10

API Frequently Asked Questions (API FAQ)

This section contains answers to common questions regarding the Oxygen XML Author customisations using the *Oxygen SDK*, *Author Component*, or *Plugins*.

For additional questions, *contact us*. The preferred approach is via email because API questions must be analysed thoroughly. We also provide code snippets in case they are required.

To stay up-to-date with the latest API changes, discuss issues and ask for solutions from other developers working with the Oxygen SDK, register to the *oXygen-SDK mailing list*.

Difference Between a Document Type (Framework) and a Plugin Extension

Question

What is the difference between a Document Type (Framework) and a Plugin Extension?

Answer

Two ways of customising the application are possible:

1. Implementing a plugin.

A plugin serves a general purpose and influences any type of XML file that you open in Oxygen XML Author.

For the Oxygen XML AuthorPlugins API, Javadoc, samples, and documentation, go to
http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins

2. Creating or modifying the document type which is associated to your specific XML vocabulary.

This document type is used to provide custom actions for your type of XML files and to mount them on the toolbar, menus, and contextual menus.

For example, if the application end users are editing DITA, all the toolbar actions which are specific for DITA are provided by the DITA Document Type. If you look in the Oxygen XML Author Preferences->"Document Type Association" there is a "DITA" document type.

If you *edit that document type* in Oxygen XML Author you will see that it has an Author tab in which it defines all custom DITA actions and adds them to the toolbars, main menus, contextual menus.

For information on developing your own document types, see *Authoring Customization Guide* on page 321.

If you look on disk in the:

```
[OXYGEN_DIR]\frameworks\dita
```

folder there is a file called dita.framework. That file gets updated when you edit a document type from the Oxygen XML Author Preferences. Then you can share that updated file with all users.

The same folder contains some JAR libraries. These libraries contain custom Java operations which are called when the user presses certain toolbar actions.

We have an Oxygen SDK which contains the Java sources from all the DITA Java customizations:

http://www.oxygenxml.com/oxygen_sdk.html#XML_Editor_Authoring_SDK

> **Important:** It is possible for a plugin to share the same classes with a framework. For further details, go to *How to Share the Classloader Between a Framework and a Plugin*.

Dynamically Modify the Content Inserted by the Author

Question

Is there a way to insert typographic quotation marks instead of double quotes?

Answer

By using the API you can set a document filter to change the text that is inserted in the Author document. You can use this method to change the insertion of double quotes with the typographic quotes.

Here is some sample code:

```
authorAccess.getDocumentController().setDocumentFilter(new AuthorDocumentFilter() {
  /**
   * @see
ro.sync.ecss.extensions.api.AuthorDocumentFilter#insertText(ro.sync.ecss.extensions.api.AuthorDocumentFilterBypass,
 int, java.lang.String)
   */
  @Override
  public void insertText(AuthorDocumentFilterBypass filterBypass, int offset, String toInsert) {
    if(toInsert.length() == 1 && "\"".equals(toInsert)) {
      //User typed a quote but he actually needs a smart quote.
      //So we either have to add \u201E (start smart quote)
      //Or we add \u201C (end smart quote)
      //Depending on whether we already have a start smart quote inserted in the current paragraph.

      try {
        AuthorNode currentNode = authorAccess.getDocumentController().getNodeAtOffset(offset);
        int startofTextInCurrentNode = currentNode.getStartOffset();
        if(offset > startofTextInCurrentNode) {
          Segment seg = new Segment();
          authorAccess.getDocumentController().getChars(startofTextInCurrentNode, offset -
startofTextInCurrentNode, seg);
          String previosTextInNode = seg.toString();
          boolean insertStartQuote = true;
          for (int i = previosTextInNode.length() - 1; i >= 0; i--) {
            char ch = previosTextInNode.charAt(i);
            if('\u201C' == ch) {
              //Found end of smart quote, so yes, we should insert a start one
              break;
            } else if('\u201E' == ch) {
              //Found start quote, so we should insert an end one.
              insertStartQuote = false;
              break;
            }
          }

          if(insertStartQuote) {
            toInsert = "\u201E";
          } else {
            toInsert = "\u201C";
          }
        }
      } catch (BadLocationException e) {
        e.printStackTrace();
      }
    }
    System.err.println("INSERT TEXT |" + toInsert + "|");
    super.insertText(filterBypass, offset, toInsert);
  }
});
```

You can find the online Javadoc for `AuthorDocumentFilter` API here:
http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/ro/sync/ecss/extensions/api/AuthorDocumentFilter.html

An alternative to using a document filtering is the use of a
`ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandlerAdapter` which has clear
callbacks indicating the source from where the API is called (Paste, Drag and Drop, Typing).

Split Paragraph on Enter (Instead of Showing Content Completion List)

Question

How to split the paragraph on Enter instead of showing the content completion list?

Answer

To obtain this behaviour, *edit your Document Type* and in the `Author` tab, `Actions` tab, add your own split action.
This action must have the **Enter** shortcut key associated and must trigger your own custom operation which handles
the split.

So, when you press **Enter**, your Java operation is invoked and it will be your responsibility to split the paragraph using
the current API (probably creating a document fragment from the caret offset to the end of the paragraph, removing the
content and then inserting the created fragment after the paragraph).

This solution has as a drawback. Oxygen XML Author hides the content completion window when you press **Enter**. If you want to show allowed child elements at that certain offset, implement your own content proposals window using the `ro.sync.ecss.extensions.api.AuthorSchemaManager` API to use information from the associated schema.

Impose Custom Options for Authors

Question

How to enable **Track Changes** at startup?

Answer

There are two ways to enable **Track Changes** for every document that you open:

1. You could *customise the default options* which are used by your authors and set the *Track Changes Initial State option* to Always On.
2. Use the API to toggle the Track Changes state after a document is opened in **Author** mode:

```
// Check the current state of Track Changes
boolean trackChangesOn = authorAccess.getReviewController().isTrackingChanges();
if (!trackChangesOn) {
  // Set Track Changes state to On
  authorAccess.getReviewController().toggleTrackChanges();
}
```

Highlight Content

Question

How can we add custom highlights to the Author document content?

Answer

There are two types of highlights you can add:

1. Not Persistent Highlights. Such highlights are removed when the document is closed and then re-opened.

   You can use the following API method:

   ```
   ro.sync.exml.workspace.api.editor.page.author.WSAuthorEditorPageBase.getHighlighter()
   ```

   to obtain an *AuthorHighlighter* which allows you to add a highlight between certain offsets with a certain painter.

   For example you can use this support to implement your custom spell checker.

2. Persistent Highlights. Such highlights are saved in the XML content as processing instructions.

   You can use the following API method:

   ```
   ro.sync.exml.workspace.api.editor.page.author.WSAuthorEditorPageBase.getPersistentHighlighter()
   ```

   to obtain an *AuthorPersistentHighlighter* which allows you to add a persistent highlight between certain offsets and containing certain custom properties and render it with a certain painter.

   For example you can use this support to implement your own way of adding review comments.

How Do I Add My Custom Actions to the Contextual Menu?

The API methods `WSAuthorEditorPageBase.addPopUpMenuCustomizer` and `WSTextEditorPage.addPopUpMenuCustomizer` allow you to customize the contextual menu shown either in the Author or in the Text modes. The API is available both in the standalone application and in the Eclipse plugin.

Here's an elegant way to add from your Eclipse plugin extension actions to the Author page:

1. Create a pop-up menu customizer implementation:

```
import org.eclipse.jface.action.ContributionManager;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.menus.IMenuService;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.structure.AuthorPopupMenuCustomizer;
/**
 * This class is used to create the possibility to attach certain
 * menuContributions to the {@link ContributionManager}, which is used for the
 * popup menu in the Author Page of the Oxygen Editor.<br />
 * You just need to use the org.eclipse.ui.menus extension and add a
 * menuContribution with the locationURI: <b>menu:oxygen.authorpage</b>
 */
public class OxygenAuthorPagePopupMenuCustomizer implements
        AuthorPopupMenuCustomizer {

    @Override
    public void customizePopUpMenu(Object menuManagerObj,
            AuthorAccess authoraccess) {
        if (menuManagerObj instanceof ContributionManager) {
            ContributionManager contributionManager = (ContributionManager) menuManagerObj;
            IMenuService menuService = (IMenuService) PlatformUI.getWorkbench()
                    .getActiveWorkbenchWindow().getService(IMenuService.class);

            menuService.populateContributionManager(contributionManager,
                    "menu:oxygen.authorpage");
            contributionManager.update(true);
        }
    }
}
```

2. Add a workbench listener and add the pop-up customizer when an editor is opened in the Author page:

```
Workbench.getInstance().getActiveWorkbenchWindow().getPartService().addPartListener(
    new IPartListener() {
      @Override
      public void partOpened(IWorkbenchPart part) {
        if(part instanceof ro.sync.exml.workspace.api.editor.WSEditor) {
          WSEditorPage currentPage = ((WSEditor)part).getCurrentPage();
          if(currentPage instanceof WSAuthorEditorPage) {
            ((WSAuthorEditorPage)currentPage).addPopUpMenuCustomizer(new OxygenAuthorPagePopupMenuCustomizer());

          }
        }
      }
      ........
    });
```

3. Implement the extension point in your `plugin.xml`:

```
<extension
      point="org.eclipse.ui.menus">
    <menuContribution
        allPopups="false"
        locationURI="menu:oxygen.authorpage">
      <command
          commandId="eu.doccenter.kgu.client.tagging.removeTaggingFromOxygen"
          style="push">
      </command>
    </menuContribution>
</extension>
```

Adding Custom Callouts

Question

I'd like to highlight validation errors, instead of underlining them, for example changing the text background color to light red (or yellow).Also I like to let oxygen write a note about the error type into the author view directly at the error position, like " [value "text" not allowed for attribute "type"] ". Is this possible using the API?

Answer

The Plugins API allows setting a `ValidationProblemsFilter` which gets notified when automatic validation errors are available. Then you can map each of the problems to an offset range in the Author page using the API `WSTextBasedEditorPage.getStartEndOffsets(DocumentPositionedInfo)`. For each of those offsets you can add either persistent or non-persistent highlights. If you add persistent highlights you can also customize callouts to appear for each of them, the downside is that they need to be removed before the document gets saved. The end result would look something like:

Here is a small working example:

```
/**
 * Plugin extension - workspace access extension.
 */
public class CustomWorkspaceAccessPluginExtension
                      implements WorkspaceAccessPluginExtension {

  /**
   * @see ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension
   *        #applicationStarted(ro.sync.exml.workspace.api.standalone.StandalonePluginWorkspace)
   */
  public void applicationStarted(final StandalonePluginWorkspace pluginWorkspaceAccess) {
    pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
      /**
       * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorOpened(java.net.URL)
       */
      @Override
      public void editorOpened(URL editorLocation) {
        final WSEditor currentEditor = pluginWorkspaceAccess.getEditorAccess(editorLocation,
StandalonePluginWorkspace.MAIN_EDITING_AREA);
        WSEditorPage currentPage = currentEditor.getCurrentPage();
        if(currentPage instanceof WSAuthorEditorPage) {
          final WSAuthorEditorPage currentAuthorPage = (WSAuthorEditorPage)currentPage;
          currentAuthorPage.getPersistentHighlighter().setHighlightRenderer(new PersistentHighlightRenderer()
{

            @Override
            public String getTooltip(AuthorPersistentHighlight highlight) {
              return highlight.getClonedProperties().get("message");
            }
            @Override
```

```
        public HighlightPainter getHighlightPainter(AuthorPersistentHighlight highlight) {
          //Depending on severity could have different color.
          ColorHighlightPainter painter = new ColorHighlightPainter(Color.COLOR_RED, -1, -1);
          painter.setBgColor(Color.COLOR_RED);
          return painter;
        }
      });
      currentAuthorPage.getReviewController()
        .getAuthorCalloutsController().setCalloutsRenderingInformationProvider(
            new CalloutsRenderingInformationProvider() {
        @Override
        public boolean shouldRenderAsCallout(AuthorPersistentHighlight highlight) {
          //All custom highlights are ours
          return true;
        }
        @Override
        public AuthorCalloutRenderingInformation getCalloutRenderingInformation(
            final AuthorPersistentHighlight highlight) {
          return new AuthorCalloutRenderingInformation() {
            @Override
            public long getTimestamp() {
              //Not interesting
              return -1;
            }
            @Override
            public String getContentFromTarget(int limit) {
              return "";
            }
            @Override
            public String getComment(int limit) {
              return highlight.getClonedProperties().get("message");
            }
            @Override
            public Color getColor() {
              return Color.COLOR_RED;
            }
            @Override
            public String getCalloutType() {
              return "Problem";
            }
            @Override
            public String getAuthor() {
              return "";
            }
            @Override
            public Map<String, String> getAdditionalData() {
              return null;
            }
          };
        }
      });
      currentEditor.addValidationProblemsFilter(new ValidationProblemsFilter() {
        List<int[]> lastStartEndOffsets = new ArrayList<int[]>();
        /**
         * @see ro.sync.exml.workspace.api.editor.validation.ValidationProblemsFilter
             #filterValidationProblems(ro.sync.exml.workspace.api.editor.validation.ValidationProblems)

         */
        @Override
        public void filterValidationProblems(ValidationProblems validationProblems) {
          List<int[]> startEndOffsets = new ArrayList<int[]>();
          List<DocumentPositionedInfo> problemsList = validationProblems.getProblemsList();
          if(problemsList != null) {
            for (int i = 0; i < problemsList.size(); i++) {
              try {
                startEndOffsets.add(currentAuthorPage.getStartEndOffsets(problemsList.get(i)));
              } catch (BadLocationException e) {
                e.printStackTrace();
              }
            }
          }
            if(lastStartEndOffsets.size() != startEndOffsets.size()) {
              //Continue
            } else {
              boolean equal = true;
              for (int i = 0; i < startEndOffsets.size(); i++) {
                int[] o1 = startEndOffsets.get(i);
                int[] o2 = lastStartEndOffsets.get(i);
                if(o1 == null && o2 == null) {
                  //Continue
                } else  if(o1 != null && o2 != null
                    && o1[0] == o2[0] && o1[1] == o2[1]){
                  //Continue
                } else {
                  equal = false;
                  break;
                }
              }
              if(equal) {
```

```
                                //Same list of problems already displayed.
                                return;
                            }
                        }
                        //Keep last used offsets.
                        lastStartEndOffsets = startEndOffsets;
                    try {
                        if(! SwingUtilities.isEventDispatchThread()) {
                            SwingUtilities.invokeAndWait(new Runnable() {
                                @Override
                                public void run() {
                                    //First remove all custom highlights.
                                    currentAuthorPage.getPersistentHighlighter().removeAllHighlights();
                                }
                            });
                        }
                    } catch (InterruptedException e1) {
                        e1.printStackTrace();
                    } catch (InvocationTargetException e1) {
                        e1.printStackTrace();
                    }
                    if(problemsList != null) {
                        for (int i = 0; i < problemsList.size(); i++) {
                            //A reported problem (could be warning, could be error).
                            DocumentPositionedInfo dpi = problemsList.get(i);
                            try {
                                final int[] currentOffsets = startEndOffsets.get(i);
                                if(currentOffsets != null) {
                                    //These are offsets in the Author content.
                                    final LinkedHashMap<String, String> highlightProps = new LinkedHashMap<String,
String>();

                                    highlightProps.put("message", dpi.getMessage());
                                    highlightProps.put("severity", dpi.getSeverityAsString());
                                    if(! SwingUtilities.isEventDispatchThread()) {
                                        SwingUtilities.invokeAndWait(new Runnable() {
                                            @Override
                                            public void run() {
                                                currentAuthorPage.getPersistentHighlighter().addHighlight(
                                                        currentOffsets[0], currentOffsets[1] - 1, highlightProps);
                                            }
                                        });
                                    }
                                }
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            } catch (InvocationTargetException e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }
            });
            currentEditor.addEditorListener(new WSEditorListener() {
                /**
                 * @see ro.sync.exml.workspace.api.listeners.WSEditorListener#editorAboutToBeSavedVeto(int)
                 */
                @Override
                public boolean editorAboutToBeSavedVeto(int operationType) {
                    try {
                        if(! SwingUtilities.isEventDispatchThread()) {
                            SwingUtilities.invokeAndWait(new Runnable() {
                                @Override
                                public void run() {
                                    //Remove all persistent highlights before saving
                                    currentAuthorPage.getPersistentHighlighter().removeAllHighlights();
                                }
                            });
                        }
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    } catch (InvocationTargetException e) {
                        e.printStackTrace();
                    }
                    return true;
                }
            });
        }
    }
  }, StandalonePluginWorkspace.MAIN_EDITING_AREA);
}


/**
 * @see ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension#applicationClosing()
 */
public boolean applicationClosing() {
    return true;
}
}
```

Change the DOCTYPE of an Opened XML Document

Question

How to change the DOCTYPE of a document opened in the **Author** mode?

Answer

The following API:

```
ro.sync.ecss.extensions.api.AuthorDocumentController.getDoctype()
```

allows you to get the DOCTYPE of the current XML file opened in the Author page.

There is also an API method available which would allow you to set the DOCTYPE back to the XML:

```
ro.sync.ecss.extensions.api.AuthorDocumentController.setDoctype(AuthorDocumentType)
```

Here is an example of how this solution would work:

```
AuthorDocumentType dt = new AuthorDocumentType("article", "testSystemID", "testPublicID",
            "<!DOCTYPE article PUBLIC \"testPublicID\" \"testSystemID\">");
docController.setDoctype(dt);
```

Basically you could take the entire content from the existing DOCTYPE,

```
ro.sync.ecss.extensions.api.AuthorDocumentType.getContent()
```

modify it to your needs, and create another `AuthorDocumentType` object with the new content and with the same public, system IDs.

For example you could use this API is you want to add unparsed entities in the XML `DOCTYPE`.

Customize the Default Application Icons for Toolbars/Menus

Question

How can we change the default icons used for the application built-in actions?

Answer

If you look inside the main JAR library `[OXYGEN_DIR]\lib\oxygen.jar` or `[OXYGEN_DIR]\lib\author.jar` it contains an `images` folder in which all the images which we use for our buttons, menus, and toolbars exist.

In order to overwrite them with your own creations:

1. In the `[OXYGEN_DIR]\lib` directory create a folder called `endorsed`;
2. In the `endorsed` folder create another folder called `images`;
3. Add your own images in the `images` folder.

You can use this mechanism to overwrite any kind of resource located in the main Oxygen JAR library. The folder structure in the endorsed directory and in the main Oxygen JAR must be identical.

Disable Context-Sensitive Menu Items for Custom Author Actions

Question

Is there a way to disable menu items for custom Author actions depending on the cursor context?

Answer

By default Oxygen XML Author does not toggle the enabled/disabled states for actions based on whether the activation XPath expressions for that certain Author action are fulfilled. This is done because the actions can be many and evaluating XPath expression on each caret move can lead to performance problems. But if you have your own `ro.sync.ecss.extensions.api.ExtensionsBundle` implementation you can overwrite the method:

```
ro.sync.ecss.extensions.api.ExtensionsBundle.createAuthorExtensionStateListener()
```

and when the extension state listener gets activated you can use the API like:

```
/**
 * @see
ro.sync.ecss.extensions.api.AuthorExtensionStateListener#activated(ro.sync.ecss.extensions.api.AuthorAccess)
 */
public void activated(final AuthorAccess authorAccess) {

  //Add a caret listener to enable/disable extension actions:
  authorAccess.getEditorAccess().addAuthorCaretListener(new AuthorCaretListener() {
    @Override
    public void caretMoved(AuthorCaretEvent caretEvent) {
      try {
        Map<String, Object> authorExtensionActions =
authorAccess.getEditorAccess().getActionsProvider().getAuthorExtensionActions();
        //Get the action used to insert a paragraph. It's ID is "paragraph"
        AbstractAction insertParagraph = (AbstractAction) authorExtensionActions.get("paragraph");
        //Evaluate an XPath expression in the context of the current node in which the caret is located
        Object[] evaluateXPath = authorAccess.getDocumentController().evaluateXPath(".[ancestor-or-self::p]",
false, false, false, false);
        if(evaluateXPath != null && evaluateXPath.length > 0 && evaluateXPath[0] != null) {
          //We are inside a paragraph, disable the action.
          insertParagraph.setEnabled(false);
        } else {
          //Enable the action
          insertParagraph.setEnabled(true);
        }
      } catch (AuthorOperationException e) {
        e.printStackTrace();
      }
    }
  });
```

When the extension is deactivated you should remove the caret listener in order to avoid adding multiple caret listeners which perform the same functionality.

Dynamic Open File in Oxygen XML Author Distributed via JavaWebStart

Question

How can we dynamically open a file in an Oxygen XML Author distributed via JWS?

Answer

The JWS packager ANT build file which comes with Oxygen XML Author signs by default the JNLP file (this means that a copy of it is included in the main JAR library) in this step:

```
<copy file="${outputDir}/${packageName}/${productName}.jnlp" tofile="${home}/JNLP-INF/APPLICATION.JNLP"/>
```

Signing the JNLP file indeed means that it is impossible to automatically generate a JNLP file containing some dynamic arguments.

But the JNLP does not need to be signed. Indeed the user probably receives this information when launching the application but at least in this way you should be able to dynamically generate a JNLP file via a PHP script based on the URL which was clicked by the user.

The generated JNLP would then take as argument the URL which needs to be opened when Oxygen XML Author starts.

Maybe a different approach (more complicated though) would be to have the JNLP file signed and always refer as a URL argument a location like this:

```
http://path/to/server/redirectEditedURL.php
```

When the URL gets clicked on the client side you would also call a PHP script on the server side which would update the redirect location for redirectEditedURL.php to point to the clicked XML resource. Then the opened Oxygen XML Author would try to connect to the redirect PHP and be redirected to open the XML.

Change the Default Track Changes (Review) Author Name

Question

How can we change the default author name used for Track Changes in the Author Component?

Answer

The Track Changes (Review) Author name is determined in the following order:

1. **API** - The review user name can be imposed through the following API:

    ```
    ro.sync.ecss.extensions.api.AuthorReviewController.setReviewerAuthorName(String)
    ```

2. **Options** - If the author name was not imposed from the API, it is determined from the Author option set from the *Review preferences page.*
3. **System properties** - If the author name was not imposed from the API or from the application options then the following system property is used:

    ```
    System.getProperty("user.name")
    ```

So, to impose the Track Changes author, use one of the following approaches:

1. Use the API to impose the reviewer Author name. Here is the online Javadoc of this method:
    http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/ro/sync/ecss/extensions/api/AuthorReviewController.html#setReviewerAuthorName(java.lang.String)
2. *Customise the default options* and set a specific value for the reviewer Author name option.
3. Set the value of user.name system property when the applet is initialising and before any document is loaded.

Multiple Rendering Modes for the Same Author Document

Question

How can we add multiple buttons, each showing different visualisation mode of the same Author document (by associating additional/different CSS style sheet)?

Answer

In the toolbar of the **Author** mode there is a drop-down button which contains alternative CSS styles for the same document. To add an alternative CSS stylesheet go to Oxygen XML Author Preferences->Document Type Association page, select the document type associated with your documents and press Edit. In the Document Type dialog that appears go to "Author" tab, "CSS" tab and add there references to alternate CSS stylesheets.

For example, one of the alternate CSS stylesheets that we offer for `DITA` document type is located here:

```
[OXYGEN_DIR]/frameworks/dita/css_classed/hideColspec.css
```

If you open it, you will see that it imports the main CSS and then adds selectors of its own.

Obtain a DOM Element from an `AuthorNode` or `AuthorElement`

Question

Can a DOM Element be obtained from an `AuthorNode` or an `AuthorElement`?

Answer

No, a DOM Element cannot be obtained from an `AuthorNode` or an `AuthorElement`. The `AuthorNode` structure is also hierarchical but the difference is that all the text content is kept in a single text buffer instead of having individual text nodes.

We have an image in the Javadoc which explains the situation:*http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/ro/sync/ecss/extensions/api/node/AuthorDocumentFragment.html*

Print Document Within the Author Component

Question

Can a document be printed within the Author Component?

Answer

You can use the following API method to either print the Author document content to the printer or to show the Print Preview dialog, depending on the `preview` parameter value:

```
AuthorComponentProvider.print(boolean preview)
```

Here is the online Javadoc for this method:
http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/ro/sync/ecss/extensions/api/component/AuthorComponentProvider.html#print(boolean)

Running XSLT or XQuery Transformations

Question

Can I run XSL 2.0 / 3.0 transformation with Saxon EE using the oXygen SDK?

Answer

The API class `ro.sync.exml.workspace.api.util.XMLUtilAccess` allows you to create an XSLT Transformer which implements the JAXP interface `javax.xml.transform.Transformer`. Then this type of transformer can be used to transform XML. Here's just an example of transforming when you have an `AuthorAccess` API available:

```
InputSource is = new org.xml.sax.InputSource(URLUtil.correct(new File("test/personal.xsl")).toString());
xslSrc = new SAXSource(is);
javax.xml.transform.Transformer transformer = authorAccess.getXMLUtilAccess().createXSLTTransformer(xslSrc,
null, AuthorXMLUtilAccess.TRANSFORMER_SAXON_ENTERPRISE_EDITION);
transformer.transform(new StreamSource(new File("test/personal.xml")), new StreamResult(new
File("test/personal.html")));
```

If you want to create the transformer from the plugins side, you can use this method instead:
`ro.sync.exml.workspace.api.PluginWorkspace.getXMLUtilAccess()`.

Use Different Rendering Styles for Entity References, Comments or Processing Instructions

Question

Is there a way to display entity references in the **Author** mode without the distinct gray background and tag markers?

Answer

There is a built-in CSS stylesheet in the Oxygen XML Author libraries which is used when styling content in the **Author** mode, no matter what CSS you use. This CSS has the following content:

```
@namespace oxy url('http://www.oxygenxml.com/extensions/author');
@namespace xi "http://www.w3.org/2001/XInclude";
@namespace xlink "http://www.w3.org/1999/xlink";
@namespace svg "http://www.w3.org/2000/svg";
@namespace mml "http://www.w3.org/1998/Math/MathML";

oxy|document {
    display:block !important;
}

oxy|cdata {
    display:morph !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|processing-instruction {
    display:block !important;
    color: rgb(139, 38, 201) !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|comment {
    display:morph !important;
    color: rgb(0, 100, 0) !important;
    background-color:rgb(255, 255, 210) !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|reference:before,
oxy|entity[href]:before{
  link: attr(href) !important;
  text-decoration: underline !important;
  color: navy !important;

  margin: 2px !important;
  padding: 0px !important;
}

oxy|reference:before {
  display: morph !important;
  content: url(../images/editContent.gif) !important;
}

oxy|entity[href]:before{
  display: morph !important;
  content: url(../images/editContent.gif) !important;
}

oxy|reference,
oxy|entity {
    editable:false !important;
    background-color: rgb(240, 240, 240) !important;
    margin:0px !important;
    padding: 0px !important;
}
```

```
oxy|reference {
    display:morph !important;
}

oxy|entity {
    display:morph !important;
}

oxy|entity[href] {
  border: 1px solid rgb(175, 175, 175) !important;
  padding: 0.2em !important;
}

xi|include {
    display:block !important;
    margin-bottom: 0.5em !important;
    padding: 2px !important;
}
xi|include:before,
xi|include:after{
    display:inline !important;
    background-color:inherit !important;
    color:#444444 !important;
    font-weight:bold !important;
}

xi|include:before {
    content:url(../images/link.gif) attr(href)  !important;
    link: attr(href) !important;
}
xi|include[xpointer]:before {
    content:url(../images/link.gif) attr(href) " " attr(xpointer) !important;
    link: oxy_concat(attr(href), "#", attr(xpointer)) !important;
}

xi|fallback {
    display:morph !important;
    margin: 2px !important;
    border: 1px solid #CB0039 !important;
}

xi|fallback:before {
    display:morph !important;
    content:"XInclude fallback: " !important;
    color:#CB0039 !important;
}

oxy|doctype {
    display:block !important;
    background-color: transparent !important;
    color:blue !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 2px !important;
}

oxy|error {
    display:morph !important;
    editable:false !important;
    white-space:pre !important;
    color: rgb(178, 0, 0) !important;
    font-weight:bold !important;
}

*[xlink|href]:before {
    content:url(../images/link.gif);
    link: attr(xlink|href) !important;
}


/*No direct display of the MathML and SVG images.*/
svg|svg{
 display:inline !important;
    white-space: trim-when-ws-only;
}
svg|svg svg|*{
    display:none !important;
    white-space:normal;
}

mml|math{
 display:inline !important;
    white-space: trim-when-ws-only;
}
mml|math mml|*{
    display:none !important;
    white-space: normal;
}
```

In the CSS used for rendering the XML in **Author** mode do the following:

• import the special Author namespace;

• use a special selector to customize the entity node.

Example:

```
@namespace oxy url('http://www.oxygenxml.com/extensions/author');
oxy|entity {
    background-color: inherit !important;
    margin:0px !important;
    padding: 0px !important;
     -oxy-display-tags:none;
}
```

You can overwrite styles in the predefined CSS in order to custom style comments, processing instructions and *CData* sections. You can also customize the way in which xi:include elements are rendered.

Insert an Element with all the Required Content

Question

I'm inserting a DITA *image* XML element, using the Author API, which points to a certain resource and has required content. Can the required content be automatically inserted by the application?

Answer

The API ro.sync.ecss.extensions.api.AuthorSchemaManager can propose valid elements which can be inserted at the specific offset. Using the method AuthorSchemaManager.createAuthorDocumentFragment(CIElement) you can convert the proposed elements to document fragments (which have all the required content filled in) which can then be inserted in the document.

```
AuthorSchemaManager schemaManager = this.authorAccess.getDocumentController().getAuthorSchemaManager();
WhatElementsCanGoHereContext context =
schemaManager.createWhatElementsCanGoHereContext(this.authorAccess.getEditorAccess().getCaretOffset());
List<CIElement> possibleElementsAtCaretPosition = schemaManager.whatElementsCanGoHere(context);
loop: for (int i = 0; i < possibleElementsAtCaretPosition.size(); i++) {
  CIElement possibleElement = possibleElementsAtCaretPosition.get(i);
  List<CIAttribute> attrs = possibleElement.getAttributes();
  if(attrs != null) {
    for (int j = 0; j < attrs.size(); j++) {
      CIAttribute ciAttribute = attrs.get(j);
      if (ciAttribute.getName().equals("class")) {
        if (ciAttribute.getDefaultValue() != null
            && ciAttribute.getDefaultValue().contains(" topic/image ")) {
          //Found a CIElement for image
          //Create a fragment for it. The fragment contains all required child elements already built.
          AuthorDocumentFragment frag = schemaManager.createAuthorDocumentFragment(possibleElement);
          //Now set the @href to it.
          //Ask the user and obtain a value for the @href
          //Then:

          String href = "test.png";
          List<AuthorNode> nodes = frag.getContentNodes();
          if(!nodes.isEmpty()) {
            AuthorElement imageEl = (AuthorElement) nodes.get(0);
            imageEl.setAttribute("href", new AttrValue(href));
          }
          //And insert the fragment.

this.authorAccess.getDocumentController().insertFragment(this.authorAccess.getEditorAccess().getCaretOffset(),
 frag);
          break loop;
        }
      }
    }
  }
}
```

Obtain the Current Selected Element Using the Author API

Question

If in the **Author** mode, an element is fully selected, I would like to perform an action on it. If not, I would like to perform an action on the node which is located at the caret position. Is this possible via the API?

Answer

When an element is fully selected by the user the selection start and end offsets are actually outside of the node's offset bounds. So using `AuthorDocumentController.getNodeAtOffset` will actually return the parent of the selected node. We have some special API which makes it easier for you to determine this situation: `WSAuthorEditorPageBase.getFullySelectedNode()`.

```
AuthorDocumentController controller = authorPageAccess.getDocumentController();
AuthorAccess authorAccess = authorPageAccess.getAuthorAccess();
int caretOffset = authorAccess.getEditorAccess().getCaretOffset();

AuthorElement nodeAtCaret = (AuthorElement) authorAccess.getEditorAccess().getFullySelectedNode();
if (nodeAtCaret == null) {
     //We have no fully selected node. We can look at the caret offset.
     nodeAtCaret = (AuthorElement) authorAccess.getDocumentController().getNodeAtOffset(caretOffset);
   //Or we could look at the selection start and end, see which node is the parent of each offset and get the
 closest common ancestor.
}
```

Debugging a Plugin Using the Eclipse Workbench

To debug problems in the code of the plugin without having to re-bundle the Java classes of the plugin in a JAR library, follow these steps:

1. Download and unpack an *all platforms standalone version* of Oxygen XML Author/Editor to a folder on your hard drive.

   > **Note:** Name the folder [OXYGEN_DIR].

2. Set up the Oxygen SDK following *this set of instructions*.
3. Create an Eclipse Java Project (let's call it MyPluginProject) from one of the sample plugins (the Workspace Access plugin for example).
4. In the Project root folder, create a folder called myPlugin and add the plugin.xml from the sample plugin in there. Modify the added plugin.xml to add a library reference to the project's classes directory: <library name="../classes"/>.
5. Copy the plugin.dtd from the [OXYGEN_DIR]/plugins folder in the root Project folder.
6. In the Project's build path add external JAR references to all the JAR libraries in the [OXYGEN_DIR]/lib folder. Now your Project should compile successfully.
7. Create a new Java Application configuration for debugging. The Main Class should be ro.sync.exml.Oxygen. The given VM Arguments should be:

```
-Dcom.oxygenxml.app.descriptor=ro.sync.exml.EditorFrameDescriptor -Xmx1024m
-XX:MaxPermSize=384m -Dcom.oxygenxml.editor.plugins.dir=D:\projects\MyPluginProject
```

8. Add a break point in one of the source Java classes.
9. Debug the created configuration. When the code reaches your breakpoint, the debug perspective should take over.

Debugging an Oxygen SDK Extension Using the Eclipse Workbench

To debug problems in the extension code without having to bundle the extension's Java classes in a JAR library, perform the following steps:

1. Download and unpack an *all platforms standalone version* of Oxygen XML Author/Editor to a folder on your hard drive.

   > **Note:** Name the folder [OXYGEN_DIR].

2. Create an Eclipse Java Project (let's call it MySDKProject) with the corresponding Java sources (for example a custom implementation of the ro.sync.ecss.extensions.api.StylesFilter interface).

3. In the Project's build path add external JAR references to all the JAR libraries in the [OXYGEN_DIR]/lib folder. Now your Project should compile successfully.

4. Start the standalone version of Oxygen from the [OXYGEN_DIR] and in the **Document Type Association** Preferences page edit the document type (for example **DITA**). In the **Classpath** tab, add a reference to your Project's classes directory and in the **Extensions** tab, select your custom StylesFilter extension as a value for the **CSS styles filter** property. Close the application to save the changes to the framework file.

5. Create a new Java Application configuration for debugging. The Main Class should be ro.sync.exml.Oxygen. The given VM Arguments should be

```
-Dcom.oxygenxml.app.descriptor=ro.sync.exml.EditorFrameDescriptor -Xmx1024m -XX:MaxPermSize=384m
```

6. Add a break point in one of the source Java classes.

7. Debug the created configuration. When the code reaches your breakpoint, the debug perspective should take over.

Extending the Java Functionality of an Existing Framework (Document Type)

Question

How can I change the way DocBook 4 xref's display in author view based on what element is at the linkend?

Please follow the steps below:

1. Create a Maven Java project and add a dependency on the oXygen classes:

```
<dependency>
    <groupId>com.oxygenxml</groupId>
    <artifactId>oxygen-sdk</artifactId>
    <version>${oxygen.version}</version>
</dependency>
```

   where ${oxygen.version} is the version of Oxygen XML Author.

   Alternatively, if the project does not use Maven, all the transitive dependencies of the above Maven artifact need to be added to the classpath of the project.

2. Also add to the project's class path the: "[OXYGEN_DIR]\frameworks\docbook\docbook.jar".

3. Create a class which extends ro.sync.ecss.extensions.docbook.DocBook4ExtensionsBundle and overwrites the method:
   ro.sync.ecss.extensions.api.ExtensionsBundle#createLinkTextResolver()

4. For your custom resolver implementation you can start from the Java sources of the ro.sync.ecss.extensions.docbook.link.DocbookLinkTextResolver (the Java code for the entire DocBook customization is present in a subfolder in the Author SDK).

5. Pack your extension classes in a JAR file. Copy the JAR to: "[OXYGEN_DIR]\frameworks\docbook\custom.jar".

6. Start Oxygen XML Author.

7. *Open the Preferences dialog* and go to **Document Type Association**. Edit the DocBook 4 document type. In the **Classpath** list add the path to the new JAR. In the extensions list select your custom extension instead of the regular DocBook one.

8. You can rename the document type and also the "docbook" framework folder to something else like "custom_docbook" and share it with others. A document type can also be installed using our *add-on support*.

Controlling XML Serialization in the Author Component

Question

How can I force the Author Component to save the XML with zero indent size and not to break the line inside block-level elements?

Answer

Usually, in a standalone version of Oxygen XML Author, the **Editor** > **Format** and **Editor** > **Format** > **XML** preferences pages allow you to control the way the XML is saved on the disk after you edit it in the **Author** mode.

In the editor application (Standalone or Eclipse-based), you can either bundle a *default set of options* or use the `PluginWorkspace.setGlobalObjectProperty(String, Object)` API:

```
//For not breaking the line
//Long line
pluginWorkspace.setObjectProperty("editor.line.width", new Integer(100000));
//Do not break before inline elements
pluginWorkspace.setObjectProperty("editor.format.indent.inline.elements", false);

//For forcing zero indent
//Force indent settings to be controlled by us
pluginWorkspace.setObjectProperty("editor.detect.indent.on.open", false);
//Zero indent size
pluginWorkspace.setObjectProperty("editor.indent.size.v9.2", 0);
```

In the Author Component, you can either bundle a *fixed set of options*, or use our Java API to set properties which overwrite the default options:

```
//For not breaking the line
//Long line
AuthorComponentFactory.getInstance().setObjectProperty("editor.line.width", new Integer(100000));
//Do not break before inline elements
AuthorComponentFactory.getInstance().setObjectProperty("editor.format.indent.inline.elements", false);

//For forcing zero indent
//Force indent settings to be controlled by us
AuthorComponentFactory.getInstance().setObjectProperty("editor.detect.indent.on.open", false);
//Zero indent size
AuthorComponentFactory.getInstance().setObjectProperty("editor.indent.size.v9.2", 0);
```

How can I add a custom Outline view for editing XML documents in the Text mode?

Let's say you have XML documents like

```
<doc startnumber="15">
    <sec counter="no">
        <info/>
        <title>Introduction</title>
     </sec>
    <sec>
        <title>Section title</title>
      <para>Content</para>
        <sec>
            <title>Section title</title>
                <para>Content</para>
        </sec>
    </sec>
    <sec>
            <title>Section title</title>
        <para>Content</para>
        </sec>
</doc>
```

and you want to display the XML content in a simplified Outline view like:

```
doc "15"
sec Introduction
```

```
sec 15 Section title
sec 15.1 Section title
sec 16 Section title
```

Usually an Outline should have the following characteristics:

1. Double clicking in the Outline the corresponding XML content would get selected.
2. When the caret moves in the opened XML document the Outline would select the proper entry.
3. When modifications occur in the document, the Outline would refresh.

A simple implementation using a Workspace Access plugin type could be something like:

```
/**
 * Simple Outline for the Text mode based on executing XPaths over the text content.
 */
public class CustomWorkspaceAccessPluginExtension implements WorkspaceAccessPluginExtension {
  /**
   * The custom outline list.
   */
  private JList customOutlineList;

  /**
   * Maps outline nodes to ranges in document
   */
  private WSXMLTextNodeRange[] currentOutlineRanges;

  /**
   * The current text page
   */
  private WSXMLTextEditorPage currentTextPage;

  /**
   * Disable caret listener when we select from the caret listener.
   */
  private boolean enableCaretListener = true;

  /**
   * @see
ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension#applicationStarted(ro.sync.exml.workspace.api.standalone.StandalonePluginWorkspace)
   */
  @Override
  public void applicationStarted(final StandalonePluginWorkspace pluginWorkspaceAccess) {
    pluginWorkspaceAccess.addViewComponentCustomizer(new ViewComponentCustomizer() {
      /**
       * @see
ro.sync.exml.workspace.api.standalone.ViewComponentCustomizer#customizeView(ro.sync.exml.workspace.api.standalone.ViewInfo)
       */
      @Override
      public void customizeView(ViewInfo viewInfo) {
        if(
            //The view ID defined in the "plugin.xml"
            "SampleWorkspaceAccessID".equals(viewInfo.getViewID())) {
          customOutlineList = new JList();
          //Render the content in the Outline.
          customOutlineList.setCellRenderer(new DefaultListCellRenderer() {
            /**
             * @see javax.swing.DefaultListCellRenderer#getListCellRendererComponent(javax.swing.JList,
java.lang.Object, int, boolean, boolean)
             */
            @Override
            public Component getListCellRendererComponent(JList<?> list, Object value, int index,
                boolean isSelected, boolean cellHasFocus) {
              JLabel label = (JLabel) super.getListCellRendererComponent(list, value, index, isSelected,
cellHasFocus);
              String val = null;
              if(value instanceof Element) {
                Element element = ((Element)value);
                val = element.getNodeName();
                if(!"".equals(element.getAttribute("startnumber"))) {
                  val += " " + "'" + element.getAttribute("startnumber") + "'";
                }
                NodeList titles = element.getElementsByTagName("title");
                if(titles.getLength() > 0) {
                  val += " \"" + titles.item(0).getTextContent() + "\"";
                }
              }
              label.setText(val);
              return label;
            }
          });
          //When we click a node, select it in the text page.
          customOutlineList.addMouseListener(new MouseAdapter() {
            @Override
```

```
                public void mouseClicked(MouseEvent e) {
                    if(SwingUtilities.isLeftMouseButton(e) && e.getClickCount() == 2) {
                        int sel = customOutlineList.getSelectedIndex();
                        enableCaretListener = false;
                        try {

currentTextPage.select(currentTextPage.getOffsetOfLineStart(currentOutlineRanges[sel].getStartLine()) +
currentOutlineRanges[sel].getStartColumn() - 1,
                            currentTextPage.getOffsetOfLineStart(currentOutlineRanges[sel].getEndLine()) +
currentOutlineRanges[sel].getEndColumn());
                        } catch (BadLocationException e1) {
                            e1.printStackTrace();
                        }
                        enableCaretListener = true;
                    }
                }
            });
            viewInfo.setComponent(new JScrollPane(customOutlineList));
            viewInfo.setTitle("Custom Outline");
          }
        }
      });

    pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
        /**
         * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorOpened(java.net.URL)
         */
        @Override
        public void editorOpened(URL editorLocation) {
          //An editor was opened
          WSEditor editorAccess = pluginWorkspaceAccess.getEditorAccess(editorLocation,
StandalonePluginWorkspace.MAIN_EDITING_AREA);
          if(editorAccess != null) {
            WSEditorPage currentPage = editorAccess.getCurrentPage();
            if(currentPage instanceof WSXMLTextEditorPage) {
              //User editing in Text mode an opened XML document.
              final WSXMLTextEditorPage xmlTP = (WSXMLTextEditorPage) currentPage;
              //Reconfigure outline on each change.
              xmlTP.getDocument().addDocumentListener(new DocumentListener() {
                @Override
                public void removeUpdate(DocumentEvent e) {
                  reconfigureOutline(xmlTP);
                }
                @Override
                public void insertUpdate(DocumentEvent e) {
                  reconfigureOutline(xmlTP);
                }
                @Override
                public void changedUpdate(DocumentEvent e) {
                  reconfigureOutline(xmlTP);
                }
              });
              JTextArea textComponent = (JTextArea) xmlTP.getTextComponent();
              textComponent.addCaretListener(new CaretListener() {
                @Override
                public void caretUpdate(CaretEvent e) {
                  if(currentOutlineRanges != null && currentTextPage != null && enableCaretListener) {
                    enableCaretListener = false;
                    //Find the node to select in the outline.
                    try {
                      int line = xmlTP.getLineOfOffset(e.getDot());
                      for (int i = currentOutlineRanges.length - 1; i >= 0; i--) {
                        if(line > currentOutlineRanges[i].getStartLine() && line <
currentOutlineRanges[i].getEndLine()) {
                          customOutlineList.setSelectedIndex(i);
                          break;
                        }
                      }
                    } catch (BadLocationException e1) {
                      e1.printStackTrace();
                    }
                    enableCaretListener = true;
                  }
                }
              });
            }
          }
        }
        /**
         * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorActivated(java.net.URL)
         */
        @Override
        public void editorActivated(URL editorLocation) {
          //An editor was selected, reconfigure the common outline
          WSEditor editorAccess = pluginWorkspaceAccess.getEditorAccess(editorLocation,
StandalonePluginWorkspace.MAIN_EDITING_AREA);
          if(editorAccess != null) {
            WSEditorPage currentPage = editorAccess.getCurrentPage();
            if(currentPage instanceof WSXMLTextEditorPage) {
```

```
                //User editing in Text mode an opened XML document.
                WSXMLTextEditorPage xmlTP = (WSXMLTextEditorPage) currentPage;
                reconfigureOutline(xmlTP);
            }
        }
    }
    }, StandalonePluginWorkspace.MAIN_EDITING_AREA);
  }

  /**
   * Reconfigure the outline
   *
   * @param xmlTP The XML Text page.
   */
  protected void reconfigureOutline(final WSXMLTextEditorPage xmlTP) {
    try {
      //These are DOM nodes.
      Object[] evaluateXPath = xmlTP.evaluateXPath("//doc | //sec");
      //These are the ranges each node takes in the document.
      currentOutlineRanges = xmlTP.findElementsByXPath("//doc | //sec");
      currentTextPage = xmlTP;
      DefaultListModel listModel = new DefaultListModel();
      if(evaluateXPath != null) {
        for (int i = 0; i < evaluateXPath.length; i++) {
          listModel.addElement(evaluateXPath[i]);
        }
      }
      customOutlineList.setModel(listModel);
    } catch(XPathException ex) {
      ex.printStackTrace();
    }
  }

  /**
   * @see ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension#applicationClosing()
   */
  @Override
  public boolean applicationClosing() {
    return true;
  }
}
```

Dynamically Adding Form Controls Using a StylesFilter

Usually, a form control is added from the CSS using *The oxy_editor() Function* on page 414. However, in some cases you don't have all the information you need to properly initialize the form control at CSS level. In these cases you can add the form controls by using the API, more specifically `ro.sync.ecss.extensions.api.StylesFilter`.

For instance, let's assume that we want a combo box form control and the values to populate the combo are specified inside a file (for a more interesting scenario we could imagine that they come from a database). Here is how to add the form control from the API:

```
public class SDFStylesFilter implements StylesFilter {

  public Styles filter(Styles styles, AuthorNode authorNode) {
    if(authorNode.getType() == AuthorNode.NODE_TYPE_PSEUDO_ELEMENT
        && "before".equals(authorNode.getName())) {
      authorNode = authorNode.getParent();
      if ("country".equals(authorNode.getName())) {
        // This is the BEFORE pseudo element of the "country" element.
        // Read the supported countries from the configuration file.
        Map<String, Object> formControlArgs = new HashMap<String, Object>();
        formControlArgs.put(InplaceEditorArgumentKeys.PROPERTY_EDIT, "#text");
        formControlArgs.put(InplaceEditorArgumentKeys.PROPERTY_TYPE, InplaceEditorArgumentKeys.TYPE_COMBOBOX);

        // This will be a comma separated enumeration: France, Spain, Great Britain
        String countries = readCountriesFromFile();
        formControlArgs.put(InplaceEditorArgumentKeys.PROPERTY_VALUES, countries);
        formControlArgs.put(InplaceEditorArgumentKeys.PROPERTY_EDITABLE, "false");

        // We also add a label in form of the form control.
        Map<String, Object> labelProps = new HashMap<String, Object>();
        labelProps.put("text", "Country: ");
        labelProps.put("styles", "* {width: 100px; color: gray;}");
        StaticContent[] mixedContent = new StaticContent[] {new LabelContent(labelProps), new
EditorContent(formControlArgs)};
        styles.setProperty(Styles.KEY_MIXED_CONTENT, mixedContent);
      }
    }

    // The previously added form control is the only way the element can be edited.
```

```
    if ("country".equals(authorNode.getName())) {
      styles.setProperty(Styles.KEY_VISIBITY, "-oxy-collapse-text");
    }

    return styles;
  }
}
```

If the execution of the `formControlArgs.put(InplaceEditorArgumentKeys.PROPERTY_VALUES,` `countries);` line consumes too much execution time (for example if it connects to a database or if it needs to extract data from a very large file), you can choose to delay it until the values are actually needed by the form control. This approach is called *lazy evaluation* and can be implemented as follows:

```
formControlArgs.put(InplaceEditorArgumentKeys.PROPERTY_VALUES, new LazyValue<List<CIValue>>() {
  public java.util.List<CIValue> get() {
    // We avoid reading the possible values until they are actually requested.
    // This will be a List with CIValues created over countries: France, Spain, Great Britain
    return readCountriesFromFile();
  }
});
```

The lazy evaluation approach can be used for the following form controls properties:

- `InplaceEditorArgumentKeys.PROPERTY_VALUES`
- `InplaceEditorArgumentKeys.PROPERTY_LABELS`
- `InplaceEditorArgumentKeys.PROPERTY_TOOLTIPS`

The full source code for this example is available inside the *Author SDK*.

Modifying the XML content on Open

Question

I have a bunch of DITA documents which have a fixed path the image `src` attributes. These paths are not valid and I am trying to move away from this practice by converting it in to relative paths. When an XML document is opened, can I trigger the Java API to change the fixed path to a relative path?

Answer

Our Plugins SDK:*http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins* contains a sample Plugin Type called *WorkspaceAccess*.Such a plugin is notified when the application starts and it can do what you want in a couple of ways:

1. You add a listener which notifies you when the user opens an XML document. Then if the XML document is opened in the Author visual editing mode you can use our Author API to change attributes:

```
    pluginWorkspaceAccess.addEditorChangeListener(new  WSEditorChangeListener() {
      /**
       * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorOpened(java.net.URL)
       */
      @Override
      public void editorOpened(URL editorLocation) {
        WSEditor openedEditor =
pluginWorkspaceAccess.getCurrentEditorAccess(StandalonePluginWorkspace.MAIN_EDITING_AREA);
        if(openedEditor.getCurrentPage() instanceof WSAuthorEditorPage) {
        WSAuthorEditorPage authPage = (WSAuthorEditorPage)  openedEditor.getCurrentPage();
        AuthorDocumentController docController = authPage.getDocumentController();
        try {
         //All changes will be undone by pressing Undo once.
         docController.beginCompoundEdit();
         fixupImageRefs(docController,
          docController.getAuthorDocumentNode());
        } finally {
          docController.endCompoundEdit();
          }
        }
      }

      private void fixupImageRefs(AuthorDocumentController docController, AuthorNode authorNode) {
          if(authorNode instanceof AuthorParentNode) {
```

```
                //Recurse
                List<AuthorNode> contentNodes = ((AuthorParentNode)authorNode).getContentNodes();
                if(contentNodes != null) {
                    for (int i = 0; i < contentNodes.size(); i++) {
                        fixupImageRefs(docController, contentNodes.get(i));
                    }
                }
            }
            if(authorNode.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
                AuthorElement elem = (AuthorElement) authorNode;
                if("image".equals(elem.getLocalName())) {
                    if(elem.getAttribute("href") != null) {
                        String originalHref = elem.getAttribute("href").getValue();
                        URL currentLocation = docController.getAuthorDocumentNode().getXMLBaseURL();
                        //TODO here you compute the new href.
                        String newHref = null;
                        docController.setAttribute("href", new AttrValue(newHref), elem);
                    }
                }
            }
        }
    },
StandalonePluginWorkspace.MAIN_EDITING_AREA);
```

2. You also have API to open XML documents in the application:

```
ro.sync.exml.workspace.api.Workspace.open(URL)
```

So you can create up a plugin which automatically opens one by one XML documents from a certain folder in the application, makes modifications to them, saves the content by calling:

```
ro.sync.exml.workspace.api.editor.WSEditorBase.save()
```

and then closes the editor:

```
ro.sync.exml.workspace.api.Workspace.close(URL)
```

Modifying the XML content on Save

Question

Is it possible to get Oxygen to update the revised date on a DITA document when it's saved?

Answer

Our Plugins SDK:*http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins* contains a sample Plugin Type called *WorkspaceAccess*.Such a plugin is notified when the application starts.

You can add a listener which notifies you before the user saves an XML document. Then if the XML document is opened in the Author visual editing mode you can use our Author API to change attributes before the save takes place:

```
    @Override
    public void applicationStarted(final StandalonePluginWorkspace pluginWorkspaceAccess) {
     pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener(){
       //An editor was opened
      @Override
      public void editorOpened(URL editorLocation) {
         final WSEditor editorAccess = pluginWorkspaceAccess.getEditorAccess(editorLocation,
 PluginWorkspace.MAIN_EDITING_AREA);
          if(editorAccess != null){
            editorAccess.addEditorListener(new ro.sync.exml.workspace.api.listeners.WSEditorListener(){
              //Editor is about to be saved
              @Override
              public boolean editorAboutToBeSavedVeto(int operationType) {
                if(EditorPageConstants.PAGE_AUTHOR.equals(editorAccess.getCurrentPageID())){
                  WSAuthorEditorPage authorPage = (WSAuthorEditorPage) editorAccess.getCurrentPage();
                  AuthorDocumentController controller = authorPage.getDocumentController();
                  try {
                    //Find the revised element
                    AuthorNode[] nodes = controller.findNodesByXPath("//revised", true, true, true);
                    if(nodes != null && nodes.length > 0){
                      AuthorElement revised = (AuthorElement) nodes[0];
                      //Set the modified attribute to it...
```

```
                            controller.setAttribute("modified", new AttrValue(new Date().toString()), revised);
                        }
                    } catch (AuthorOperationException e) {
                        e.printStackTrace();
                    }
                }
                //And let the save continue..
                return true;
            }
        });
    }
  }
 }, PluginWorkspace.MAIN_EDITING_AREA);
}
```

Save a new document with a predefined file name pattern

Question

Is it possible using the SDK to get Oxygen Author to automatically generate a file name comprising a UUID plus file extension?

Answer

This could be done implementing a plugin for Oxygen XML Author using our Plugins SDK:

http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins

There is a type of plugin called *Workspace Access* which can be used to add a listener to be notified before an opened editor is saved. The implemented plugin would intercept the save events when a newly created document is untitled and show an alternative chooser dialog to the user, then save the topic with the proper name.

The Java code for this would look like:

```
  private static class CustomEdListener extends WSEditorListener{
    private final WSEditor editor;
    private final StandalonePluginWorkspace
        pluginWorkspaceAccess;
    private boolean saving = false;
    public CustomEdListener(StandalonePluginWorkspace pluginWorkspaceAccess, WSEditor editor) {
            this.pluginWorkspaceAccess = pluginWorkspaceAccess;
            this.editor = editor;
    }
    @Override
    public boolean editorAboutToBeSavedVeto(int operationType) {
      if(! saving &&
        editor.getEditorLocation().toString().contains("Untitled")) {
        File chosenDir = pluginWorkspaceAccess.chooseDirectory();
        if(chosenDir != null) {
          final File chosenFile = new File(chosenDir, UUID.randomUUID().toString() + ".dita");
          SwingUtilities.invokeLater(new Runnable() {
            @Override
              public void run() {
              try {
                saving = true;
                editor.saveAs(new URL(chosenFile.toURI().toASCIIString()));
              } catch (MalformedURLException e) {
                e.printStackTrace();
              } finally {
                saving = false;
              }
            }
          });
        }

        //Reject the original save request.
        return false;
      }
      return true;
    }
  }

    @Override
    public void applicationStarted(final StandalonePluginWorkspace pluginWorkspaceAccess) {
        pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
        @Override
        public void editorOpened(URL editorLocation) {
```

```
        final WSEditor editor = pluginWorkspaceAccess.getEditorAccess(editorLocation,
 PluginWorkspace.MAIN_EDITING_AREA);
        if(editor != null && editor.getEditorLocation().toString().contains("Untitled")) {

        //Untitled editor
          editor.addEditorListener(new CustomEdListener(pluginWorkspaceAccess, editor));
        }
      }
    },
  PluginWorkspace.MAIN_EDITING_AREA);
...................................................
```

Auto-generate an ID when a document is opened or created

Question

Is it possible to configure how the application generates ids? For project compliance we need ids having a certain format for each created topic.

Answer

This could be done implementing a plugin for Oxygen XML Author using our Plugins SDK:

http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins

There is a type of plugin called "Workspace Access" which can be used to add a listener to be notified when an editor is opened.

The implemented plugin would intercept the editor opened and editor page changed events (which occur when a new editor is created) and generate a new ID attribute value on the root element.

The Java code for this would look like:

```
    pluginWorkspaceAccess.addEditorChangeListener(new WSEditorChangeListener() {
    /**
     * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorOpened(java.net.URL)
     */
    @Override
    public void editorOpened(URL editorLocation) {
     WSEditor ed = pluginWorkspaceAccess.getEditorAccess(editorLocation, PluginWorkspace.MAIN_EDITING_AREA);

      generateID(ed);
    }
    /**
     * @see ro.sync.exml.workspace.api.listeners.WSEditorChangeListener#editorPageChanged(java.net.URL)
     */
    @Override
    public void editorPageChanged(URL editorLocation) {
     WSEditor ed = pluginWorkspaceAccess.getEditorAccess(editorLocation, PluginWorkspace.MAIN_EDITING_AREA);

      generateID(ed);
    }

    private void generateID(WSEditor ed) {
      if(ed.getCurrentPage() instanceof WSAuthorEditorPage) {
        WSAuthorEditorPage authorEditPage = (WSAuthorEditorPage) ed.getCurrentPage();
        AuthorDocumentController ctrl = authorEditPage.getDocumentController();
        AuthorElement root = ctrl.getAuthorDocumentNode().getRootElement();
        if(root.getAttribute("id") == null || !root.getAttribute("id").getValue().startsWith("generated_"))
{
          ctrl.setAttribute("id", new AttrValue("generated_" + Math.random()), root);
        }
      }
    }

  }, PluginWorkspace.MAIN_EDITING_AREA);
```

Use a custom view with the Oxygen XML Author distribution

Question

Is it possible to create a custom view in Eclipse which can insert certain XML fragments in the documents opened with the Oxygen XML Author?

Answer

Here you can find more information about the Eclipse part of the oXygen SDK:

http://www.oxygenxml.com/oxygen_sdk.html#oXygen_Eclipse_plugin

Use the provided Oxygen XML Author sample project as a starting point. From any custom view/component you can have singleton access to the using the
`ro.sync.exml.workspace.api.PluginWorkspaceProvider.getPluginWorkspace()` API.

The Java code for inserting a certain XML fragment in the currently open editor (either in the Text or Author editing modes) would look like this:

```
    WSEditor currentEditorAccess =
 PluginWorkspaceProvider.getPluginWorkspace().getCurrentEditorAccess(PluginWorkspace.MAIN_EDITING_AREA);
    if(currentEditorAccess.getCurrentPage() instanceof WSXMLTextEditorPage) {
      //Editor opened in Text page
      WSXMLTextEditorPage tp = (WSXMLTextEditorPage) currentEditorAccess.getCurrentPage();
      //You can access an API to insert text in the XML content
//      tp.getDocument().insertString(tp.getCaretOffset(), "<testTag/>", null);
      //This is the internal StyledText implementation
//      tp.getTextComponent()
      //You can use this XPath API to find the range of an XML element.
//      tp.findElementsByXPath(xpathExpression)
    } else if(currentEditorAccess.getCurrentPage() instanceof WSAuthorEditorPage) {
      //Editor opened in Author page
//      try {
      WSAuthorEditorPage authPage = (WSAuthorEditorPage) currentEditorAccess.getCurrentPage();
      //Then you can do stuff like this to insert XML at caret position
//        authPage.getDocumentController().insertXMLFragment("<testTag/>", authPage.getCaretOffset());
//      } catch (AuthorOperationException e) {
//        // TODO Auto-generated catch block
//        e.printStackTrace();
//      }
    }
```

Chapter

11

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, do one of the following:

- use an XML-compliant user agent;
- transform the XML document to a format that can be read by other user agents. This process is known as **Transformation**.

Output Formats

Within the current version of Oxygen XML Author 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 Author 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 Author supports XSLT/XPath version 1.0 using Saxon 6.5.5, Xalan, Xsltproc, MSXML (3.0, 4.0, .NET) and XSLT/XPath 2.0 by using Saxon 9.5.1.7 HE, Saxon 9.5.1.7 PE, and Saxon 9.5.1.7 EE.

Transformation Scenario

A transformation scenario is a set of complex operations and settings that gives you the possibility to obtain outputs of multiple types (XML, HTML, PDF, EPUB, and others) from the same source of XML files and stylesheets.

Executing a transformation scenario implies multiple actions, such as:

- validating the input file
- obtaining intermediate output files (for example formatting objects for the XML to PDF transformation)
- using transformation engines to produce the output

Before transforming an XML document in Oxygen XML Author, define a transformation scenario to apply to that document. A scenario is a set of values for various parameters defining a transformation. It is not related to any particular document but to a document type:

- **Scenarios that apply to XML files** - Such a scenario contains the location of an XSLT stylesheet that is applied on the edited XML document and other transform parameters.
- **Scenarios that apply to XSLT files** - Such a scenario contains the location of an XML document that the edited XSLT stylesheet is applied on and other transform parameters.
- **Scenarios that apply to XQuery files** - Such a scenario contains the location of an XML source that the edited XQuery file is applied on and other transform parameters. When the XML source is a native XML database the XML source field of the scenario is empty because the XML data is read with XQuery specific functions like document(). When the XML source is a local XML file, the URL of the file is specified in the XML input field of the scenario.
- **Scenarios that apply to SQL files** - Such a scenario specifies a database connection for the database server that runs the SQL file associated with the scenario. The data processed by the SQL script is located in the database.
- **Scenarios that apply to XProc files** - Such a scenario contains the location of an XProc script and other transform parameters.
- **DITA-OT scenarios** - Such a scenario provides the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Author comes with a built-in version of Ant and a built-in version of DITA-OT but different versions can be set in the scenario.

  Note:

  Status messages generated during the transformation process are displayed in the *Information view*.

Defining a New Transformation Scenario

Defining a transformation scenario is the first step in the process of transforming a document. The following types of scenarios are available:

- *XML transformation with XSLT* - Specifies transform parameters and the location of an XSLT stylesheet that Oxygen XML Author applies to the edited XML document. This scenario is useful when you develop an XML document and the XSLT document is in its final form.
- *XML transformation with XQuery* - Specifies transform parameters and the location of an XQuery file that Oxygen XML Author applies to the edited XML document.
- *DITA-OT transformation* - Specifies the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Author comes with a built-in version of Ant and a built-in version of DITA-OT but different versions can be set in the scenario.
- *ANT transformation* - Allows you to configure options and parameters of an ANT script.
- *XSLT transformation* - Specifies transform parameters and the location of an XML document to which the edited XSLT stylesheet is applied. This scenario is useful when you develop an XSLT document and the XML document is in its final form.

- *XProc transformation* - Contains the location of an XProc script and other transform parameters.
- *XQuery transformation* -Specifies transform parameters and the location of an XML source to which the edited XQuery file is applied. When the XML source is a native XML database the XML source field of the scenario is empty because the XML data is read with XQuery specific functions like `document()`. When the XML source is a local XML file, the URL of the file is specified in the XML input field of the scenario.
- *SQL transformation* - Specifies a database connection for the database server that runs the SQL file associated with the scenario. The data processed by the SQL script is located in the database.

XML transformation with XSLT

To create an **XML transformation with XSLT** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XML transformation with XSLT**;

- Click the **Configure Transformation Scenario(s)** (**Ctrl+Shift+C (Command+Shift+C on OS X)**) button on the **Transformation** toolbar, then click the **New** button and select **XML transformation with XSLT**;

- Select **Ctrl+Shift+T (Command+Shift+T on OS X)** on your keyboard or click the **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XML transformation with XSLT**.

    Note: In case a scenario is already associated with the edited document, selecting **Ctrl+Shift+T (Command+Shift+T on OS X)** or **Apply Transformation Scenario** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- *XSLT*;
- *Output*;
- *FO Processors*.

The XSLT Tab

The **XSLT** tab contains the following options:

- **XML URL** - specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

    Note: In case the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

    Note: In case the transformer engine is one of the built-in XSLT 2.0 / 3.0 engines and *the name of an initial template* is specified in the scenario, the **XML URL** field can be empty. The **XML URL** field can also be empty in case of *external XSLT processors*. In all other cases a non-empty XML URL value is mandatory.

- **XSL URL** - specifies the source XSL file that the transformation uses. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XSL URL** fields:

Insert Editor Variables

Opens a pop-up menu allowing to introduce special *Oxygen XML Author editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing to select a local file.

Browse for remote file

Opens an URL browser dialog box allowing to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing to select a file from a zip archive.

Browse Data Source Explorer

Opens the *Data Source Explorer* window.

Search for file

Allows you to find a file in the current project.

Open in editor

Opens in an editor panel the file with the path specified in the **XML URL** text box.

The rest of the options available in the **XSLT** tab allow you to further customize the transformation scenario:

- **Use "xml-stylesheet" declaration** - use the stylesheet declared with an `xml-stylesheet` declaration instead of the stylesheet specified in the **XSL URL** field. By default this checkbox is not selected and the transformation applies the XSLT stylesheet specified in the **XSL URL** field. If it is checked, the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction

- **Transformer** - this combo box presents all the transformation engines available to Oxygen XML Author for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog is used as the default transformation engine. In case no validation scenario is associated with an XSLT or XQuery document, the transformation engine is used in the validation process, if it provides validation support;

    - **Advanced options** - allows you to configure advanced options of the Saxon HE / PE / EE engine for the current transformation scenario. To configure the same options globally, go to the *Saxon-HE/PE/EE preferences page*. For the current transformation scenario, these **advanced options** override the options configured in the *Saxon-HE/PE/EE preferences page*. The **Initial mode and template** option is available only in the **advanced options**. It is a Saxon-specific option that sets the name of the first XSLT template which starts the XSLT transformation or the initial mode of transformation.

- **Parameters** - opens *the Configure parameters dialog*, allowing you to configure the XSLT parameters used in the current transformation. In this dialog you can also configure the parameters of additional stylesheets, set with the **Additional XSLT stylesheets** button. If the XSLT transformation engine is custom-defined you can not use this dialog to configure the parameters sent to the custom engine. In this case, you can copy all parameters from the dialog using the contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line

- **Extensions** - opens *the dialog for configuring the XSLT/XQuery extension jars or classes* which define extension Java functions or extension XSLT elements used in the transformation

- **Additional XSLT stylesheets** - opens *the dialog for adding XSLT stylesheets* which are applied on the result of the main stylesheet specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document

The Output Tab

The **Output** tab contains the following options:

- **Prompt for file** - At the end of the transformation a file browser dialog is displayed for specifying the path and name of the file which stores the transformation result.

- **Save As** - The path of the file where the transformation result are stored. The path can include *special Oxygen XML Author editor variables* or *custom editor variables*.
- **Open in Browser/System Application** - If enabled, Oxygen XML Author opens the transformation result automatically, in a system application associated with the type of the result (HTML/XHTML, PDF, text) file.

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

- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Author 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 Author 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 Author editor variables* or *custom editor variable*.
- **Open in editor** - When this is enabled, the transformation result set in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type: if the result is an XML file it is opened with the built-in XML editor, if it is an XSL-FO file it is opened with the built-in FO editor, and so on.
- **Show in results view as XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked, Oxygen XML Author displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

> ⚠ **Important:** When transforming very large documents, you should be aware that enabling this feature results in a very long time necessary for rendering the transformation result in the XHTML result viewer panel. This drawback appears due to the built-in Java XHTML browser implementation. In these cases, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

- **Show in results view as XML** - If this is checked Oxygen XML Author displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlight* specific for XML documents.
- **Show in results view as SVG** - If this is checked Oxygen XML Author displays the transformation result in an SVG viewer panel at the bottom of the application window by rendering the result as an SVG image.
- **Image URLs are relative to** - If **Show As XHTML** is checked this text field specifies the path used to resolve image paths contained in the transformation result.

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation;
- **XSLT result as input** - the FO processor is applied to the result of the XSLT transformation defined in the **XSLT** tab;
- **XML URL as input** - the FO processor is applied to the input XML file;
- **Method** - the output format of the FO processing. Available options depend on the selected processor type;
- **Processor** - specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

XSLT Stylesheet Parameters

The global parameters of the XSLT stylesheet used in the transformation scenario are configured from the dialog available from the **Parameters** button of the **Configure Transformation** dialog:

The table presents all the parameters of the XSLT stylesheet, all imported and included stylesheets and all *additional stylesheets* with their current values. Use the **Filter** text box to search for a specific term in the entire parameters collection. The following font type and color conventions are used:

- blue font values are the defaults collected from the stylesheet;
- black font values and bold font names indicate edited parameters.

If a parameter value was not edited, then the table presents its default value. The bottom panel presents:

- the default value of the parameter selected in the table
- a description of the parameter, if available
- the system ID of the stylesheet that declares it

> For example setting the value of a parameter having a declared namespace like:
>
> ```
> <xsl:param name="p:param" xmlns:p="namespace">default</xsl:param>
> ```
>
> use the following expression in the **Name** column of the **Parameters** dialog:
>
> ```
> {namespace}param
> ```

If the **XPath** column is checked, the parameter value is evaluated as an XPath expression before starting the XSLT transformation.

> For example, you can use expressions like:
>
> ```
> doc('test.xml')//entry
> //person[@atr='val']
> ```

Note:

1. The **doc** function solves the argument relative to the XSL stylesheet location. You can use full paths or editor variables like *${cfdu}* (current file directory) to specify other locations: `doc('${cfdu}/test.xml')//*`
2. You cannot use XSLT Functions. Only XPath functions are allowed.

The following actions are available for managing parameters:

New

Adds a new parameter to the list.

Edit

Edits the value of the selected parameter.

Unset

Resets the selected parameter to its default value. Available only for parameters with set values.

Delete

Removes the selected parameter from the list. It is enabled only for parameters added to the list with the **New** button.

The *editor variables* displayed at the bottom of the dialog can be used in the values of the parameters to make them independent of the location of the XSLT stylesheet or the XML document.

Additional XSLT Stylesheets

The list of additional XSLT stylesheets can be edited in the dialog opened by the button **Additional XSLT Stylesheets** from the **Configure Transformation** dialog. The following actions are available:

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.

Open

Opens the selected stylesheet in a separate view.

Up

Moves the selected stylesheet up in the list.

Down

Moves the selected stylesheet down in the list.

This dialog allows the user to add additional XSLT stylesheets to the transformation.

The path specified in the URL text field can include *special Oxygen XML Author editor variables.*

XML Transformation with XQuery

To create an **XML transformation with XQuery** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XML transformation with XQuery**;
- Click the **Configure Transformation Scenario(s)** (**Ctrl+Shift+C (Command+Shift+C on OS X)**) button on the **Transformation** toolbar, then click the **New** button and select **XML transformation with XQuery**;
- Select **Ctrl+Shift+T (Command+Shift+T on OS X)** on your keyboard or click the **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XML transformation with XQuery**.

  > **Note:** In case a scenario is already associated with the edited document, selecting **Ctrl+Shift+T (Command+Shift+T on OS X)** or **Apply Transformation Scenario** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- *The XQuery tab;*
- *The FO Processor tab;*
- *The Output tab.*

The XQuery Tab

The **XQuery** tab contains the following options:

- **XML URL** - specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  > **Note:** In case the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

- **XQuery URL** - specifies the source XQuery file that the transformation uses. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XQuery URL** fields:

 Insert Editor Variables

Opens a pop-up menu allowing to introduce special *Oxygen XML Author editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing to select a local file.

Browse for remote file

Opens an URL browser dialog box allowing to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing to select a file from a zip archive.

Browse Data Source Explorer

Opens the *Data Source Explorer* window.

Search for file

Allows you to find a file in the current project.

Open in editor

Opens in an editor panel the file with the path specified in the **XML URL** text box.

The rest of the options available in the **XQuery** tab allow you to further customize the transformation scenario:

- **Transformer** - this combo box presents all the transformation engines available to Oxygen XML Author for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog is used as the default transformation engine. In case no validation scenario is associated with an XSLT or XQuery document, the transformation engine is used in the validation process, if it provides validation support;

    - **Advanced options** - *configure advanced options specific for the Saxon HE / PE / EE engine*.

- **Parameters** - opens the **Configure parameters** dialog for configuring the XQuery parameters. If the XQuery/XSLT transformation engine is custom-defined you can not use this dialog to set parameters. Instead, copy all parameters from the dialog using the contextual menu actions and edit the custom XSLT/XQuery engine to include the necessary parameters in the command line which starts the transformation process;

    Note: Use the **Filter** text box to search for a specific term in the entire parameters collection.

- **Extensions** - opens *the dialog for configuring the XSLT/XQuery extension jars or classes* which define extension Java functions or extension XQuery elements used in the transformation;

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation;
- **XQuery result as input** - the FO processor is applied to the result of the XQuery transformation defined in the **XQuery** tab;
- **XML URL as input** - the FO processor is applied to the input XML file;
- **Method** - the output format of the FO processing. Available options depend on the selected processor type;
- **Processor** - specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Present as a sequence** - enable this option to avoid the long time necessary for fetching the full result. This option fetches only the first chunk of the result;
- **Prompt for file** - At the end of the transformation a file browser dialog is displayed for specifying the path and name of the file which stores the transformation result;
- **Save As** - The path of the file where the transformation result are stored. The path can include *special Oxygen XML Author editor variables* or *custom editor variables*;

- **Open in Browser/System Application** - If enabled, Oxygen XML Author opens the transformation result automatically, in a system application associated with the type of the result (HTML/XHTML, PDF, text) file.

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

- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Author 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 Author 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 Author editor variables* or *custom editor variable*
- **Open in editor** - When this is enabled, the transformation result set in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type: if the result is an XML file it is opened with the built-in XML editor, if it is an XSL-FO file it is opened with the built-in FO editor, and so on;
- **Show As XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked, Oxygen XML Author displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

    ⚠ **Important:** When transforming very large documents, you should be aware that enabling this feature results in a very long time necessary for rendering the transformation result in the XHTML result viewer panel. This drawback appears due to the built-in Java XHTML browser implementation. In these cases, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

- **Show in results view as XML** - If this is checked Oxygen XML Author displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlight* specific for XML documents
- **Show in results view as SVG** - If this is checked Oxygen XML Author displays the transformation result in an SVG viewer panel at the bottom of the application window by rendering the result as an SVG image;
- **Image URLs are relative to** - If **Show As XHTML** is checked this text field specifies the path used to resolve image paths contained in the transformation result.

DITA OT Transformation

To create a **DITA OT Transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select 🔧 **Transformation Scenarios** to display this view. Click the **New** button and select **DITA OT Transformation**;
- Click the 🔧 **Configure Transformation Scenario(s)**(**Ctrl+Shift+T (Command+Shift+T on OS X)**) button on the **Transformation** toolbar, then click the **New** button and select **DITA OT Transformation**;
- Select **Ctrl+Shift+T (Command+Shift+T on OS X)** on your keyboard or click the ▶ **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **DITA OT Transformation**.

    📋 **Note:** In case a scenario is already associated with the edited document, selecting **Ctrl+Shift+T (Command+Shift+T on OS X)** or ▶ **Apply Transformation Scenario** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the ▶ **Apply Transformation Scenario** button.

All three methods open the **DITA transformation type** dialog box. This dialog presents the list of possible outputs that the **DITA OT Transformation** is able to produce. Select the transformation type, click **OK** and move on to configuring the options in the **New Scenario** dialog. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;

- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- *Parameters*;
- *Filters*;
- *Advanced*;
- *Output*;
- *FO Processor*.

> **Note:** To display the console during the transformation process, click 🗏 **Show console output** in the status bar.

The Skins Tab

A *skin* is a collection of CSS properties that can alter the look of the output by changing colors, font types, borders, margins and paddings. This allows you to rapidly adapt the output look and feel to your organization's standard rules.

Oxygen XML Author provides a set of predefined *skins* for the **DITA Map WebHelp** and **DITA Map WebHelp with Feedback** transformation scenarios.
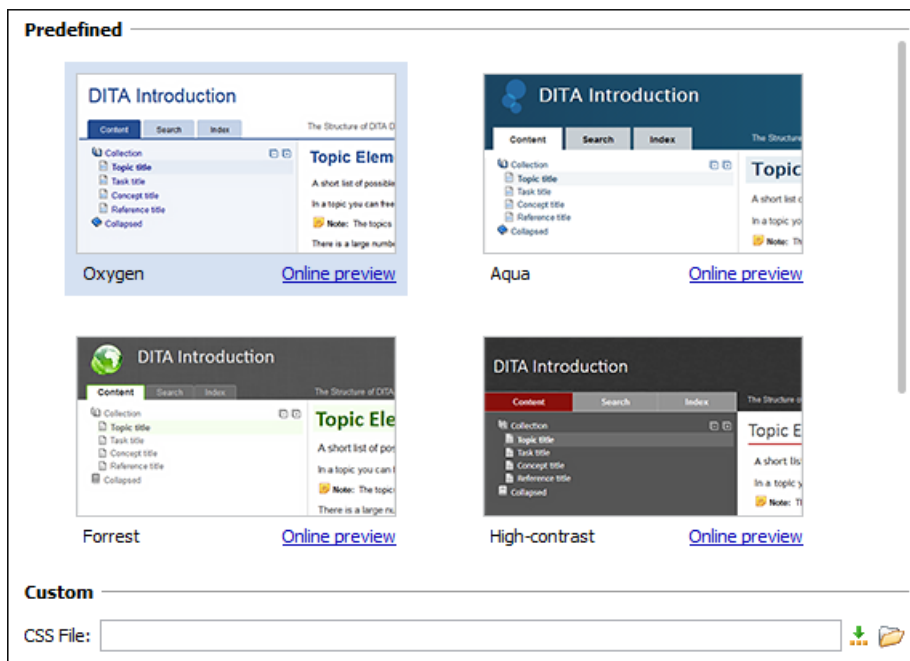
Figure 166: The Skins Tab

The predefined skins cover a wide range of chromatic themes, ranging from a very light one to a high-contrast variant. By default, the **Oxygen** skin is selected (note the light blue border around the skin's preview). If you want to obtain an output without any customization, deselect the currently selected skin.

To see how the *skin* looks like applied on a documentation sample project stored on the Oxygen XML Author website, press the **Online preview** link.

> **Note:** Press the **Create custom skin** link to open the *WebHelp Skin Builder* tool.

To further customize the look of the output, set the **CSS File** field to point to your custom CSS stylesheet or to a customized skin.

> **Note:** A custom CSS file will overwrite a skin selection.

Note: The output can also be styled by setting the `args.css` parameter in the **Parameters tab**. The properties taken from the stylesheet referred in this parameter take precedence over the properties declared in the skin set in the **Skins tab**.

The Parameters Tab

This dialog allows you to configure the parameters sent to the DITA-OT build file.

All the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (for example: 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. Use the **Filter** text box to search for a specific term in the entire parameters collection.

Using the toolbar buttons you can add, edit or remove a parameter.

Depending on the type of a parameter, its value can be one of the following:

- a simple text field for simple parameter values
- a combo box with some predefined values
- a file chooser and an editor variables selector to simplify setting a file path as value to a parameter.

The value of a parameter can be entered at runtime if a value *ask('user-message', param-type, 'default-value' ?)* is used as value of parameter in the Configure parameters dialog.

Examples:

- `${ask('message')}` - Only the message displayed for the user is specified.
- `${ask('message', generic, 'default')}` - `'message'` is displayed, the type is not specified (the default is string), the default value is `'default'`.
- `${ask('message', password)}` - `'message'` is displayed, the characters typed are masked with a circle symbol.
- `${ask('message', password, 'default')}` - same as before, the default value is `'default'`.
- `${ask('message', url)}` - `'message'` is displayed, the parameter type is URL.
- `${ask('message', url, 'default')}` - same as before, the default value is `'default'`.

The *Filters* Tab

In the scenario **Filters** tab you can add filters to remove certain content elements from the generated output.

Figure 167: Edit Filters tab

There are three ways to define filters:

- **Use DITAVAL file** - If you already have a DITAVAL file associated with the transformed map you can specify the path to it and it will be used when filtering content. You can find out more about constructing a DITAVAL file in the *DITA OT Documentation* topic.
- **Use profiling condition set** - Sets the *profiling condition set* that applies to the document you transform.
- **Exclude from output all elements with any of the following attributes** - You can configure a simple list of attribute (name, value) pairs which when present on an element in the input will remove it from output.

The *Advanced* Tab

In the **Advanced** tab, you can specify advanced options for the transformation.

Figure 168: 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* parameter 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 path configured in the *Ant preferences page*. A custom value can also be set;
- **Java Home** - You can specify a custom Java Virtual Machine to run the ANT transformation. By default it is the Java Virtual Machine used by .
- **JVM Arguments** - This parameter allows you to set specific parameters to the Java Virtual Machine used by ANT. By default it is set to -Xmx384m which means the transformation process is allowed to use 384 megabytes of memory. Sometimes, when performing a large DITA map transformation you may want to increase the memory allocated to the Java Virtual Machine from the default value (384 MB) to a higher value, like 512 MB. This way, you can avoid the Out of Memory error messages (**OutOfMemoryError**) received from the ANT process.

    Note: If you are publishing DITA to PDF and still experience problems, you should also increase the amount of memory allocated to the FO transformer. To do this, go to the **Advanced** tab and increase the value of the **Java Arguments** parameter.

- **Libraries** - adds by default as high priority libraries which are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can specify all the additional libraries (jar files or additional class paths) which are used by the ANT transformer. You can also decide to control all libraries added to the classpath.

The Output Tab

In the **Output** tab, you can configure options related to the place where the output is generated.
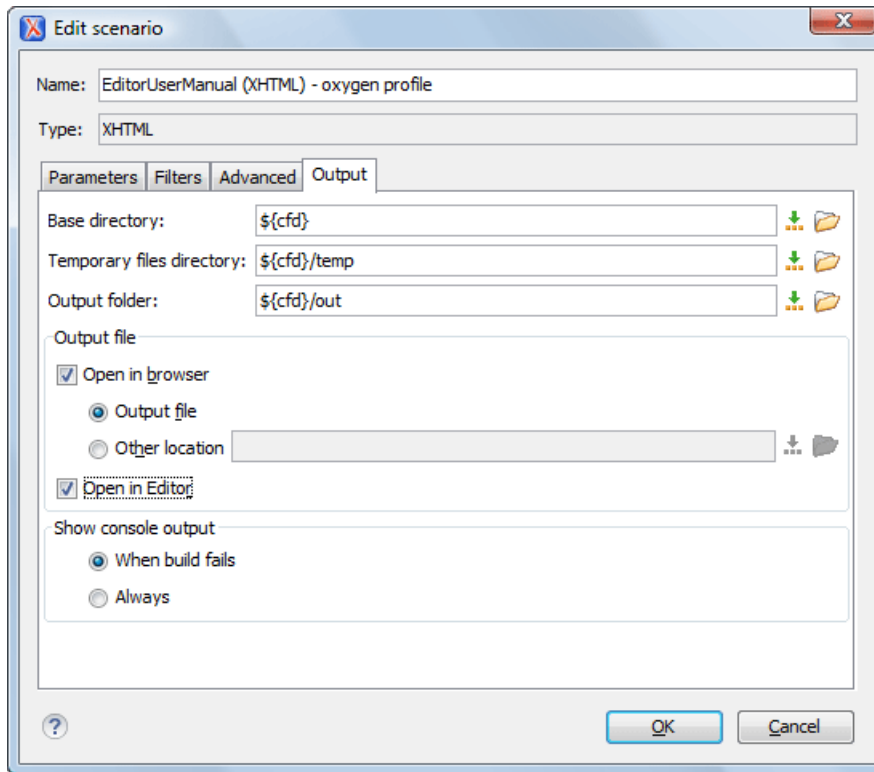
Figure 169: Output settings tab

You have several parameters that you can specify here:

- **Base directory** - All the relative paths which appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located.
- **Temporary files directory** - This directory is used to store pre-processed temporary files until the final output is obtained.
- **Output folder** - The folder where the final output content is copied.
- **Output file options** - The transformation output can then be opened in a browser or even in the editor, if specified.
- **Show console output** - Specifies whether the console is always displayed or only when the build fails.

> **Note:** If the DITA Map or topic is opened from a remote location or from a ZIP file, the scenario must specify absolute output, temporary and base file paths.

The *FO Processor* Tab

This tab allows you to set an FO Processor, when you choose to generate PDF output.

Figure 170: FO Processor configuration tab

You can choose between:

- **Apache FOP** - Default setting. This processor comes bundled with .
- **XEP** - The *RenderX* XEP processor.

  If you select **XEP** in the combo and XEP was already installed in you can see the detected installation path appear under the combo box.

  XEP is considered as installed if it was detected in one of the following sources:

  - XEP was configured as an external FO Processor in the *FO Processors option page*;
  - The system property *com.oxygenxml.xep.location* was set to point to the XEP executable file for the platform (for example: `xep.bat` on Windows);
  - XEP was installed in the `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/lib` directory of the installation directory.

- **Antenna House** - The *Antenna House* AH (v5) or XSL (v4) Formatter processor.

  If Antenna House was already installed on your computer and you select **Antenna House** in the combo box, in you can see the detected installation path appear under the combo.

  Antenna House is considered as installed if it was detected in one of the following sources:

  - Environment variable set by Antenna House installation (the newest installation version will be used, *v5* being preferred over *v4*).
  - Antenna House was added as an external FO Processor in the preferences pages.

  To further customize the PDF output obtained from the Antenna House processor:

  - edit the transformation scenario
  - open the *Parameters tab*
  - add the `env.AXF_OPT` parameter and point to Antenna House configuration file

ANT Transformation

An **ANT** transformation scenario is usually associated with an Ant build script. Oxygen XML Author runs an **ANT** transformation scenario as an external process that executes the Ant build script with the built-in Ant distribution (Apache Ant version 1.8.2) that comes with the application or optionally with a custom Ant distribution configured in the scenario.

To create an **ANT** transformation scenario, use one of the following methods:

*   Go to **Window** > **Show View**, select **Transformation Scenarios**, click **New**, and select **ANT**;
*   Click the **Configure Transformation Scenario(s)** (**Ctrl+Shift+C (Command+Shift+C on OS X)**) button on the **Transformation** toolbar, then click the **New** button and select **ANT**;
*   Select **Ctrl+Shift+T (Command+Shift+T on OS X)** on your keyboard or click the **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **ANT**.

    > **Note:** In case a scenario is already associated with the edited document, selecting **Ctrl+Shift+T (Command+Shift+T on OS X)** or **Apply Transformation Scenario** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and the **Storage** options:

*   **Global Options** - the scenario is saved in the global options stored in the user home directory;
*   **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

*   *the **Options** tab*;
*   *the **Parameters** tab*;
*   *the **Output** tab*.

    > **Note:** To display the console during the transformation process, click **Show console output** in the status bar.

The Options Tab

The **Options** tab contains the following options:

*   **Working directory** - path of the current directory of the Ant external process. An *editor variable* can be inserted in this text box using the small green arrow button ();
*   **Build file** - ant script file that is the input of the Ant external process. An *editor variable* can be inserted in this text box using the small green arrow button ();
*   **Build target** - optionally a build target from the Ant script file can be specified. If no target is specified, the Ant target that is specified as default in the Ant script file is executed;
*   **Additional arguments** - additional command-line arguments to be passed to the Ant transformation (for example -verbose);
*   **Ant Home** - path to the Ant installation to run the transformation. By default it is the path configured in the *Ant preferences page*. A custom value 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 Author. 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. 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 to a higher one, like 512 MB. This way, you can avoid running out of memory (**OutOfMemoryError**) when running an Ant process;

- **Libraries** - this button allows adding to the classpath of the Ant process any external libraries that are not bundled with Ant (that is they are not built-in Ant libraries).

The Parameters Tab

In the **Parameters** tab you can use the **New**, **Edit**, and **Delete** buttons to set the parameters accessible as Ant properties in the Ant build script. Use the **Filter** text box to search for a specific term in the entire parameters collection.

The Output Tab

The following details can be configured in the **Output** tab:

- the file to open automatically when the transformation is finished in the **Open** text box, usually the output file of the Ant process; an *editor variable* can be inserted in this text box using the small green arrow button (⬇);
- if the file specified in the **Open** text box is opened in the system application that is set in the operating system as the default application for that type of files (for example the *Acrobat Reader* application for .pdf files);
- if the file specified in the **Open** text box is opened in Oxygen XML Author; for example if it is an .xml file it is opened automatically in *the XML editor panel*, if it is a .zip file or an .epub file it is opened in *the Archive Browser view*, etc.;

XSLT Transformation

To create an **XSLT transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select 🔧 **Transformation Scenarios** to display this view. Click the **New** button and select **XSLT transformation**;
- Click the 🔧 **Configure Transformation Scenario(s)** (**Ctrl+Shift+C (Command+Shift+C on OS X)**) button on the **Transformation** toolbar, then click the **New** button and select **XSLT transformation**;
- Select **Ctrl+Shift+T (Command+Shift+T on OS X)** on your keyboard or click the ▶ **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XSLT transformation**.

  > 📄 **Note:** In case a scenario is already associated with the edited document, selecting **Ctrl+Shift+T (Command+Shift+T on OS X)** or ▶ **Apply Transformation Scenario** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the ▶ **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- *XSLT*;
- *FO Processor*;
- *Output*;

The XSLT Tab

The **XSLT** tab contains the following options:

- **XML URL** - specifies the source XML file that the transformation uses. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  > 📄 **Note:** In case the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

> **Note:** In case the transformer engine is one of the built-in XSLT 2.0 / 3.0 engines and *the name of an initial template* is specified in the scenario, the **XML URL** field can be empty. The **XML URL** field can also be empty in case of *external XSLT processors*. In all other cases a non-empty XML URL value is mandatory.

- **XSL URL** - specifies the source XSL. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XSL URL** fields:

Insert Editor Variables

Opens a pop-up menu allowing to introduce special *Oxygen XML Author editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing to select a local file.

Browse for remote file

Opens an URL browser dialog box allowing to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing to select a file from a zip archive.

Browse Data Source Explorer

Opens the *Data Source Explorer* window.

Search for file

Allows you to find a file in the current project.

Open in editor

Opens in an editor panel the file with the path specified in the **XML URL** text box.

The rest of the options available in the **XSLT** tab allow you to further customize the transformation scenario:

- **Use "xml-stylesheet" declaration** - use the stylesheet declared with an `xml-stylesheet` declaration instead of the stylesheet specified in the **XSL URL** field. By default this checkbox is not selected and the transformation uses the XSLT stylesheet specified in the **XSL URL** field. If it is checked, the scenario is applied to the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction;

- **Transformer** - this combo box presents all the transformation engines available to Oxygen XML Author for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog is used as the default transformation engine. In case no validation scenario is associated with an XSLT or XQuery document, the transformation engine is used in the validation process, if it provides validation support;

- **Advanced options** - allows you to configure advanced options of the Saxon HE / PE / EE engine for the current transformation scenario. To configure the same options globally, go to the *Saxon-HE/PE/EE preferences page*. For the current transformation scenario, these **advanced options** override the options configured in the *Saxon-HE/PE/EE preferences page*. The **Initial mode and template** option is available only in the **advanced options**. It is a Saxon-specific option that sets the name of the first XSLT template which starts the XSLT transformation or the initial mode of transformation;

- **Parameters** - opens *the Configure parameters dialog*, allowing you to configure the XSLT parameters used in the current transformation. In this dialog you can also configure the parameters of additional stylesheets, set with the **Additional XSLT stylesheets** button. If the XSLT transformation engine is custom-defined you can not use this dialog to configure the parameters sent to the custom engine. In this case, you can copy all parameters from the dialog using the contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line ;

  > **Note:** Use the **Filter** text box to search for a specific term in the entire parameters collection.

- **Extensions** - opens *the dialog for configuring the XSLT/XQuery extension jars or classes* which define extension Java functions or extension XSLT elements used in the transformation;

- **Additional XSLT stylesheets** - opens *the dialog for adding XSLT stylesheets* which are applied on the result of the main stylesheet specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document;

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation;
- **XSLT result as input** - the FO processor is applied to the result of the XSLT transformation defined in the **XSLT** tab;
- **XML URL as input** - the FO processor is applied to the input XML file;
- **Method** - the output format of the FO processing. Available options depend on the selected processor type;
- **Processor** - specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Prompt for file** - At the end of the transformation a file browser dialog is displayed for specifying the path and name of the file which stores the transformation result;
- **Save As** - The path of the file where the transformation result are stored. The path can include *special Oxygen XML Author editor variables* or *custom editor variables*;
- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Author 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 Author 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 Author editor variables* or *custom editor variable*;
- **Open in editor** - When this is enabled, the transformation result set in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type: if the result is an XML file it is opened with the built-in XML editor, if it is an XSL-FO file it is opened with the built-in FO editor, and so on;
- **Show in results view as XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked, Oxygen XML Author displays the transformation result in a built-in XHTML browser panel at the bottom of the application window;

  > **Important:** When transforming very large documents, you should be aware that enabling this feature results in a very long time necessary for rendering the transformation result in the XHTML result viewer panel. This drawback appears due to the built-in Java XHTML browser implementation. In these cases, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

- **Show in results view as XML** - If this is checked Oxygen XML Author displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlight* specific for XML documents;
- **Show in results view as SVG** - If this is checked Oxygen XML Author displays the transformation result in an SVG viewer panel at the bottom of the application window by rendering the result as an SVG image;
- **Image URLs are relative to** - If **Show As XHTML** is checked this text field specifies the path used to resolve image paths contained in the transformation result.

XProc Transformation

A sequence of transformations described by an XProc script can be executed with an XProc transformation scenario. To create an **XProc transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XProc transformation**;

- Click the ⚙ **Configure Transformation Scenario(s)** (**Ctrl+Shift+C (Command+Shift+C on OS X)**) button on the **Transformation** toolbar, then click the **New** button and select **XProc transformation**;

- Select **Ctrl+Shift+T (Command+Shift+T on OS X)** on your keyboard or click the ▶ **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XProc transformation**.

  > 📄 **Note:** In case a scenario is already associated with the edited document, selecting **Ctrl+Shift+T (Command+Shift+T on OS X)** or ▶ **Apply Transformation Scenario** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the ▶ **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and the **Storage** options:

- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

- *the XProc tab*;
- *the Inputs tab*;
- *the Parameters tab*;
- *the Outputs tab*;
- *the Options tab*.

The XProc Tab

The **XProc** tab contains the following options:

- **XProc URL** - specifies the URL of the XProc script;
- **Processor** - specifies the XProc engine. You can select the built-in *Calabash* engine or a custom engine *configured in the Preferences dialog*.

The Inputs Tab

The **Inputs** tab contains a list with the ports that the XProc script uses to read input data. To add, modify, and delete ports from this list, use the **New**, **Edit**, and **Delete** buttons.

> 📄 **Note:** Use the **Filter** text box to search for a specific term in the entire ports collection.

Each input port has an assigned name in the XProc script. The XProc engine reads data from the URLs specified in the **URLs** list. The *built-in editor variables* and the *custom editor variables* can be used to specify these URLs.

The Parameters Tab

The **Parameters** tab presents a list with the parameters collected from the XProc script. To add new parameters click the **New** button.

> 📄 **Note:** Use the **Filter** text box to search for a specific term in the entire parameters collection.

Each port where the output of the XProc transformation is sent is associated with an URL on the **Outputs** tab of the dialog. The *built-in editor variables* and the *custom editor variables* can be used for specifying this URL.

The Outputs Tab

The **Outputs** tab displays a list of output ports collected from the XProc script. To define additional output ports click the **New** button .

The result of the XProc transformation can be displayed as a sequence in an output view with two sections:

- a list with the output ports on the left side;
- the content of the document(s) that correspond to the selected output port on the right side.

If the **Open results in editor** option is selected, the XProc transformation result is opened automatically in an editor panel. By selecting the **Open in System Application** option, you can specify a file that is opened at the end of the XProc transformation in the system application associated with that file type.
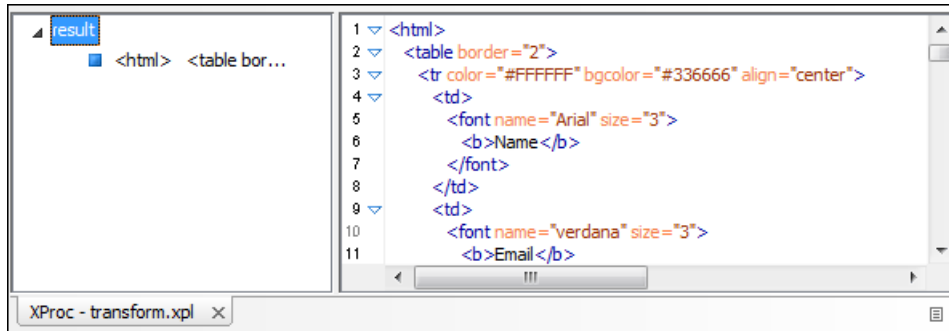
Figure 171: XProc Transformation results view

The Options Tab

The **Options** displays a list with the options collected from the XProc script. To define new options click the **New** button.

📄   **Note:** Use the **Filter** text box to search for a specific term in the entire options collection.

The options collected from the script are rendered plain. Edited options as well as new ones are rendered bold.

Configuring Calabash with XEP

To generate PDF output from your XProc pipeline (when using the Calabash XProc processor), follow these steps:

1. Open the `[OXYGEN_DIR]/lib/xproc/calabash/engine.xml` file.
2. Uncomment the `<system-property name="com.xmlcalabash.fo-processor" value="com.xmlcalabash.util.FoXEP"/>` system property.
3. Uncomment the `<system-property name="com.renderx.xep.CONFIG" file="../../../tools/xep/xep.xml"/>` system property. Edit the `file` attribute to point to the configuration file that is usually located in the XEP installation folder.
4. Uncomment the references to the XEP libraries. Edit them to point to the matching library names from the XEP installation directory.
5. Restart Oxygen XML Author.

Integration of an External XProc Engine

The Javadoc documentation of the XProc API is available for download from the application website as a zip file: *xprocAPI.zip*. In order to create an XProc integration project follow the next steps:

1. Take the `oxygen.jar` from `[OXYGEN_DIR]/lib` and put it in the `lib` folder of your project.
2. Implement the `ro.sync.xml.transformer.xproc.api.XProcTransformerInterface` interface.
3. Create a Java archive (jar) from the classes you created.
4. Create a `engine.xml` file according with the `engine.dtd` file. The attributes of the `engine` element have the following meanings:

   1. `name` - The name of the XProc engine.
   2. `description` - A short description of the XProc engine.
   3. `class` - The complete name of the class that implements `ro.sync.xml.transformer.xproc.api.XProcTransformerInterface`.
   4. `version` - The version of this integration.

5. engineVersion - The version of the integrated engine.
6. vendor - The name of the vendor / implementer.
7. supportsValidation - true if the engine supports validation, false otherwise.

The engine element has only one child, runtime. The runtime element contains several library elements with the name attribute containing the relative or absolute location of the libraries necessary to run this integration.

5. Create a folder with the name of the integration in the [OXYGEN_DIR]/lib/xproc.
6. Put there the engine.xml, and all the libraries necessary to run the new integration.

XQuery Transformation

To create an **XQuery transformation** scenario, use one of the following methods:

* Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XQuery transformation**;
* Click the **Configure Transformation Scenario(s)** (**Ctrl+Shift+C (Command+Shift+C on OS X)**) button on the **Transformation** toolbar, then click the **New** button and select **XQuery transformation**;
* Select **Ctrl+Shift+T (Command+Shift+T on OS X)** on your keyboard or click the **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **XQuery transformation**.

  > **Note:** In case a scenario is already associated with the edited document, selecting **Ctrl+Shift+T (Command+Shift+T on OS X)** or **Apply Transformation Scenario** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the options that control the transformation.

The upper part of the dialog box contains the **Name** field and the **Storage** options:

* **Global Options** - the scenario is saved in the global options stored in the user home directory;
* **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined.

The lower part of the dialog box contains the following tabs:

* *the XQuery tab*;
* *the FO Processor tab*;
* *the Output tab*.

The XQuery Tab

The **XQuery** tab contains the following options:

* **XML URL** - specifies the source XML file that the transformation uses. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location;

  > **Note:** In case the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

* **XQuery URL** - specifies the source XQuery file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XQuery URL** fields:

Insert Editor Variables

Opens a pop-up menu allowing to introduce special *Oxygen XML Author editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing to select a local file.

Browse for remote file

Opens an URL browser dialog box allowing to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing to select a file from a zip archive.

Browse Data Source Explorer

Opens the *Data Source Explorer* window.

Search for file

Allows you to find a file in the current project.

Open in editor

Opens in an editor panel the file with the path specified in the **XML URL** text box.

The rest of the options available in the **XQuery** tab allow you to further customize the transformation scenario:

- **Transformer** - this combo box presents all the transformation engines available to Oxygen XML Author for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog is used as the default transformation engine. In case no validation scenario is associated with an XSLT or XQuery document, the transformation engine is used in the validation process, if it provides validation support;

  - **Advanced options** - *configure advanced options specific for the Saxon HE / PE / EE engine*

- **Parameters** - opens the **Configure parameters** dialog for configuring the XQuery parameters. If the XQuery/XSLT transformation engine is custom-defined you can not use this dialog to set parameters. Instead, copy all parameters from the dialog using the contextual menu actions and edit the custom XSLT/XQuery engine to include the necessary parameters in the command line which starts the transformation process

  > **Note:** Use the **Filter** text box to search for a specific term in the entire parameters collection.

- **Extensions** - opens *the dialog for configuring the XSLT/XQuery extension jars or classes* which define extension Java functions or extension XQuery elements used in the transformation

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation;
- **XQuery result as input** - the FO processor is applied to the result of the XQuery transformation defined in the **XQuery** tab;
- **XML URL as input** - the FO processor is applied to the input XML file;
- **Method** - the output format of the FO processing. Available options depend on the selected processor type;
- **Processor** - specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Prompt for file** - At the end of the transformation a file browser dialog is displayed for specifying the path and name of the file which stores the transformation result.
- **Save As** - The path of the file where the transformation result are stored. The path can include *special Oxygen XML Author editor variables* or *custom editor variables*.
- **Open in Browser/System Application** - If enabled, Oxygen XML Author opens the transformation result automatically, in a system application associated with the type of the result (HTML/XHTML, PDF, text) file.

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

- **Saved file** - When **Open in Browser/System Application** is selected this button can be used to specify that Oxygen XML Author 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 Author 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 Author editor variables* or *custom editor variable*.
- **Open in editor** - When this is enabled, the transformation result set in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type: if the result is an XML file it is opened with the built-in XML editor, if it is an XSL-FO file it is opened with the built-in FO editor, and so on.
- **Show As XHTML** - It is enabled only when **Open in browser** is disabled. If this is checked, Oxygen XML Author displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

  > **Important:** When transforming very large documents, you should be aware that enabling this feature results in a very long time necessary for rendering the transformation result in the XHTML result viewer panel. This drawback appears due to the built-in Java XHTML browser implementation. In these cases, if you wish to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

- **Show As XML** - If this is checked Oxygen XML Author displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlight* specific for XML documents.
- **Show As SVG** - If this is checked Oxygen XML Author displays the transformation result in an SVG viewer panel at the bottom of the application window by rendering the result as an SVG image.
- **Image URLs are relative to** - If **Show As XHTML** is checked this text field specifies the path used to resolve image paths contained in the transformation result.

SQL Transformation

To create an **SQL transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **SQL transformation**;
- Click the **Configure Transformation Scenario(s)** (**Ctrl+Shift+C (Command+Shift+C on OS X)**) button on the **Transformation** toolbar, then click the **New** button and select **SQL transformation**;
- Select **Ctrl+Shift+T (Command+Shift+T on OS X)** on your keyboard or click the **Apply Transformation Scenario** button on the **Transformation** toolbar to open the **Transform With** dialog. In this dialog click the **New** button and select **SQL transformation**.

  > **Note:** In case a scenario is already associated with the edited document, selecting **Ctrl+Shift+T (Command+Shift+T on OS X)** or **Apply Transformation Scenario** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog allows you to configure the following options that control the transformation:

- **Name** - the unique name of the transformation scenario;
- **Global Options** - the scenario is saved in the global options stored in the user home directory;
- **Project Options** - the scenario list is stored in the project file. In case your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, and so on.), your team can use the scenarios you defined;
- **SQL URL** - specifies the URL of the SQL script. This field also accepts *editor variables*.
- **Connection** - opens the *data source preferences page*;
- **Parameters** - allows you to configure the parameters of the transformation.

XSLT/XQuery Extensions

The **Libraries** dialog is used to specify the jars and classes containing extension functions called from the XSLT/XQuery file of the current transformation scenario.

An extension function called from the XSLT or XQuery file of the current transformation scenario will be searched in the specified extensions in the order of the list displayed in the dialog. For changing the order of the items the user must select the item that must be moved to other position in the list and press the ⬆ up and ⬇ down buttons.

The Configure Transformation Scenario(s) Dialog

You can use this dialog to manage both the *built-in transformation scenarios* and the ones you create.

To open it, either click the 🔧▶ **Configure Transformation Scenario(s)** button on the application toolbar, or go to **Document** > **Transformation** > **Configure transformation scenario. (Ctrl+Shift+C (Command+Shift+C on OS X))**.
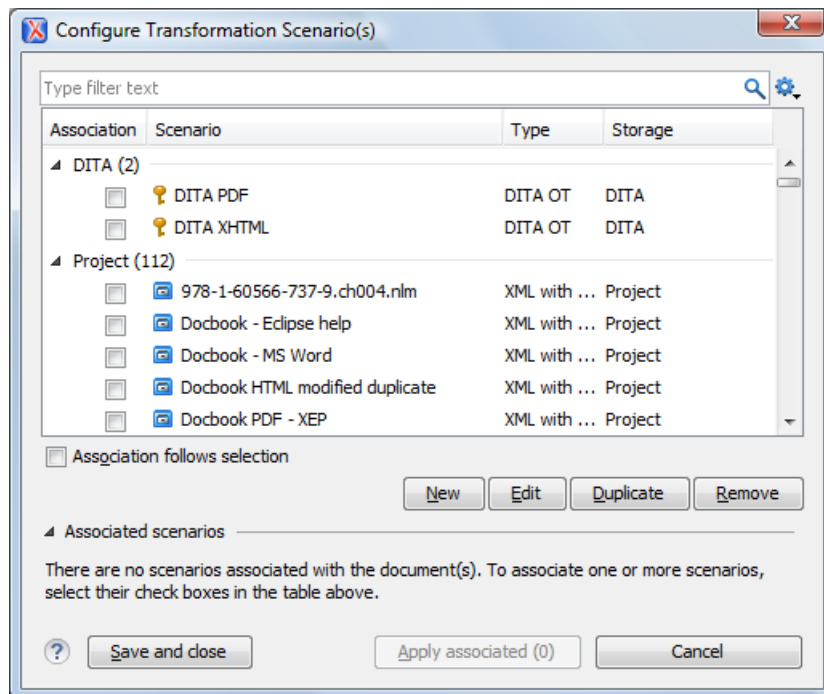
Figure 172: Configure Transformation Scenario(s) Dialog

The top section of the dialog contains a filter that allows you to search through the scenarios list. Using the **Settings** button you can configure the following options:

- **Show all scenarios** - select this option to display all the available scenarios, regardless of the document they are associated with;
- **Show only the scenarios available for the editor** - select this option to display in the **Transformation scenarios** view only the scenarios which Oxygen XML Author can execute for the current document type;
- **Show associated scenarios** - select this option to display only the scenarios associated with the document you are editing;
- 📤 **Import scenarios** - use this option to open the **Import scenarios** dialog. The **Import scenarios** dialog allows you to select which scenario of the `scenarios` files you want to import. In case one of the scenarios you import is identical with an existing scenario, Oxygen XML Author ignores the scenario you are importing. In case a conflict appears (an imported scenario has the same name with an existing one), you can choose between two options:

  - keep or replace the existing scenario;
  - keep both scenarios.

> **Note:** When you keep both scenarios, Oxygen XML Author adds `imported` to the name of the imported scenario.

- **Export selected scenarios** - use this option to export transformation and validation scenarios individually. Oxygen XML Author creates a `scenarios` file which contains the scenarios you export.

The middle section of the dialog presents the scenarios that you can apply to the document you are editing. You can view both the scenarios associated with the current document type and the scenarios defined at project level. The following columns are used to display the transformation scenarios:

- **Association** - the check-boxes in this column mark whether a transformation scenario is associated with the document you are editing;
- **Scenario** - this column presents the names of the transformation scenarios;
- **Type** - specifies the type of the transformation scenario. For further details about the different types of transformation scenarios available in Oxygen XML Author go to *Defining a New Transformation Scenario*;
- **Storage** - specifies whether a transformation scenario is defined at project level.

Left click the header of each column to sort its items. The contextual menu of each header allows you to show or hide the **Type** and **Storage** columns, group the columns by **Association**, **Type**, and **Storage**, ungroup all of them, or reset the layout.

The bottom section of the dialog contains the following actions:

- **Association follows selection** - enable this check-box to associate selected transformation scenarios automatically with the document you are editing. This option also works for multiple selection;

  > **Note:** When this option is enabled, the **Association** column is no longer presented.

- **New** - this button allows you to create a new transformation scenario *depending on its type*;
- **Edit** - this button opens the **Edit scenario** dialog which allows you to configure the options of the transformations scenario you are editing;

  > **Note:** In case you try to edit a transformation scenario associated with a document type, Oxygen XML Author displays a warning message to inform you that this is not possible. You have the option to create a *duplicated transformation scenario* and edit this one instead.

- **Duplicate** - use this button to create a *duplicated transformation scenario*;
- **Remove** - use this button to remove transformation scenarios.

  > **Note:** Removing scenarios associated with a document type is not permitted.

The **Edit**, **Duplicate**, and **Remove** actions are also available in the contextual menu of the transformation scenarios listed in the middle section of the **Configure Transformation Scenario(s)**. This contextual menu also contains the *Import scenarios* and *Export selected scenarios* actions and allows you to change the storage of a transformation scenario.

Duplicating a Transformation Scenario

Use the following procedure to duplicate a transformation scenario.

1. Go to menu **Document** > **Transformation** > **Configure Transformation Scenario (Ctrl+Shift+C (Command+Shift+C on OS X)**) to open the **Configure Transformation** dialog.
2. Click the **Duplicate Scenario** button of the dialog to create a copy of the current scenario.
3. Click the **Name** field and type a new name.
   a) You can choose to save the scenarios at project level by setting the **Project Scenarios** setting.
4. Click **OK** or **Transform Now** to save the scenario.

Editing a Transformation Scenario

Editing a transformation scenario is useful in case you run an existing one and you need to configure some of its parameters.

Oxygen XML Author allows you to configure exiting transformation scenarios either from the **Transformation Scenarios** view, or form the 🔧 **Configure Transformation Scenario(s)**, and **Transform with** dialog boxes.

Use one of the following methods to configure a transformation scenario:

- go to the **Transformation Scenarios** view, select a transformation scenario and click the 🔧 **Edit** button on the toolbar of the view;
- click the 🔧 **Configure Transformation Scenario(s)** button on the **Transformation** toolbar, select a transformation scenario from the **Configure Transformation Scenario(s)** dialog box and click the **Edit** button;
- click the ▶ **Apply Transformation Scenario** button on the **Transformation** toolbar, select a transformation scenario from the **Transform With** dialog box and click the **Edit** button.

  📄 **Note:** In case a transformation scenario is associated with the document you are editing, selecting the ▶ **Apply Transformation Scenario** button starts the transformation process automatically.

You can edit transformation scenarios that are defined at project level only. To edit a transformation scenario that is associated with a document type, duplicate it and edit the duplicated scenario.

Batch Transformation

A transform action can be applied on a batch of files *from the **Project** view's contextual menu* without having to open the files involved in the transformation:

▶ **Apply Transformation Scenario(s)**

Applies the transformation scenario associated to each of the selected files. If the currently processed file does not have an associated transformation scenario then a warning is displayed in the **Warnings** view.

🔧 **Configure Transformation Scenario(s)...**

Opens a dialog box that allows you to create, manage and set default transformation scenarios.

🔧 **Transform with...**

Allows you to select one transformation scenario to be applied to each one of the currently selected files.

xsl:message output information is collected and displayed in the *Results View*. All entries in the Results View point to the location of the code that triggered them.

📄 **Note:** When you trigger a batch transformation, the output messages from the previous one are deleted.

Built-in Transformation Scenarios

Oxygen XML Author comes with preconfigured built-in transformation scenarios used for usual transformations. To obtain the output simply one of the built-in scenarios with the currently edited document and click the ▶ **Apply Transformation Scenario(s)** button.

You can use the ▶ **Apply Transformation Scenario(s)** button even if the document you are editing is not associated with a transformation scenario.

In case the document contains an `xml-stylesheet` processing instruction referring to an XSLT stylesheet (commonly used to display the document in web browsers), Oxygen XML Author prompts you to associate the document with a built-in transformation scenario.

The default transformation scenario is suggested based on the processing instruction from the edited document. The **XSL URL** field of the default transformation scenario contains the URL from the `href` attribute of the processing

instruction. The **Use xml-stylesheet declaration** check-box of the scenario is selected by default. Saxon is used as transformation engine and no FO processing is performed. The result of the transformation is store in a file with the same URL as the edited document, but the extension is changed to `html`. The name and path are preserved because the output file name is specified with the help of two *editor variables*: ${cfd} and ${cfn}.

Sharing the Transformation Scenarios

The transformation scenarios can be shared with other users by saving them at project level. In the lower part of the dialog showing the list of scenarios you will find two radio buttons controlling where the scenarios are stored: **Global Scenarios** and **Project Scenarios**.
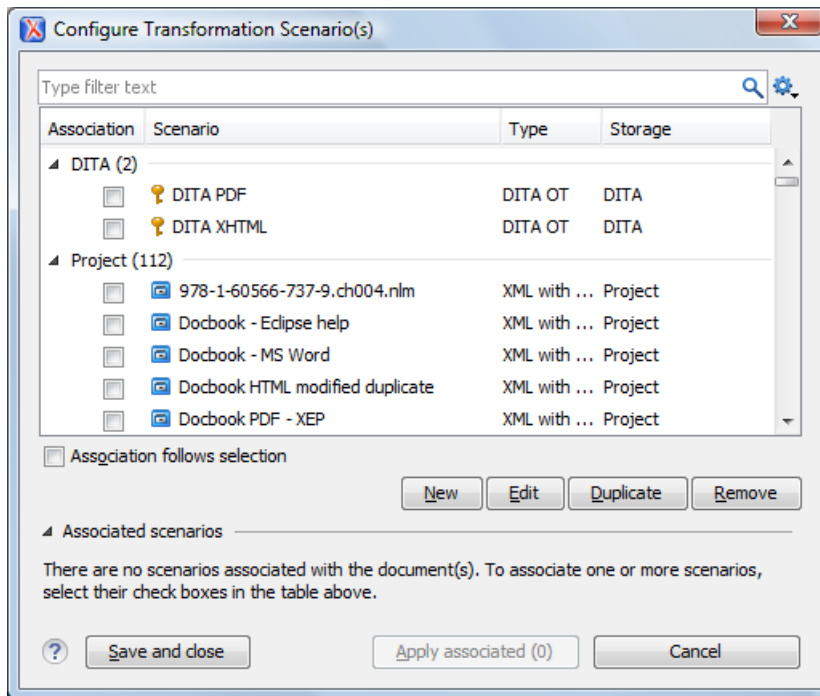
Figure 173: Transformation Scenario List Dialog

Selecting **Global Scenarios** ensures that the scenarios are saved in the global options stored in the user home directory.

After changing the selection to **Project Scenarios**, the scenario list will be stored in the project file. If your project is saved on a source versioning/sharing system (CVS, SVN, Source Safe, etc..) then your team can use the scenarios you defined. When you create a scenario at the project level, the URLs from the scenario become relative to the project URL.

Predefined scenarios are presented according to the current document's detected type. The screenshot above shows all default scenarios for a *DITA* document and several custom transformation scenarios. The key symbol ⬛ before the scenario name, indicates that the scenario can only be modified from the *Document Type Association* preferences page.

Other preferences can also be stored at the project level. For more information, see the *Preference Sharing* section.

Transformation Scenarios View

You are able to manage the transformation scenarios easily, using the **Transformation Scenarios** view. To open this view, go to **Window** > **Show View** > **Transformation Scenarios**.
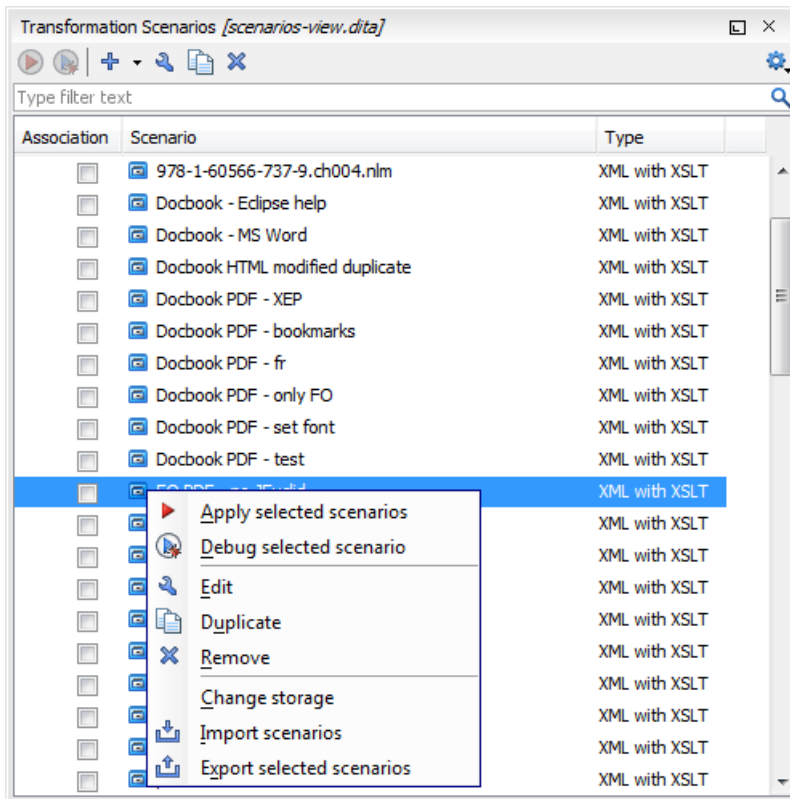
Figure 174: The Scenarios view

The following options are available in the contextual menu of the **Transformation Scenarios** view:

Apply selected scenarios

Select this option to run the current transformation scenario.

Debug selected scenario

Select this option to switch to the **Debugger** perspective and initialize it with the parameters from the scenario: the XML input, the XSLT, or XQuery input, the transformation engine, the XSLT parameters.

Duplicate

Adds a new scenario to the list that is a duplicate of the current scenario. It is useful for creating a scenario that is the same as an existing one but needs some changes.

Edit

Opens the dialog for editing the parameters of a transformation scenario.

Remove

Removes the current scenario from the list. This action is also available on the **Delete** key.

Change storage

Use this option to change the storage location of the selected scenario. You are also able to keep the original storage location and make a copy of the selected scenario in the target storage location.

Import scenarios

Use this option to open the **Import scenarios** dialog. The **Import scenarios** dialog allows you to select which scenario of the `scenarios` file you want to import. In case one of the scenarios you import is identical with an existing scenario, Oxygen XML Author ignores the scenario you are importing. In case a conflict appears (an imported scenario has the same name with an existing one), you can choose between two options:

- keep or replace the existing scenario
- keep both scenarios

Note: When you keep both scenarios, Oxygen XML Author adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export transformation and validation scenarios individually. Oxygen XML Author creates a `scenarios` file which contains the scenarios you export.

Apart from the options available in the contextual menu, the **Transformation Scenarios** view toolbar contains a

New drop-down button. The menu of this drop down button contains a list of the scenarios you can create. Oxygen XML Author analyses which scenario is appropriate and displays it as the first item in the list of scenarios, separated by the rest through a horizontal line.

The **Settings** drop-down menu of the **Transformation Scenarios** view allows you to configure the following options:

- **Show all scenarios** - select this option to display all the available scenarios, regardless of the document they are associated with
- **Show only the scenarios available for the editor** - select this option to display in the **Transformation scenarios** view only the scenarios which Oxygen XML Author can execute for the current document type
- **Show associated scenarios** - select this option to display only the scenarios associated with the document you are editing
- **Change storage** - use this option to change the storage location of the selected scenario. You are also able to keep the original storage location and make a copy of the selected scenario in the target storage location
- **Import scenarios** - use this option to open the **Import scenarios** dialog. The **Import scenarios** dialog allows you to select which scenario of the `scenarios` file you want to import. In case one of the scenarios you import is identical with an existing scenario, Oxygen XML Author ignores the scenario you are importing. In case a conflict appears (an imported scenario has the same name with an existing one), you can choose between two options:

  - keep or replace the existing scenario
  - keep both scenarios

    Note: When you keep both scenarios, Oxygen XML Author adds `imported` to the name of the imported scenario.

- **Export selected scenarios** - use this option to export transformation and validation scenarios individually. Oxygen XML Author creates a `scenarios` file which contains the scenarios you export
- **Show Type** - use this option to display the transformation type of each scenario in the **Transformation Scenarios** view
- **Show Storage** - use this option to display the storage location of the scenarios in the **Transformation Scenarios** view
- **Group by Association** - select this option to group the scenarios in the **Transformation Scenarios** view depending on whether they are associated or not with the document you are editing
- **Group by Type** - select this option to group the scenarios in the **Transformation Scenarios** view by their type
- **Group by Storage** - select this option to group the scenarios in the **Transformation Scenarios** view by their storage location
- **Ungroup all** - select this option to ungroup all the scenarios of the **Transformation Scenarios** view
- **Reset Layout** - select this option to restore the default settings of the layout of the **Transformation Scenarios** view

Oxygen XML Author supports multiple scenarios association. To associate multiple scenarios with a document, enable the check-boxes in front of each scenario. You can also associate multiple scenarios with a document from the **Configure Transformation Scenario(s)** or **Configure Validation Scenario(s)** dialogs.

The **Transformation Scenarios** presents both global scenarios and project scenarios. By default, Oxygen XML Author presents the items in the **Transformation Scenarios** in the following order: scenarios matching the current framework, scenarios matching the current project, scenarios matching other frameworks. You can group the scenarios depending

on the columns in the **Transformation Scenarios** view. Right click the name of a column to choose how to group the scenarios. The following grouping options are available:

- **Group by Association** - select this option to group the scenarios in the **Transformation Scenarios** view depending on whether they are associated or not with the document you are editing
- **Group by Type** - select this option to group the scenarios in the **Transformation Scenarios** view by their type
- **Group by Storage** - select this option to group the scenarios in the **Transformation Scenarios** view by their storage location

The WebHelp Skin Builder

The **WebHelp Skin Builder** is a simple, easy-to-use tool, specially designed to assist users to visually customize the look and feel of the WebHelp output. It is implemented as an online tool hosted on the Oxygen XML Author website and allows you to experiment with different styles and colors over an inert documentation sample.

To be able to use the **Skin Builder**, you need:

- an Internet connection and unrestricted access to Oxygen XML Author website;
- a later version web browser.

To start the **Skin Builder**, do one of the following:

- from a web browser navigate to *http://www.oxygenxml.com/webhelp-skin-builder*;
- from the Oxygen XML Author in the *Skins tab*, click the **Online preview** link. In the upper section of the preview, click the **Select Skin** button, then choose **Customize Skin**.

The Skin Builder Layout

The left side panel of the *Skin Builder* is divided into 3 sections:

- **Actions** - holds two buttons:
  - **Import** - allows you to load a CSS stylesheet and applies it over the documentation sample;
  - **Export** - saves all properties as a CSS file.
- **Settings** - contains the **Highlight selection** checkbox which helps you identify the areas affected by a particular element customization:
  - when hovering an item in the customizable elements menu, the affected sample area is highlighted with a dotted blue border;
  - when an item in the customizable elements menu is selected, the affected sample area is highlighted with a solid red border.
- **Customize** - provides a series of customizable elements organized under four main categories:
  - Header
  - TOC Area
  - Vertical Splitter
  - Content

  For each customizable element you can alter properties like background color or font face. Any alteration made in the customizable elements menu is applied in real time over the sample area.

Create a Customization Skin

- The starting point can be either one of the predefined skins or a CSS stylesheet applied over the sample using the **Import** button.
- Use the elements in the **Customize** section to set properties that modify the skin's look. By default, all customizable elements display a single property, but you can make more properties visible if you click the ■ button and choose from the available ones.

📑 **Note:** If you want to revert a setting of a particular property to its initial value, press the ■ button.

- When you are happy with the skin customization you have made, press the **Export** button. All settings will be saved in a CSS file.

Apply a Customization Skin to a DITA Map to WebHelp Transformation Scenario

- Start Oxygen XML Author.
- Load the DITA Map you want to produce as a WebHelp output.
- Edit a *DITA Map to WebHelp*-type transformation scenario. Set the previously exported CSS file in the **Custom** section of the **Skins** tab.
- Execute the transformation to obtain the WebHelp output.

Apply a Customization Skin to a DocBook to WebHelp Transformation Scenario

- Start Oxygen XML Author.
- Load the DocBook file you want to produce as a WebHelp output.
- Edit a *DocBook to WebHelp*-type transformation scenario. Set the previously exported CSS file in the **Custom** section of the **Skins** tab.
- In the **Parameters** tab, set the `webhelp.skin.css` parameter to point to the previously exported CSS.
- To customize the logo, use the following parameters:

  - `webhelp.logo.image` - specifies a path to an image displayed as a logo in the left side of the output's header;
  - `webhelp.logo.image.target.url` - specifies a target URL set on the logo image. When you click the logo image, you will be redirected to this address.

Using the Oxygen XML WebHelp Plugin

Oxygen XML WebHelp allows you to transform DITA and DocBook documents outside of Oxygen XML Author, from a command line. The WebHelp output files created with the Oxygen XML WebHelp plugin are the same with the output files that Oxygen XML Author produces when you run DITA or DocBook to WebHelp transformation scenarios.

⚠ **Important:** It is prohibited to:

- embed Oxygen XML WebHelp or any derivative works into another application and to distribute such to third parties;
- use Oxygen XML WebHelp outside of Oxygen XML Author (by means of a command line or external process).

  If you wish to use Oxygen XML WebHelp for such purposes, you need a separate license, available here: *http://www.oxygenxml.com/buy_webhelp.html*

Oxygen XML WebHelp Plugin for DITA

To transform DITA documents using the Oxygen XML WebHelp plugin, first integrate the plugin with the DITA Open Toolkit. The purpose of the integration is to add to the DITA Open Toolkit the following transformation types:

- **webhelp** - the transformation that produces *Web help* output for desktop
- **webhelp-feedback** - the transformation that produces feedback-enabled *Web help* for desktop
- **webhelp-mobile** - the transformations that produces *Web help* output for mobile devices

Integrating the Oxygen XML WebHelp Plugin with the DITA Open Toolkit

The requirements of the Oxygen XML WebHelp plugin for the DITA Open Toolkit are:

- Java Virtual Machine 1.6 or later

- DITA Open toolkit 1.6.x or 1.7.x (Full Easy Install)
- Saxon 9.1.0.8

To integrate the Oxygen XML WebHelp plugin with the DITA Open Toolkit, follow these steps:

1. Download and install a *Java Virtual Machine* 1.6 or later.
2. Download and install *DITA Open Toolkit* 1.6.x or 1.7.x (Full Easy Install).
3. Navigate to the `plugins` directory located in the installation directory of the DITA Open Toolkit.
4. Copy the `com.oxygenxml.webhelp` and `com.oxygenxml.highlight` directories inside the `plugins` directory. The `com.oxygenxml.highlight` directory add syntax highlight capabilities to your WebHelp output for codeblock sections that contain source code snippets (XML, Java, JavaScript etc.).
5. In the home directory of the DITA Open Toolkit, run `ant -f integrator.xml`.
6. Go to *http://sourceforge.net/projects/saxon/files/Saxon-B/9.1.0.8/*, download and unzip the `processor.saxonb9-1-0-8j.zip` file (contains the Saxon 9.1.0.8).

Registering the Oxygen XML WebHelp Plugin

To register the Oxygen XML WebHelp plugin for the DITA Open Toolkit, follow these steps:

1. Create a `.txt` file named `license` in the `[DITA-OT-install-dir]/plugins/com.oxygenxml.webhelp` directory.
   The `license.txt` file is created.
2. In this file, copy your license key which you purchased for your Oxygen XML WebHelp plugin.

   The WebHelp transformation process reads the Oxygen XML Author license key from this file. In case the file does not exit, or it contains an invalid license, an error message will be displayed.

Running a DITA Transformation Using the Oxygen XML WebHelp Plugin

To run a DITA to WebHelp (**webhelp**, **webhelp-feedback**, **webhelp-mobile**) transformation using the Oxygen XML WebHelp plugin, use:

- the `dita.bat` script file for Windows based systems;
- the `dita.sh` script file for Unix/Linux based systems.

   > **Note:** You can call these files in an automated process or from the command line.

The `dita.bat` and the `dita.sh` files are located in the home directory of the Oxygen XML WebHelp Plugin. Before using them to generate an WebHelp system, customize them to match the paths to the JVM, DITA Open Toolkit and Saxon engine, and also to set the transformation type. To do this, open a script file and edit the following variables:

- `JVM_INSTALL_DIR` - specifies the path to the Java Virtual Machine installation directory on your disk;
- `DITA_OT_INSTALL_DIR` - specifies the path to DITA Open Toolkit installation directory on your disk;
- `SAXON_9_DIR` - specifies the path to the directory on your disk where you unzipped the Saxon 9 archive files;
- `TRANSTYPE` - specifies the type of the transformation you want to execute. You can set it to `webhelp`, `webhelp-feedback` and `webhelp-mobile`;
- `DITA_MAP_BASE_DIR` - specifies the path to the directory where the input DITA Map file is located;
- `DITAMAP_FILE` - specifies the input DITA Map file;
- `DITAVAL_FILE` - specifies the `.ditaval` input filter that the transformation process applies to the input DITA Map file;
- `DITAVAL_DIR` - specifies the path to the directory where the `.ditaval` file is located;
- `Doutput.dir` - specifies the output directory of the transformation.

The `-Dargs.filter` and the `-Ddita.input.valfile` parameters are optional.

Additional Oxygen XML WebHelp Plugin Parameters for DITA

You are able to append the following parameters to the command line that runs the transformation:

- `-Dwebhelp.copyright` - the copyright note that is added in the footer of the Table of Contents frame;

- `-Dwebhelp.footer.file` - specifies the location of a well-formed XHTML file containing your custom footer for the document body. Corresponds to the WEBHELP_FOOTER_FILE XSLT parameter. The fragment must be a well-formed XHTML, with a single root element. As a common practice, place all the content into a <div> element;
- `-Dwebhelp.footer.include` - specifies whether the content of file set in the `-Dwebhelp.footer.file` is used as footer in the WebHelp pages. Its values can be `yes`, or `no`;
- `-Dwebhelp.product.id` - the value of this parameter is a text string, that the **webhelp-feedback** transformation requires. It represents a short name of the documentation target (product). All the user comments that are posted in the WebHelp output pages and are added in the comments database are bound to this product ID;

  > **Note:** You can deploy documentation for multiple products on the same server.

- `-Dwebhelp.product.version` - the value of this parameter is a text string, that the **webhelp-feedback** transformation requires. It specifies the documentation version number, for example: 1.0, 2.5, etc. New user comments are bound to this version.

  > **Note:** Multiple documentation versions can be deployed on the same server.

In case you need to further customize the transformation process, you are able to append other DITA-OT parameters as well. Any parameter that you want to append must follow the `-D` model of the above parameters. For example, to append the `args.hdr` parameter, use:

```
-Dargs.hdr=[HEADER_FILE_DIR]
```

where `[HEADER_FILE_DIR]` is the location of the directory that contains the header file.

Database Configuration for DITA WebHelp with Feedback

In case you run the **webhelp-feedback** transformation, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `installation.html` file, located at `[DITA_MAP_BASE_DIR]/out/[TRANSFORM_TYPE]/oxygen-webhelp/resources`. The `installation.html` file is created by the transformation process.

Oxygen XML WebHelp Plugin for DocBook

To transform DocBook documents using the Oxygen XML WebHelp plugin, first integrate the plugin with the DocBook XSL distribution. The purpose of the integration is to add to the DocBook XSL distribution the following transformation types:

- **webhelp** - the transformation that produces *Web help* output for desktop
- **webhelp-feedback** - the transformation that produces feedback-enabled *Web help* for desktop
- **webhelp-mobile** - the transformations that produces *Web help* output for mobile devices

Integrating the Oxygen XML WebHelp Plugin with the DocBook XSL Distribution

The WebHelp plugin transformations run as an ANT build script. The requirements are:

- ANT 1.8 or later;
- Java Virtual Machine 1.6 later;
- DocBook XSL 1.78.1 later;
- Saxon 6.5.5;
- Saxon 9.1.0.8.

To integrate the Oxygen XML WebHelp plugin with the DocBook XSL distribution, follow these steps:

1. Download and install a *Java Virtual Machine* 1.6 or later.
2. Download and install *ANT 8.0* or later.
3. Download and unzip on your computer the DocBook XSL distribution.
4. Unzip the Oxygen XML WebHelp distribution package in the DocBook XSL installation directory.

The DocBook XSL directory now contains a new subdirectory named `com.oxygenxml.webhelp` and two new files, `oxygen_custom.xsl` and `oxygen_custom_html.xsl`.

5. Download and unzip *saxon6-5-5.zip* on your computer.

6. Download and unzip *saxonb9-1-0-8j.zip* on your computer.

Registering the Oxygen XML WebHelp Plugin

To register the Oxygen XML WebHelp plugin for the DocBook XSL distribution, follow these steps:

1. Create a `.txt` file named `license` in the `com.oxygenxml.webhelp` subdirectory of the DocBook XSL directory.

2. In this file, copy the license key, which you purchased for your Oxygen XML WebHelp plugin.

   The WebHelp transformation process reads the Oxygen XML Author license key from this file. If the file does not exit, or it contains an invalid license, an error message is displayed.

Running a DocBook Transformation Using the WebHelp Plugin

To run a DocBook to WebHelp (**webhelp**, **webhelp-feedback**, **webhelp-mobile)** transformation using the Oxygen XML WebHelp plugin, use:

- the `docbook.bat` script file for Windows based systems;
- the `docbook.sh` script file for Unix/Linux based systems.

> **Note:** You can call these files in an automated process or from the command line.

The `docbook.bat` and the `docbook.sh` files are located in the home directory of the Oxygen XML WebHelp Plugin. Before using them to generate an WebHelp system, customize them to match the paths to the JVM, DocBook XSL distribution and Saxon engine, and also to set the transformation type. To do this, open a script file and edit the following variables:

- `JVM_INSTALL_DIR` - specifies the path to the Java Virtual Machine installation directory on your disk;
- `ANT_INSTALL_DIR` - specifies the path to the installation directory of ANT;
- `SAXON_6_DIR` - specifies the path to the installation directory of Saxon 6.5.5;
- `SAXON_9_DIR` - specifies the path to the installation directory of Saxon 9.1.0.8;
- `DOCBOOK_XSL_DIR` - specifies the path to the installation directory of the DocBook XSL distribution;
- `TRANSTYPE` - specifies the type of the transformation you want to execute. You can set it to `webhelp`, `webhelp-feedback` and `webhelp-mobile`;
- `INPUT_DIR` - specifies the path to the input directory, containing the input XML file;
- `XML_INPUT_FILE` - specifies the name of the input XML file;
- `OUTPUT_DIR` - specifies the path to the output directory where the transformation output is generated;
- `DOCBOOK_XSL_DIR_URL` - specifies the path to the directory of the DocBook XSL distribution in URL format.

Additional Oxygen XML WebHelp Plugin Parameters for DocBook

You are able to append the following parameters to the command line that runs the transformation:

- `-Dwebhelp.copyright` - the copyright note (a text string value) that is added in the footer of the table of contents frame (the left side frame of the WebHelp output);
- `-Dwebhelp.footer.file` - specifies the location of a well-formed XHTML file containing your custom footer for the document body. Corresponds to the `WEBHELP_FOOTER_FILE` XSLT parameter . The fragment must be an well-formed XHTML, with a single root element. As a common practice, place all the content inside a <div> element;
- `-Dwebhelp.footer.include` - specifies whether the content of file set in the `-Dwebhelp.footer.file` is used as footer in the WebHelp pages. Its values can be `yes`, or `no`;
- `-Dwebhelp.product.id` - the value of this parameter is a text string, that the **webhelp-feedback** transformation requires. It represents a short name of the documentation target (product). All the user comments that are posted in the WebHelp output pages and are added in the comments database are bound to this product ID;

Note: You can deploy documentation for multiple products on the same server.

- `-Dwebhelp.product.version` - the value of this parameter is a text string, that the **webhelp-feedback** transformation requires. It specifies the documentation version number, for example: 1.0, 2.5, etc. New user comments are bound to this version.

    Note: Multiple documentation versions can be deployed on the same server.

In case you need to further customize your transformation, other DocBook XSL parameters can be appended. Any parameter that you want to append must follow the `-D` model of the above parameters. For example, you can append the `html.stylesheet` parameter in the following form:

```
-Dhtml.stylesheet=/path/to/directory/of/stylesheet.css
```

Database Configuration for DocBook WebHelp with Feedback

In case you run the **webhelp-feedback** transformation, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `installation.html` file, located at `[OUTPUT_DIR]/oxygen-webhelp/resources/installation.html`. The `installation.html` file is created by the transformation process.

XSLT Processors

This section explains how to configure an XSLT processor and extensions for such a processor in Oxygen XML Author.

Supported XSLT Processors

Oxygen XML Author comes with the following XSLT processors:

- **Xalan 2.7.1** - *Xalan-Java* is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements XSL Transformations (XSLT) Version 1.0 and XML Path Language (XPath) Version 1.0.
- **Saxon 6.5.5** - *Saxon 6.5.5* is an XSLT processor, which implements the Version 1.0 XSLT and XPath with a number of powerful extensions. This version of Saxon also includes many of the new features that were first defined in the XSLT 1.1 working draft, but for conformance and portability reasons these are not available if the stylesheet header specifies `version="1.0"`.
- **Saxon 9.5.1.7 Home Edition (HE), Professional Edition (PE)** - *Saxon-HE/PE* implements the basic conformance level for XSLT 2.0 / 3.0 and XQuery 1.0. The term *basic XSLT 2.0 / 3.0 processor* is defined in the draft XSLT 2.0 / 3.0 specifications: it is a conformance level that requires support for all features of the language other than those that involve schema processing. The HE product remains open source, but removes some of the more advanced features that were present in Saxon-PE.
- **Saxon 9.5.1.7 Enterprise Edition (EE)** - *Saxon EE* is the schema-aware edition of Saxon and it is one of the built-in processors of Oxygen XML Author. Saxon EE includes an XML Schema processor, and schema-aware XSLT, XQuery, and XPath processors.

    The validation in schema aware transformations is done according to the W3C XML Schema 1.0 or 1.1. This can be *configured in Preferences*.

    Note: Oxygen XML Author implements a Saxon framework that allows you to create Saxon configuration files. Two templates are available: **Saxon collection catalog** and **Saxon configuration**. Both these templates support content completion, element annotation and attribute annotation.

Besides the above list Oxygen XML Author 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 Author is 2.7.6 and the libxslt version is .

Oxygen XML Author uses Libxslt through its command line tool (`Xsltproc`). The XSLT processor is included into the distribution kit of the stand-alone version for Windows and Mac OS X. Because there are differences between different Linux distributions, on Linux you must install `Libxslt` on your machine as a separate application and set the PATH variable to contain the `Xsltproc` executable.

The Xsltproc processor can be configured from the *XSLTPROC options page*.

⚠️ **Caution:** Known problem: file paths containing spaces are not handled correctly in the LIBXML processor. For example the built-in XML catalog files of the predefined document types (DocBook, TEI, DITA, etc) are not handled by LIBXML if Oxygen XML Author is installed in the default location on Windows (C:\Program Files) because the built-in XML catalog files are stored in the `[OXYGEN_DIR]/frameworks` subdirectory of the installation directory which in this case contains at least a space character.

- **MSXML 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* .

  Oxygen XML Author 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* .

  Oxygen XML Author 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 Author 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*

The button ⚙ **Transformation options** available on the **Transformation** toolbar allows quick access to the *XSLT options* in the Oxygen XML Author user preferences.

Configuring Custom XSLT Processors

You can configure and run XSLT and XQuery transformations with processors other than *the ones which come with the Oxygen XML Author distribution*.

> 📋 **Note:**

The output messages of a custom processor are displayed in an output view at the bottom of the application window. If an output message follows *the format of an Oxygen XML Author linked message*, then a click on the message in the output view highlights the location of the message in an editor panel containing the file referred in the message.

Configuring the XSLT Processor Extensions Paths

The Xalan and Saxon processors support the use of extension elements and extension functions. Unlike a literal result element, which the stylesheet simply transfers to the result tree, an extension element performs an action. The extension is usually used because the XSLT stylesheet fails in providing adequate functions to the user for accomplishing a more complex task.

The DocBook extensions for Xalan and Saxon are included in the `[OXYGEN_DIR]\frameworks\docbook\xsl\extensions` folder.

Samples on how to use extensions can be found at:

- for Xalan - *http://xml.apache.org/xalan-j/extensions.html*
- for Saxon 6.5.5 - *http://saxon.sourceforge.net/saxon6.5.5/extensions.html*
- for Saxon 9.5.1.7 - *http://www.saxonica.com/documentation/extensibility/intro.xml*

To set an XSLT processor extension (a directory or a `jar` file), use *the **Extensions** button* of the scenario edit dialog. The old way of setting an extension (using the parameter -Dcom.oxygenxml.additional.classpath) was deprecated and you should use the extension mechanism of the XSLT transformation scenario.

XSL-FO Processors

This section explains how to apply XSL-FO processors when transforming XML documents to various output formats in Oxygen XML Author.

The Built-in XSL-FO Processor

The Oxygen XML Author installation package is distributed with the *Apache FOP* that is a Formatting Objects processor for rendering your XML documents to PDF. *FOP* is a print and output independent formatter driven by XSL Formatting Objects. *FOP* is implemented as a Java application that reads a formatting object tree and renders the resulting pages to a specified output.

To include PNG images in the final PDF document you need the *JIMI* or *JAI* libraries. For PDF images you need the `fop-pdf-images` library. These libraries are not bundled with Oxygen XML Author 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 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) Open the preferences dialog (**Options** > **Preferences**), go to **XML** > **XSLT/FO/XQuery** > **FO Processors**, and
      enter the path of the FOP configuration file in the **Configuration file for the built-in FOP** text field.

2. Set the font on the document content.

   This is done usually with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

   - For DocBook documents you can start with the predefined scenario called **DocBook PDF**, *edit the XSLT parameters*
     and set the font name (in our example the font family name is **Arial Unicode MS**) to the parameters
     body.font.family and title.font.family.
   - For TEI documents you can start with the predefined scenario called **TEI PDF**, *edit the XSLT parameters* and
     set the font name (in our example **Arial Unicode MS**) to the parameters bodyFont and sansFont.
   - For DITA transformations to PDF using DITA-OT you should modify the following two files:

     - [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml
       - the font-face element included in each element physical-font having the attribute
       char-set="default" must contain the name of the font (**Arial Unicode MS** in our example)
     - [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/fop/conf/fop.xconf
       - an element auto-detect must be inserted in the element fonts which is inside the element renderer
       having the attribute mime="application/pdf":

```
<renderer mime="application/pdf">
  . . .
  <fonts>
      <auto-detect/>
  </fonts>
  . . .
</renderer>
```

Add a Font to the Built-in FOP

If an XML document is transformed to PDF using the built-in Apache FOP processor but it contains some Unicode
characters that cannot be rendered by the default PDF fonts, then a special font that is capable to render these characters
must be configured and embedded in the PDF result.

> ⚠ **Important:** If this special font is installed in the operating system, there is a simple way of telling FOP to look
> for it. See *the simplified procedure for adding a font to FOP*.

1. Locate the font.

   First, find out the name of a font that has the glyphs for the special characters you used. One font that covers most characters, including Japanese, Cyrillic, and Greek, is Arial Unicode MS.

   On Windows the fonts are located into the `C:\Windows\Fonts` directory. On Mac, they are placed in `/Library/Fonts`. To install a new font on your system, is enough to copy it in the `Fonts` directory.

2. Generate a font metrics file from the font file.

   a) Open a terminal.
   b) Change the working directory to the Oxygen XML Author install directory.
   c) Create the following script file in the Oxygen XML Author installation directory.

      For OS X and Linux create a file `ttfConvert.sh`:

      ```
      #!/bin/sh

      export LIB=lib
      export CP=$LIB/fop.jar
      export CP=$CP:$LIB/avalon-framework-4.2.0.jar
      export CP=$CP:$LIB/xercesImpl.jar
      export CP=$CP:$LIB/commons-logging-1.1.1.jar
      export CP=$CP:$LIB/commons-io-1.3.1.jar
      export CP=$CP:$LIB/xmlgraphics-commons-1.5.jar
      export CP=$CP:$LIB/xml-apis.jar
      export CMD="java -cp $CP org.apache.fop.fonts.apps.TTFReader"
      export FONT_DIR='.'

      $CMD $FONT_DIR/Arialuni.ttf Arialuni.xml
      ```

      For Windows create a file `ttfConvert.bat`:

      ```
      @echo off
      set LIB=lib
      set CP=%LIB%\fop.jar
      set CP=%CP%;%LIB%\avalon-framework-4.2.0.jar
      set CP=%CP%;%LIB%\xercesImpl.jar
      set CP=%CP%;%LIB%\commons-logging-1.1.1.jar
      set CP=%CP%;%LIB%\commons-io-1.3.1.jar
      set CP=%CP%;%LIB%\xmlgraphics-commons-1.5.jar
      set CP=%CP%;%LIB%\xml-apis.jar
      set CMD=java -cp "%CP%" org.apache.fop.fonts.apps.TTFReader
      set FONT_DIR=C:\Windows\Fonts
      %CMD% %FONT_DIR%\Arialuni.ttf Arialuni.xml
      ```

      The paths specified in the file are relative to the Oxygen XML Author installation directory. If you decide to create it in other directory, change the file paths accordingly.

      The *FONT_DIR* can be different on your system. Check that it points to the correct font directory. If the Java executable is not in the *PATH*, specify the full path of the executable.

      If the font has bold and italic variants, convert them too by adding two more lines to the script file:

      • for 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 OS X, execute the command `sh ttfConvert.sh` from the command line. On Windows, run the command `ttfConvert.bat` from the command line or double click on the file `ttfConvert.bat`.

3. Register the font in FOP configuration. (not necessary in case of DITA PDF transformations, see next step)

a) Create a FOP configuration file that specifies the font metrics file for your font.

```
<fop version="1.0">
  <base>./</base>
  <font-base>file:/C:/path/to/FOP/font/metrics/files/</font-base>
  <source-resolution>72</source-resolution>
  <target-resolution>72</target-resolution>
  <default-page-settings height="11in" width="8.26in"/>
  <renderers>
    <renderer mime="application/pdf">
      <filterList>
        <value>flate</value>
      </filterList>
      <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. 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 the FOP website.

b) Open the preferences dialog (**Options** > **Preferences**), go to **XML** > **XSLT/FO/XQuery** > **FO Processors**, and enter the path of the FOP configuration file in the **Configuration file for the built-in FOP** text field.

4. Set the font on the document content.

This is usually done with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

For DocBook documents, you can start with the predefined scenario called **DocBook PDF**, *edit the XSLT parameters*, and set the font name (in our example **Arialuni**) to the parameters body.font.family and title.font.family.
For TEI documents, you can start with the predefined scenario called **TEI PDF**, *edit the XSLT parameters*, and set the font name (in our example **Arialuni**) to the parameters bodyFont and sansFont.
For DITA to PDF transformations using DITA-OT modify the following two files:

- [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml - the `font-face` element included in each element `physical-font` having the `attribute char-set="default"` must contain the name of the font (*Arialuni* in our example)

- [OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/fop/conf/fop.xconf
  - an element `font` must be inserted in the element `fonts` which is inside the element `renderer` having the
  attribute `mime="application/pdf"`:

```
<renderer mime="application/pdf">
  . . .
   <fonts>
       <font metrics-url="Arialuni.xml" kerning="yes"
           embed-url="file:/Library/Fonts/Arialuni.ttf">
           <font-triplet name="Arialuni" style="normal"
               weight="normal"/>
       </font>
   </fonts>
  . . .
</renderer>
```

Adding Libraries to the Built-in FOP

You can extend the functionality of the built-in FO processor by dropping additional libraries in the
[OXYGEN_DIR]/lib/fop directory.

Hyphenation
To add support for hyphenation:

1. download the pre-compiled JAR from *OFFO* ;
2. place the JAR in [OXYGEN_DIR]/lib/fop;
3. restart the Oxygen XML Author.

Chapter

12

Querying Documents

Topics:

- *Running XPath Expressions*
- *Working with XQuery*

This chapter shows how to query XML documents in Oxygen XML Author with XPath expressions and the XQuery language.

Running XPath Expressions

This section covers the views, toolbars, and dialogs in Oxygen XML Author, dedicated to running XPath expressions.

What is XPath

XPath is a language for addressing specific parts of an XML document. XPath, like the Document Object Model (DOM), models an XML document as a tree of nodes. An XPath expression is a mechanism for navigating through and selecting nodes from the XML document. An XPath expression is, in a way, analogous to an SQL query used to select records from a database.

There are different types of nodes, including element nodes, attribute nodes and text nodes. XPath defines a way to compute a string-value for each type of node.

XPath defines a library of standard functions for working with strings, numbers and boolean expressions.

> - `child::*` - Selects all children of the root node.
> - `.//name` - Selects all elements having the name "name", descendants of the current node.
> - `/catalog/cd[price>10.80]` - Selects all the `cd` elements that have a price element with a value larger than 10.80.

To find out more about XPath, go to *http://www.w3.org/TR/xpath*.

Oxygen XPath Toolbar

XPath is a query language for selecting nodes from an XML document. To use XPath expressions effectively, you need a good understanding of *the XPath Core Function Library*.

The XPath Toolbar

Oxygen XML Author provides an XPath toolbar to let you query XML documents fast and easy using XPath expressions.

XPath 2.0 ▾ | ▾ **/personnel/person/name** ▾ ⚙▾

Figure 175: The XPath Toolbar

You can choose the XPath version from the drop-down menu available in the left side of the toolbar. Available options include XPath 1.0, XPath 2.0, XPath 2.0 SA, XPath 3.0, XPath 3.0 SA.

> **Note:** XPath 2.0 SA and XPath 3.0 SA use the Saxon EE *XML Schema version* option.

> **Note:** The results returned by XPath 2.0 SA and XPath 3.0 SA have a location limited to the line number of the start element (there are no column information and no end specified).

> **Note:** Oxygen XML Author uses Saxon to execute XPath 3.0 expressions, but implements a part of the 3.0 functions. When using a function that is not implemented, Oxygen XML Author can return a compilation error.

XPath scope menu - Oxygen XML Author allows you to define a scope on which the XPath operation will be executed. You can choose where the XPath expression will be executed:

- **Current file** - current selected file only.
- **Project** - all the files in the project.
- **Selected project resources** - the files selected in the project.
- **All opened files** - all files opened in the application.
- **Current DITA Map hierarchy** - all resources referred in the DITA maps opened in the DITA Maps Manager view.

- ▦ **Opened archive** - files open in the *Archive Browser* view.
- ▥ **Working sets** - selected working sets.

At the bottom of the scope menu there are available the following scope configuration actions:

- 🔍 **Configure XPath working sets** - allows you to configure and manage collections of files and folders, encapsulated in logical containers called *working sets*;
- ⇸ **XPath file filter** - you can filter the files from the selected scope on which the XPath expression will be executed. By default the XPath expression will be executed only on XML files, but you can also define a set of patterns that will filter out files from the current scope. If you select the **Include archive** option, the XPath expression will be also executed on the files in any archive (including EPUB and DocX) found at the current scope.

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

- ⬑ **Update on caret move** - when enabled and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed;
- ▶ **Evaluate as you type** - when you select this option, the XPath expression you are composing is evaluated in real time;

  > 📄 **Note:** The ▶ **Evaluate as you type** option and the automatic validation are disabled when you edit *huge documents* or when the scope is other than **Current file**.

- ⚙ **Options** - opens the Preferences page of the currently selected processing engine.

> 📄 **Note:** During the execution of an XPath expression, the XPath toolbar displays a stop button ■. Press this button to stop the XPath execution.

When you type expressions longer than 60 characters, a dialog pops up and offers you the possibility to switch to the *XPath builder* view.

The XPath/XQuery Builder View

The **XPath/XQuery Builder** view allows you to compose complex XPath and XQuery expressions and execute them over the currently edited XML document. For XPath 2.0 / 3.0, or XQuery expressions, you are able to use the doc() function to specify the source file over which the expressions are executed. When you connect to a database, the expressions are executed over that database. If you are using the **XPath/XQuery Builder** view and the current file is an XSLT document, Oxygen XML Author executes the expressions over the XML document in the associated scenario.

To open the **XPath/XQuery Builder** view, go to **Window** > **Show View** > **XPath/XQuery Builder**.

The upper part of the view contains the following actions:

- a drop-down list that allows you to select the type of the expression you want to execute. You can choose between:
  - XPath 1.0 (Xerces-driven);
  - XPath 2.0, XPath 2.0SA, XPath 3.0, XPath 3.0SA, XQuery 1.0, XQuery 3.0, Saxon-HE XQuery, Saxon-PE XQuery, or Saxon-EE XQuery (all of them are Saxon-driven);
  - custom connection to XML databases that can execute XQuery expressions.

    > 📄 **Note:** The results returned by XPath 2.0 SA and XPath 3.0 SA have a location limited to the line number of the start element (there are no column information and no end specified).

    > 📄 **Note:** Oxygen XML Author uses Saxon to execute XPath 3.0 expressions. Because Saxon implements a part of the 3.0 functions, when using a function that is not implemented, Oxygen XML Author returns a compilation error.

- ▶ **Execute XPath** button - press this button to start the execution of the XPath or XQuery expression you are editing. The result of the execution is displayed in the *Results view* in a separate tab;

- **Favorites** button - allows you to save certain expressions that you can later reuse. To add an expression as favorite, press the star button and enter a name under which the expression is saved. The star turns yellow to confirm that the expression was saved. Expand the drop-down list next to the star button to see all your favorites. Oxygen XML Author automatically groups favorites in folders named after the method of execution;

- **History** drop-down box - keeps a list of the last 15 executed XPath or XQuery expressions. Use the **Clear history** action from the bottom of the list to remove them.

- **Settings** drop-down menu - contains three options:

  - **Update on caret move** - when enabled and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed;

  - **Evaluate as you type** - when you select this option, the XPath expression you are composing is evaluated in real time;

    > **Note:** The **Evaluate as you type** option and the automatic validation are disabled when you edit *huge documents* or when the scope is other than **Current file**.

  - **Options** - opens the Preferences page of the currently selected processing engine.

- **XPath scope** menu - Oxygen XML Author allows you to define a scope on which the XPath operation will be executed. You can choose where the XPath expression will be executed:

  - **Current file** - current selected file only.
  - **Project** - all the files in the project.
  - **Selected project resources** - the files selected in the project.
  - **All opened files** - all files opened in the application.
  - **Current DITA Map hierarchy** - all resources referred in the DITA maps opened in the DITA Maps Manager view.
  - **Opened archive** - files open in the *Archive Browser* view.
  - **Working sets** - selected working sets.

  At the bottom of the scope menu there are available the following scope configuration actions:

  - **Configure XPath working sets** - allows you to configure and manage collections of files and folders, encapsulated in logical containers called *working sets*;

  - **XPath file filter** - you can filter the files from the selected scope on which the XPath expression will be executed. By default the XPath expression will be executed only on XML files, but you can also define a set of patterns that will filter out files from the current scope. If you select the **Include archive** option, the XPath expression will be also executed on the files in any archive (including EPUB and DocX) found at the current scope.

Figure 176: The XPath/XQuery Builder View

While you edit an XPath or XQuery expression, Oxygen XML Author assists you with the following features:

- Content Completion Assistant - It offers context-dependent proposals and takes into account the cursor position in the document you are editing. The set of functions proposed by the **Content Completion Assistant** also depends on the engine version. Select the engine version from the drop-down menu available in the toolbar.

- Syntax highlight - allows you to identify the components of an expression. To customize the colors of the components of the expression, *open the Preferences dialog* and go to *Editor > Colors*.

- automatic validation of the expression as you type;

> **Note:** When you type invalid syntax a red serrated line underlines the invalid fragments.

- function signature and documentation balloon, when the cursor is located inside a function.

The *usual edit actions* **Cut**, **Copy**, **Paste**, **Select All**, **Undo**, **Redo** are available in the pop-up menu of the top editable part of the view.

XPath Results View

When you run an XPath expression, Oxygen XML Author displays the results of its execution in the *Results View*. This view contains five columns:

- **Description** - Holds the result thatOxygen XML Author displays when you run an XPath expression.
- **XPath location** - Holds the path to the matched node.
- **Resource** - Holds the name of the document on which you run the XPath expression.
- **System ID** - Holds the path to the document itself.
- **Location** - Holds the location of the result in the document.

To arrange the results depending on a column click on its header. To group the results by their resource, or by their system id, right click the header of any column in the results view and select **Group by "Resource"** or **Group by "System ID"**. If no information regarding location is available, Oxygen XML Author displays **Not available** in the **Location** column. Oxygen XML Author displays the results in a valid XPath expression format.

```
- /node[value]/node[value]/node[value] -
```


Figure 177: XPath results highlighted in editor panel with character precision

> The following snippets are taken from a DocBook book based on the DocBook XML DTD. The book contains a number of chapters. To return all the chapter nodes of the book, enter `//chapter` in the XPath expression field and press **(Enter)**. This action returns all the `chapter` nodes of the DocBook book in the **Results View**. Click a record in the **Results View** to locate and highlight its corresponding chapter element and all its children nodes in the document you are editing.
>
> To find all `example` nodes contained in the `sect2` nodes of a DocBook XML document, use the following XPath expression: `//chapter/sect1/sect2/example`. Oxygen XML Author adds a result in the **Results View** for each `example` node found in any `sect2` node.
>
> For example, if the result of the above XPath expression is:
>
> ```
> - /chapter[1]/sect1[3]/sect2[7]/example[1]
> ```
>
> it means that in the edited file the `example` node is located in the first chapter, third section level one, seventh section level 2.

Catalogs

The evaluation of the XPath expression tries to resolve the locations of documents referred in the expression through the *XML catalogs*. These catalogs are *configured in the Preferences* pages and *the current XInclude preferences*.

Let's take as an example the evaluation of the `collection(URIofCollection)` function (XPath 2.0). To resolve the references from the files returned by the `collection()` function with an XML catalog, specify the class name of the XML catalog enabled parser for parsing these collection files. The class name is `ro.sync.xml.parser.CatalogEnabledXMLReader`. Specify it as it follows:

```
let $docs := collection(iri-to-uri(
    "file:///D:/temp/test/XQuery-catalog/mydocsdir?recurse=yes;select=*.xml;
    parser=ro.sync.xml.parser.CatalogEnabledXMLReader"))
```

XPath Prefix Mapping

To define default mappings between prefixes (that you can use in the XPath toolbar) and namespace URIs *go to* ⚙≡ *XPath Options preferences panel* and enter the mappings in the **Default prefix-namespace mappings** table. The same preferences panel allows you to configure the default namespace used in XPath 2.0 expressions.

⚠ **Important:** If you define a default namespace, Oxygen XML Author binds this namespace to the first free prefix from the list: `default`, `default1`, `default2`, and so on. For example, if you define the default namespace `xmlns="something"` and the prefix `default` is not associated with another namespace, you can match tags without prefix in an XPath expression typed in the XPath toolbar by using the prefix `default`. To find all the `level` elements when you define a default namespace in the root element, execute this expression: `//default:level` in the XPath toolbar.

Working with XQuery

This section explains how to edit and run XQuery queries in Oxygen XML Author.

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.

Transforming XML Documents Using XQuery

XQueries are similar with the XSL stylesheets, both being capable of transforming an XML input into another format. You specify the input URL when you *define the transformation scenario*. The result can be saved and opened in the associated application. You can even run a *FO processor* on the output of an XQuery. The transformation scenarios may be shared between many XQuery files, are *exported* together with the XSLT scenarios and can be managed in *the Configure Transformation Scenario dialog* ,or in *the Scenarios view*. The transformation can be performed on the XML document specified in the **XML URL** field, or, if this field is empty, the documents referred from the query expression. The parameters of XQuery transforms must be set in *the Parameters dialog*. Parameters that are in a namespace must be specified using the qualified name, for example a `param` parameter in the *http://www.oxygenxml.com/ns* namespace must be set with the name `{http://www.oxygenxml.com/ns}param`.

The transformation uses one of the Saxon 9.5.1.7 HE, Saxon 9.5.1.7 PE, Saxon 9.5.1.7 EE processors, a database connection or any XQuery processor that provides an XQJ API implementation.

The Saxon 9.5.1.7 EE processor supports also XQuery 3.0 transformations.

Note: In case the XQuery 3.0 support is active, the XQuery Update support is no longer available.

XQJ Transformers

This section describes the necessary procedures before running an XQJ transformation.

How to Configure an XQJ Data Source

Any transformer that offers an XQJ API implementation can be used when validating XQuery or transforming XML documents. (An example of an XQuery engine that implements the XQJ API is *Zorba*.

1. In case your XQJ Implementation is native, make sure the directory containing the native libraries of the engine is added to your system environment variables: to PATH - on Windows, to LD_LIBRARY_PATH - on Linux, or to DYLD_LIBRARY_PATH - on OS X. Restart Oxygen XML Author after configuring the environment variables.
2. *Open the Preferences dialog* and go to **Data Sources**.
3. Click the **New** button in the **Data Sources** panel.
4. Enter a unique name for the data source.
5. Select **XQuery API for Java(XQJ)** in the **Type** combo box.
6. Press the **Add** button to add XQJ API-specific files.

   You can manage the driver files using the **Add**, **Remove**, **Detect**, and **Stop** buttons.

   Oxygen XML Author detects any implementation of `javax.xml.xquery.XQDataSource` and presents it in **Driver class** field.
7. Select the most suited driver in the **Driver class** combo box.
8. Click the **OK** button to finish the data source configuration.

How to Configure an XQJ Connection

The steps for configuring an XQJ connection are the following:

1. *Open the Preferences dialog* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for this connection.
4. Select one of the previously configured *XQJ data sources* in the **Data Source** combo box.
5. Fill-in the connection details.

   The properties presented in the connection details table are automatically detected depending on the selected data source.
6. Click the **OK** button.

Display Result in Sequence View

The result of an XQuery executed on a database can be very large and sometimes only a part of the full result is needed. To avoid the long time necessary for fetching the full result, select the **Present as a sequence** option in the **Output** tab of the **Edit scenario** dialog. This option fetches only the first chunk of the result. Clicking the **More results available** label that is displayed at the bottom of the **Sequence** view fetches the next chunk of results.
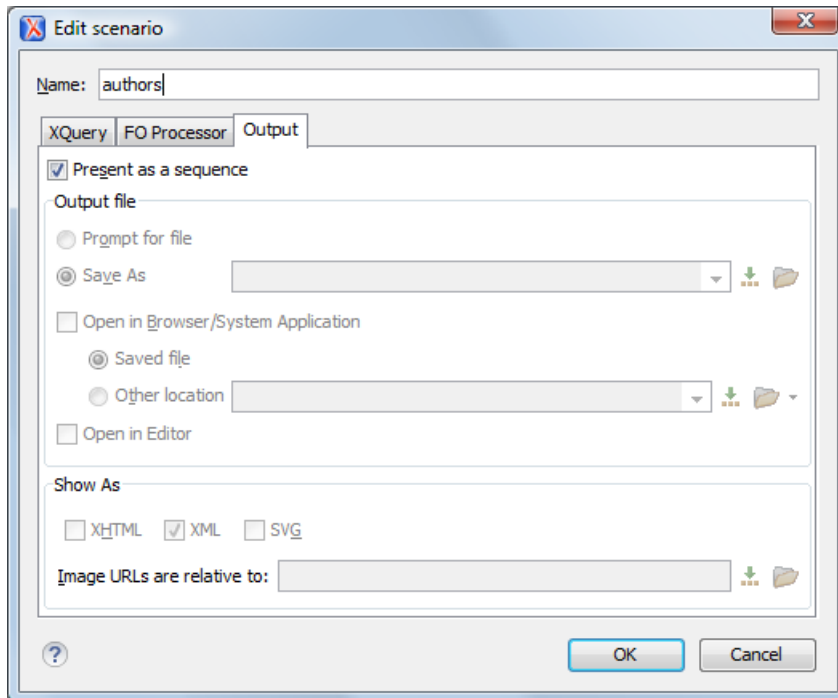
Figure 178: The XQuery transformation result displayed in Sequence view

A chunk of the XQuery transformation result is displayed in the **Sequence** view.
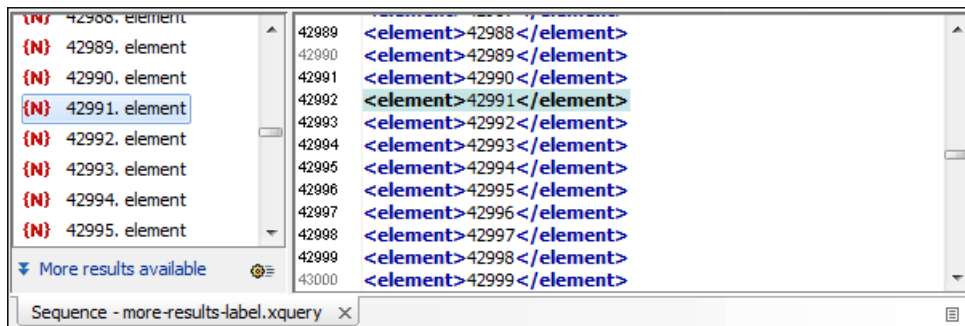
Figure 179: The XQuery transformation result displayed in Sequence view

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. The advanced options specific for Saxon HE / PE / EE are:

- **Use a configuration file** - when enabled, the specified Saxon configuration file are used for determining the Saxon advanced options;
- **Recoverable errors** - policy for handling recoverable errors in the stylesheet. Allows the user to choose how dynamic errors are handled. Either one of the following options can be selected:
  - recover silently;
  - recover with warnings;
  - signal the error and do not attempt recovery.

- **Strip whitespaces** - can have one of the following values:
  - **All** - strips all whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the XML source document;

- **Ignorable** - strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the XML source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content;
- **None** - strips no whitespace before further processing.

- **Optimization level** - allows optimization to be suppressed in cases where reducing compile time is important, or where optimization gets in the way of debugging, or causes extension functions with side-effects to behave unpredictably;
- **Allow calls on extension functions** - when enabled, calling external Java functions is allowed;
- **Validation of the source file** - available only for Saxon EE. It can have three values:

  - **Schema validation** - this mode requires an XML Schema and enables parsing the source documents with schema-validation enabled;
  - **Lax schema validation** - if an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled;
  - **Disable schema validation** - this mode means parsing the source documents with schema validation disabled.

- **Validation errors in the results tree treated as warnings** - available only for Saxon EE. If checked, all validation errors are treated as warnings, otherwise they are treated as fatal;
- **Generate bytecode ("--generateByteCode:(on|off)")** - When you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such treatment. For further details regarding this option, go to *http://www.saxonica.com/documentation/javadoc/*.
- **Enable XQuery 3.0 support ("-qversion:(1.0|3.0)")** - if checked, Saxon EE runs the XQuery transformation with the XQuery 3.0 support;

  > **Note:** In case the XQuery 3.0 support is active, the XQuery Update support is no longer available.

- **Backup files updated by XQuery ("-backup:(on|off)")** - if checked, backup versions for any XML files updated with XQuery Update are generated.

Updating XML Documents using XQuery

Using the bundled Saxon 9.5.1.7 EE XQuery processor Oxygen XML Author offers support for XQuery Update 1.0. The XQuery Update Facility provides expressions that can be used to make persistent changes to instances of the XQuery 1.0 and XPath 2.0 Data Model. Thus, besides querying XML documents, you can modify them using the various insert/delete/modify/create methods available in the *XQuery Update 1.0* standard.

Choose Saxon 9.5.1.7 EE as a transformer in the scenario associated with the XQuery files containing update statements and Oxygen XML Author 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

13

Working with Archives

Oxygen XML Author offers the means to manipulate files directly from ZIP type archives. By manipulation one should understand opening and saving files directly in archives, browsing and modifying archive structures. The archive support is available for all ZIP-type archives, which includes:

- ZIP archives
- EPUB books
- JAR archives
- Office Open XML (OOXML) files
- Open Document Format (ODF) files
- IDML files

This means that you can modify, transform, validate files directly from OOXML or ODF packages. The structure and content of an EPUB book, OOXML file or ODF file *can be opened, edited and saved* as for any other ZIP archive.

You can transform, validate and perform many other operations on files directly from an archive. When selecting an URL for a specific operation like transformation or validation you can click the **Browse for archived file** button to navigate and choose the file from a certain archive.

Browsing and Modifying Archive Structure

You can open an archive in the **Archives Browser** view doing one of the following:

- Open an archive from the *Project view*;
- Choose an archive in the Oxygen XML Author file chooser dialog;
- Drag an archive from the file explorer and drop it in the **Archives Browser** view.

When displaying an archive, the **Archive Browser** view locks the archive file. It is then automatically unlocked when the **Archive Browser** view is closed.

> ⚠ **Important:** If a file is not recognized by Oxygen XML Author as a supported archive type, you can add it from the *Archive preferences page*.
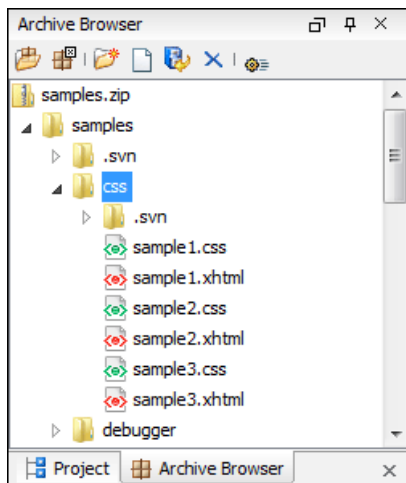
Figure 180: Browsing an Archive

The following operations are available on the **Archive Browser** toolbar:

Reopen

You can use this drop-down to reopen recently edited archives. Apart from the history of the recently edited archives, the drop-down also contains the **Clear history** and **Open Archive** actions.

Open Archive menu

Provides access to the **Open Archive...** action which opens a new archive in the browser. If the extension is not known as an archive extension, you will be directed to the *Archive preferences page* to add a new extension. The sub-menu keeps a list of recently open archive files and a **Clear history** action which allows you to delete the list.

Close

Closes the browsed archive and unlocks the archive file.

New folder...

Creates a 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.

Extract...

Extracts a resource from the archive in a specified folder.

📂 New folder...

Creates a folder as child of the selected folder in the browsed archive.

📄 New file...

Creates a file as child of the selected folder in the browsed archive.

Add files...

Adds existing files as children of the selected folder in the browsed archive.

> 📄 **Note:** On OS X, there is also available the **Add file...** action, which allows you to add one file at a time.

Rename

Renames a resource in the archive.

🔍 Find/Replace in Files

Allows you to search for and replace specific pieces of text inside the archive.

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.

Preview

Previews an image contained in the archive See the *Image Preview* section for more details.

Copy location

Copies the URL location of the selected resource.

🔄 Refresh

Refreshes the selected resource.

Properties

Views properties for the selected resource.

Working with EPUB

EPUB is a free and open electronic book standard by the International Digital Publishing Forum (IDPF). It was designed for *reflowable content*, meaning that the text display can be optimized for the particular display device used by the reader of the EPUB-formatted book. Oxygen XML Author supports both EPUB 2.0 and EPUB 3.0.

EPUB files are opened in the **Archive Browser** view, exposing all their internal components:

- document content (XHTML and image files);
- packaging files;
- container files.
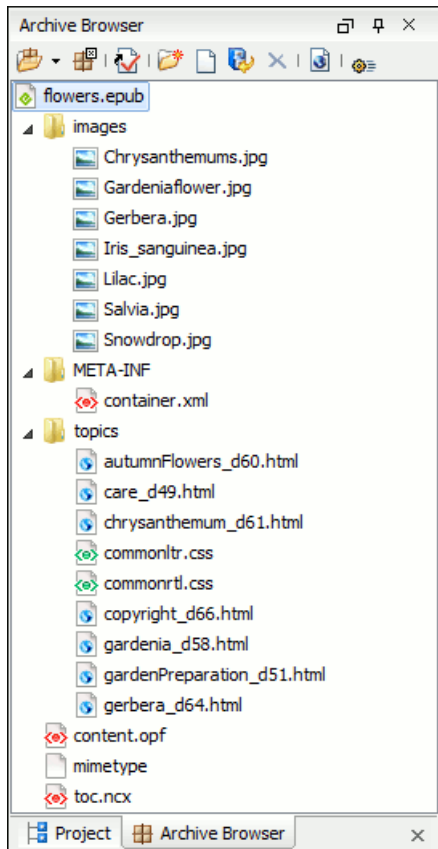
Figure 181: EPUB file displayed in the Archive Browser view

Here you can edit, delete and add files that compose the EPUB structure. To check that the EPUB file you are editing is valid, invoke the ☑ **Validate and Check for Completeness** action. Oxygen XML Author uses the open-source EpubCheck validator to perform the validation. This validator detects many types of errors, including OCF container structure, OPF and OPS mark-up, as well as internal reference consistency. All errors found during validation are displayed in a separate tab in the **Errors** view.

> **Note:** Invoke the 📘 **Open in System Application** action to see how the EPUB is rendered in your system default EPUB reader application.

> **Note:** All changes made to the structure of an EPUB, or to the contents of the files inside an EPUB are immediately saved.

To watch our video demonstration about the EPUB support in Oxygen XML Author, go to *http://oxygenxml.com/demo/Epub.html*.

Create an EPUB

To begin writing an EPUB file from scratch, do the following:

1. Go to **File** > **New** , press <u>**Ctrl+N (Command+N on OS X)**</u> on your keyboard. or click **New** on the main toolbar.

2. Choose **EPUB Book** template. Click **Create**. Choose the name and location of the file. Click **Save**.
   A skeleton EPUB file is saved on disk and open in the **Archive Browser** view.

3. Use the **Archive Browser** view specific actions to edit, add and remove resources from the archive.

4. Use the **Validate and Check for Completeness** action to verify the integrity of the EPUB archive.

Publish to EPUB

Oxygen XML Author comes with built-in support for publishing DocBook and DITA XML documents directly to EPUB.

1. Open the **Configure Transformation Scenario(s)** dialog box and select a predefined transformation scenario. To publish from DITA, select the **DITA Map EPUB** transformation scenario. To publish from DocBook select the **DocBook EPUB** transformation scenario.

2. Click **Apply associated** to run the transformation scenario.

Editing Files From Archives

You can open and edit files directly from an archive using the **Archive Browser** view. When saving the file back to archive, you are prompted to choose if you want the application to make a backup copy of the archive before saving the new content. If you choose **Never ask me again**, you will not be asked again to make backup copies. You can re-enable the dialog pop-up from the *Messages preferences page*.

> **Note:** All changes made to the structure of an archive, or to the contents of the files inside an archive are immediately saved.

Chapter

14

Working with Databases

Topics:

- *Relational Database Support*
- *Native XML Database (NXD) Support*
- *WebDAV Connection*
- *BaseX Support*

XML is a storage and interchange format for structured data and it is supported by all major database systems. Oxygen XML Author 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 Author offers support for the following relational databases: IBM DB2, JDBC-ODBC Bridge, MySQL, Microsoft SQL Server, Oracle 11g:

- browsing the tables of these types of database in the **Data Source Explorer** view
- executing SQL queries against them
- calling stored procedures with input and output parameters

To watch our video demonstration about the integration between the relational databases and Oxygen XML Author, go to *http://www.oxygenxml.com/demo/Author_Database_Integration.html*.

Configuring Database Data Sources

This section describes the procedures for configuring the data sources for relational databases.

How to Configure an IBM DB2 Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to an IBM DB2 server are the following:

1. *Open the Preferences dialog* and go to **Data Sources**.
2. Click the ✚ **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.
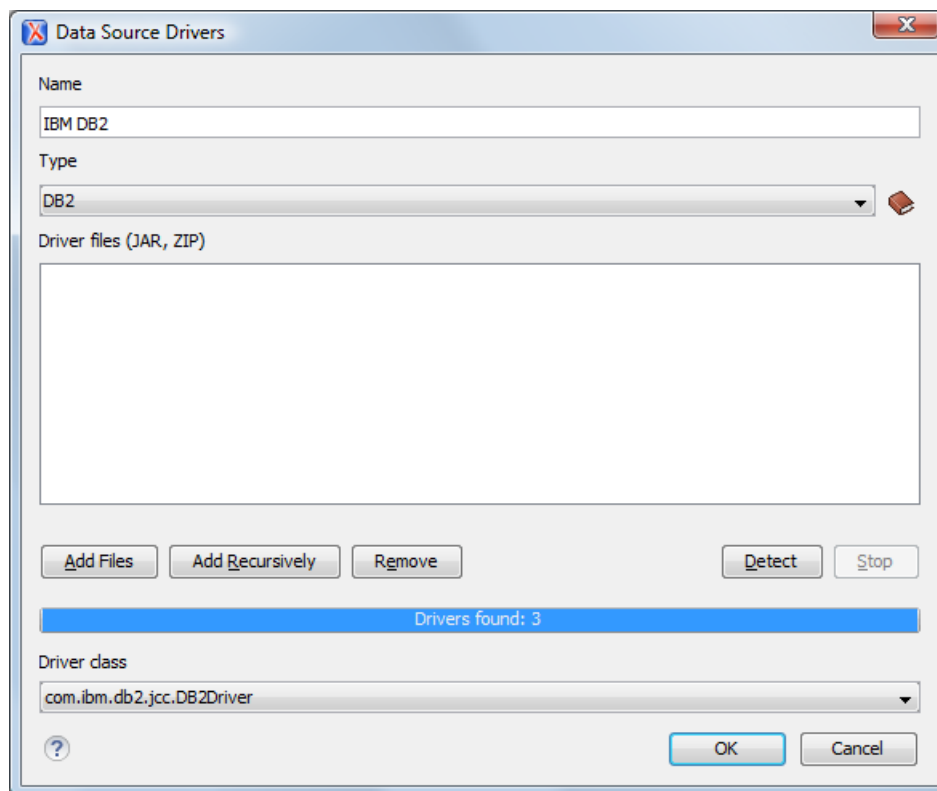
Figure 182: Data Source Drivers Configuration Dialog

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 Author.

6. Select the most suited **Driver class**.

7. Click the **OK** button to finish the data source configuration.

To watch our video demonstration about running XQuery against an IBM DB2 Pure XML database, go to *http://www.oxygenxml.com/demo/DB2.html*.

How to Configure a Microsoft SQL Server Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to a Microsoft SQL server are the following:

1. *Open the Preferences dialog* and go to **Data Sources**.

2. Click the ✚ **New** button in the **Data Sources** panel.

The dialog for configuring a data source will be opened.
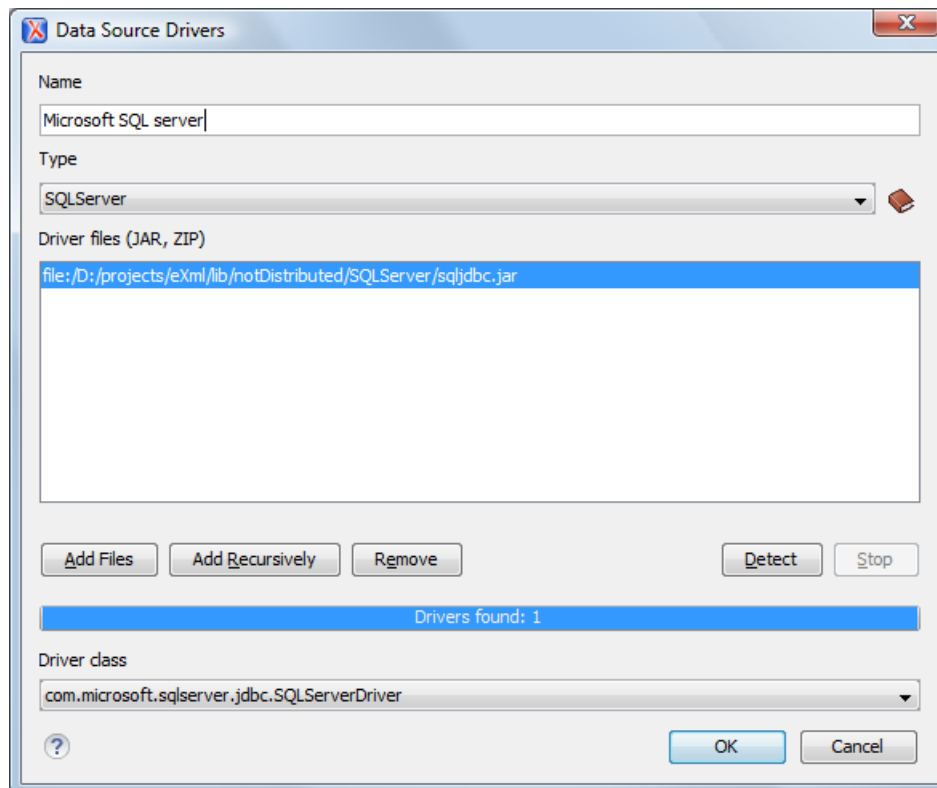
Figure 183: Data Source Drivers Configuration Dialog

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 Author.

6. Select the most suited **Driver class**.

7. Click the **OK** button to finish the data source configuration.

How to Configure a Generic JDBC Data Source

By default, Oxygen XML Author contains a generic JDBC data source called **JDBC-ODBC Bridge**. Oxygen XML Author 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. *Open the Preferences dialog* and go to **Data Sources**.

2. Click the ✚ **New** button in the **Data Sources** panel.

The following dialog is displayed:
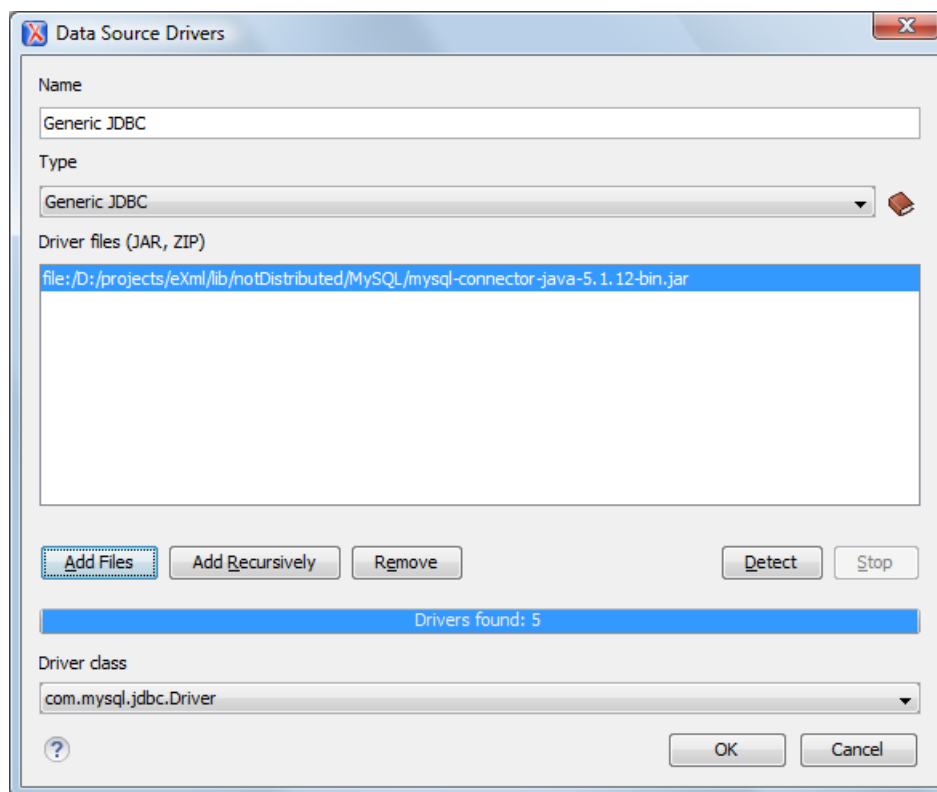
Figure 184: Data Source Drivers Configuration Dialog

3. Enter a unique name for the data source.

4. Select **Generic JDBC** in the driver type combo box.

5. Add the driver file(s) using the **Add** button.

6. Select the most suited **Driver class**.

7. Click the **OK** button to finish the data source configuration.

How to Configure a MySQL Data Source

Older versions of Oxygen XML Author (up to version 11.2) include a built-in type of data sources called **MySQL** based on the JDBC driver for the MySQL 4 server. That type of data source is still available but is marked *outdated* because it does not support more recent versions of the MySQL server (starting from version 5.0) and it will be removed in a

future version of Oxygen XML Author. To connect to a MySQL server, create a data source of type Generic JDBC based on *the MySQL JDBC driver available on the MySQL website*. The following steps describe how you can configure such a data source:

1. *Open the **Preferences** dialog* and go to **Data Sources**.

2. Click the ✚ **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.
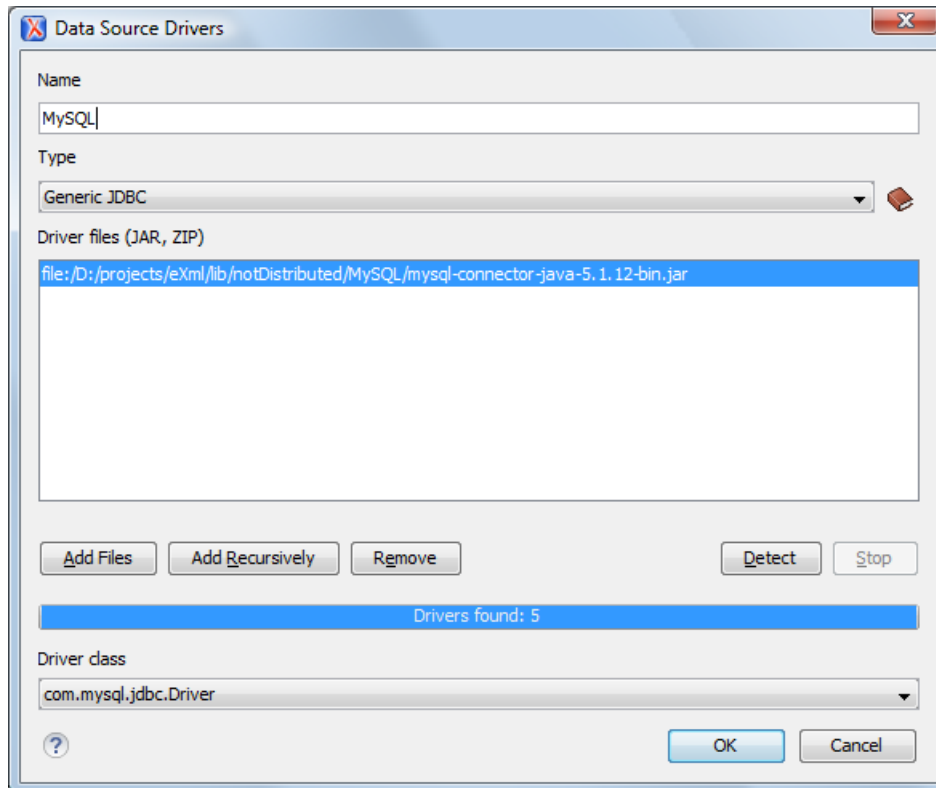
Figure 185: Data Source Drivers Configuration Dialog

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 Author.

6. Select the most suited **Driver class**.

7. Click the **OK** button to finish the data source configuration.

How to Configure an Oracle 11g Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to an Oracle 11g server are the following:

1. *Open the **Preferences** dialog* and go to **Data Sources**.

2. Click the ✚ **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.
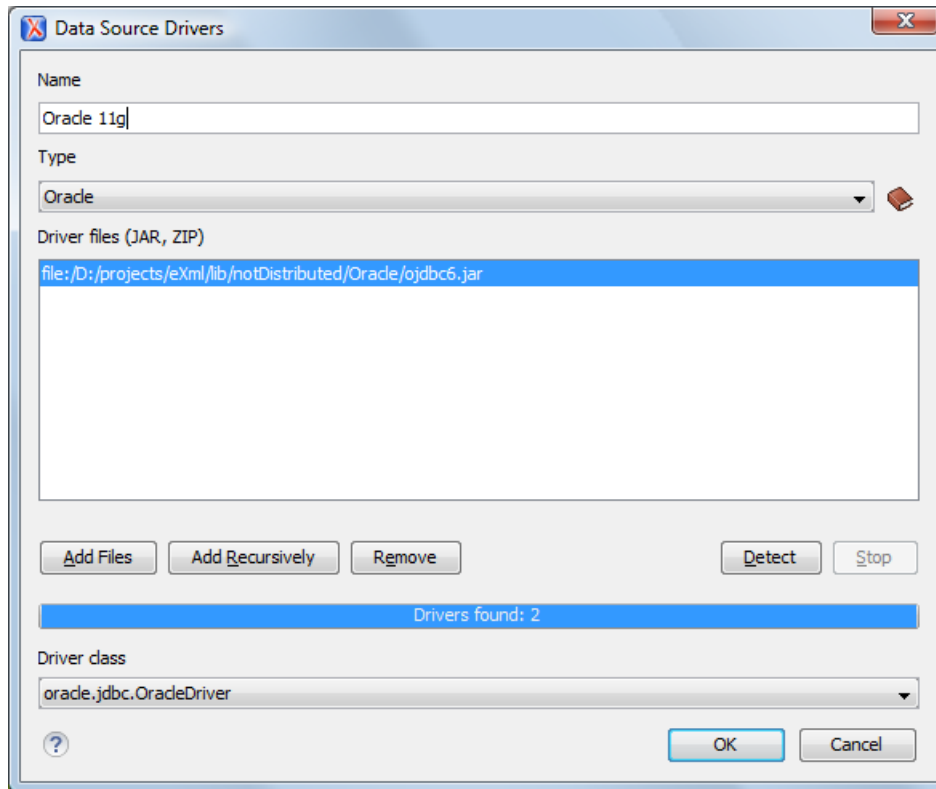
Figure 186: Data Source Drivers Configuration Dialog

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 Author.

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. *Open the **Preferences** dialog* and go to **Data Sources**.

2. Click the ✚ **New** button in the **Data Sources** panel.

   The dialog for configuring a data source will be opened.

3. Enter a unique name for the data source.

4. Select **PostgreSQL** in the driver type combo box.

5. Add the PostgreSQL driver file using the **Add** button.

   The PostgreSQL driver file is called `postgresql-8.3-603.jdbc3.jar`. In the *Download links for database drivers* section there are listed the URLs from where to download the drivers necessary for accessing PostgreSQL databases in Oxygen XML Author.

6. Select the most suited **Driver class**.

7. Click the **OK** button to finish the data source configuration.

Configuring Database Connections

This section describes the procedures for configuring the connections for relational databases:

How to Configure an IBM DB2 Connection

The support to create an IBM DB2 connection is available in the Enterprise edition only.

To configure a connection to an IBM DB2 server, follow these steps:

1. *Open the Preferences dialog* and go to **Data Sources**.
2. In the **Connections** panel, click the ✚ **New** button.

   The dialog for configuring a database connection will be displayed.
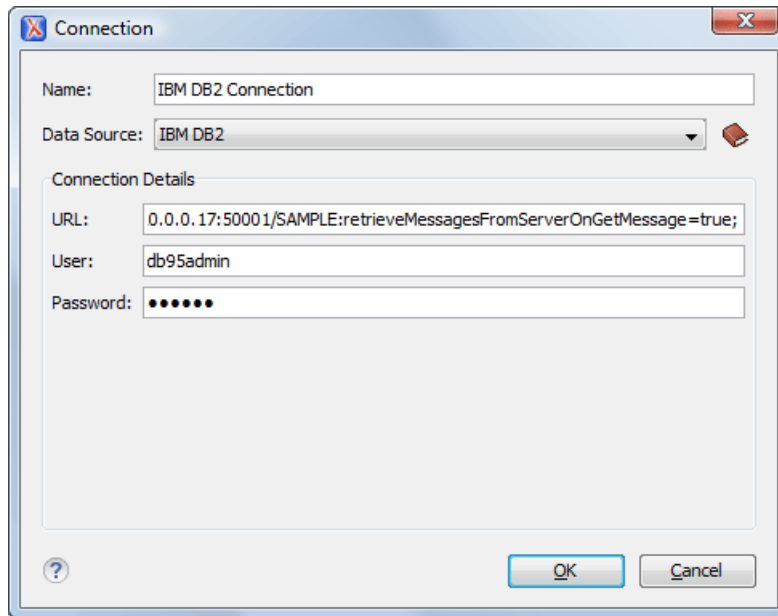
Figure 187: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select an IBM DB2 data sources in the **Data Source** combo box.
5. Fill-in the connection details.
   a) Fill-in the URL to the installed IBM DB2 engine.
   b) Fill-in the user name to access the IBM DB2 engine.
   c) Fill-in the password to access the IBM DB2 engine.
6. Click the **OK** button to finish the configuration of the database connection.

To watch our video demonstration about running XQuery against an IBM DB2 Pure XML database, go to *http://www.oxygenxml.com/demo/DB2.html*.

How to Configure a JDBC-ODBC Connection

To configure a connection to an ODBC data source, follow these steps:

1. *Open the Preferences dialog* and go to **Data Sources**.
2. In the **Connections** panel, click the ✚ **New** button.

   The dialog for configuring a database connection will be displayed.
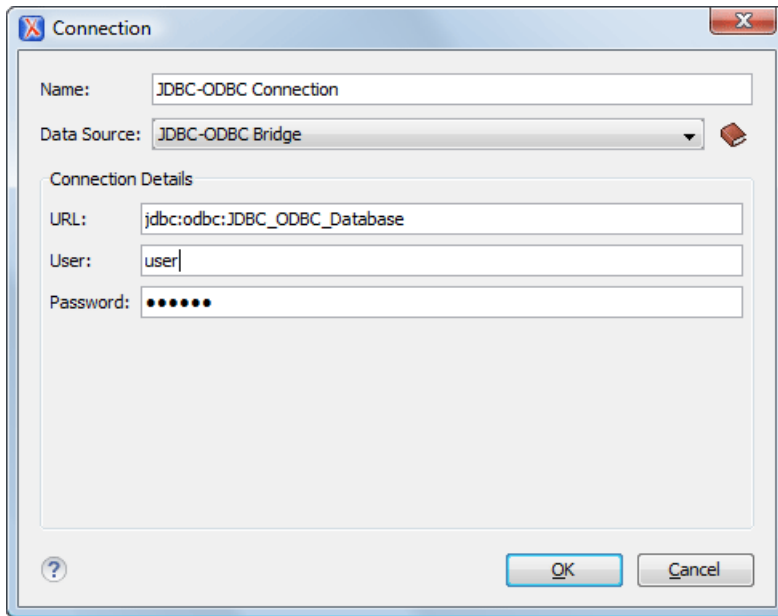
Figure 188: The Connection Configuration Dialog

3. Enter a unique name for the connection.
4. Select JDBC-ODBC bridge in the **Data Source** combo box.
5. Fill-in the connection details.
   a) Fill-in the URL of the ODBC source.
   b) Fill-in the user name of the ODBC source.
   c) Fill-in the password of the ODBC source.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure a Microsoft SQL Server Connection

The support to configure a Microsoft SQL Server Connection is available in the Enterprise edition only.

To configure a connection to a Microsoft SQL Server, follow these steps:

1. *Open the Preferences dialog* and go to **Data Sources**.
2. In the **Connections** panel, click the ✚ **New** button.

   The dialog for configuring a database connection will be displayed.
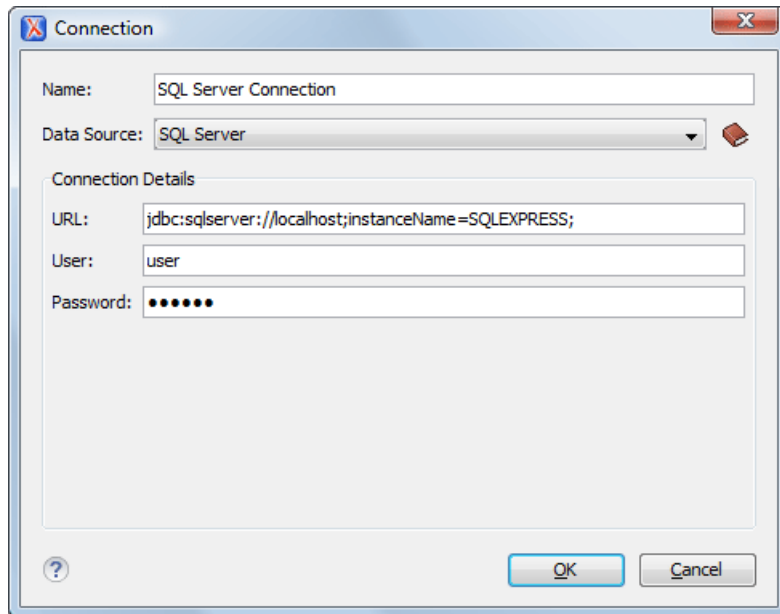
Figure 189: The Connection Configuration Dialog

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;instanceName=SQLEXPRESS;integratedSecurity=true;
```

> **Note:** For integrated authentication, leave the **User** and **Password** fields empty.

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

To configure a connection to a MySQL server, follow these steps:

1. *Open the Preferences dialog* and go to **Data Sources**.

2. In the **Connections** panel, click the ✚ **New** button.

The dialog for configuring a database connection will be displayed.
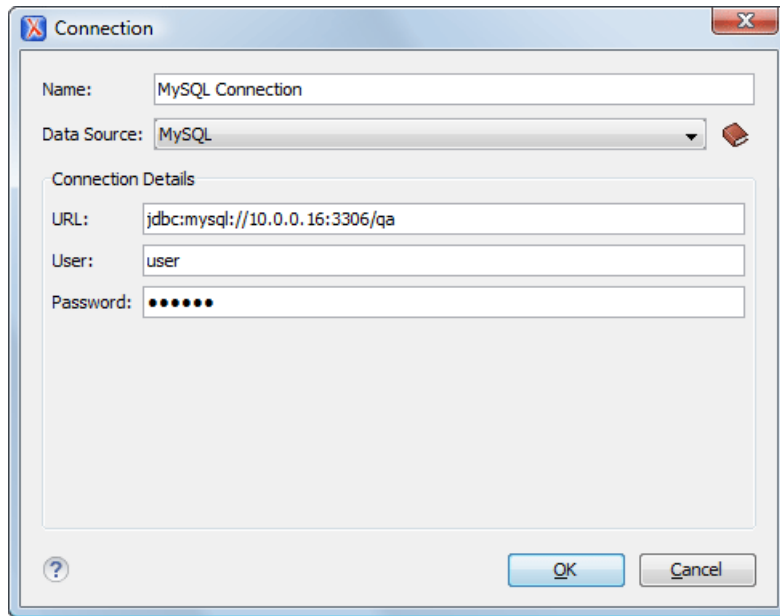
Figure 190: The Connection Configuration Dialog

3. Enter a unique name for the connection.

4. Select a MySQL data source in the **Data Source** combo box.

5. Fill-in the connection details.

   a) Fill-in the URL of the MySQL server.
   b) Fill-in the user name for the connection to the MySQL server.
   c) Fill-in the password for the connection to the MySQL server.

6. Click the **OK** button to finish the configuration of the database connection.

How to Configure a Generic JDBC Connection

To configure a connection to a generic JDBC database, follow these steps:

1. *Open the Preferences dialog* and go to **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. *Open the Preferences dialog* and go to **Data Sources**.

2. In the **Connections** panel, click the ✚ **New** button.

The dialog for configuring a database connection will be displayed.
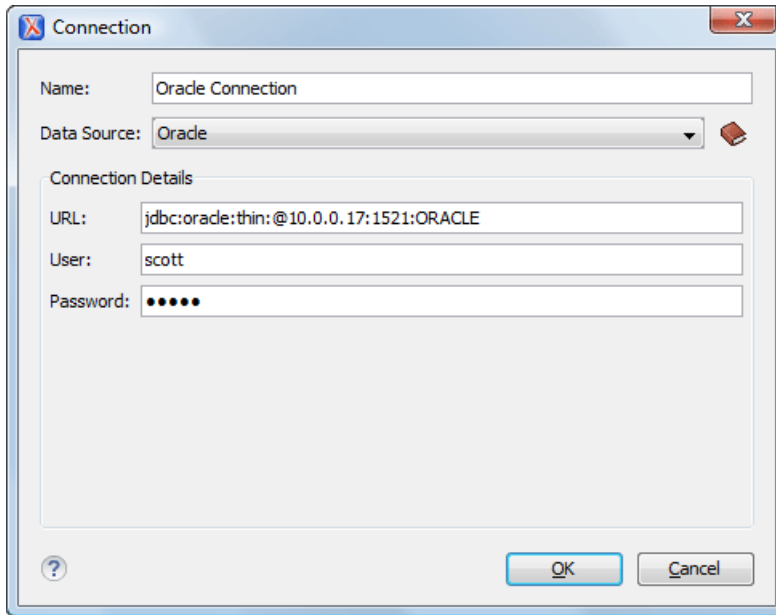
Figure 191: The Connection Configuration Dialog

3. Enter a unique name for the connection.

4. Select an Oracle 11g data source in the **Data Source** combo box.

5. Fill-in the connection details.

    a) Fill-in the URL of the Oracle server.

    b) Fill-in the user name for the connection to the Oracle server.

    c) Fill-in the password for the connection to the Oracle server.

6. Click the **OK** button to finish the configuration of the database connection.

How to Configure a PostgreSQL 8.3 Connection

The steps for configuring a connection to a PostgreSQL 8.3 server are the following:

1. *Open the Preferences dialog* and go to **Data Sources**.

2. In the **Connections** panel, click the ✚ **New** button.

The dialog for configuring a database connection will be displayed.
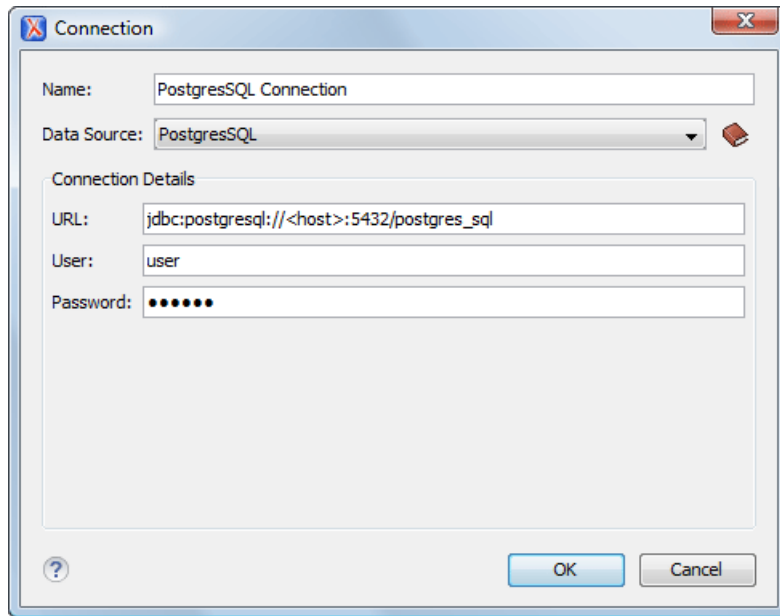
Figure 192: The Connection Configuration Dialog

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 shows your database connections. You can connect to a database simply by expanding the connection node. The database structure can be expanded up to column level. Oxygen XML Author supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.
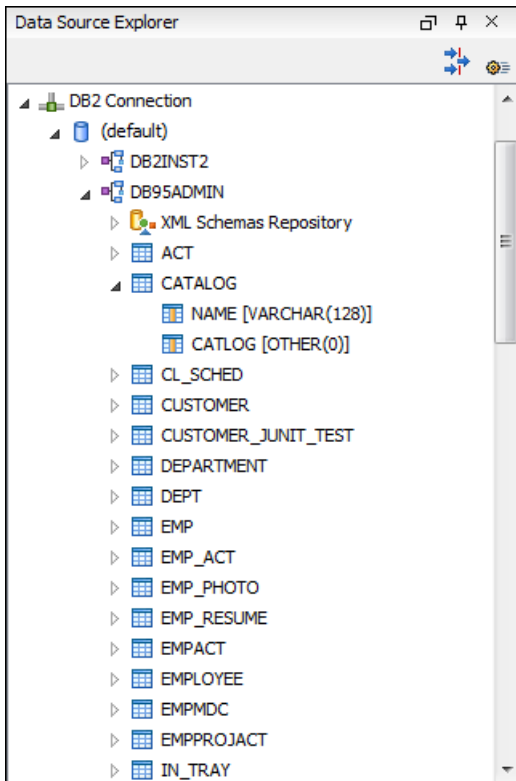
Figure 193: Data Source Explorer View

The following objects are displayed in the **Data Source Explorer** view:

- **Connection**;
- **Collection (Catalog)**;
- **XML Schema Repository**;
- **XML Schema Component**;
- **Schema**;
- **Table**;
- **System Table**;
- **Table Column**.

A **collection** (called *catalog* in some databases) is a hierarchical container for **resources** and further sub-collections. There are two types of resources:

- **XML resource** - an XML document or a document fragment, selected by a previously executed XPath query.
- **non XML resource**

Note: For some connections you can add or move resources into a container by dragging them from:

- **Project view**;
- the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example);
- or from another database container.

The following actions are available in the view's toolbar:

Filters

Opens the **Data Sources / Table Filters** *Preferences page*, allowing you to decide which table types will be displayed in the **Data Source Explorer** view.

Configure Database Sources

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

Actions Available at Connection Level in Data Source Explorer View

The contextual menu of a **Connection** node 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 .

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 Author.

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 its subtree).

Add

Adds a new schema to the XML Schema files.

Unregister
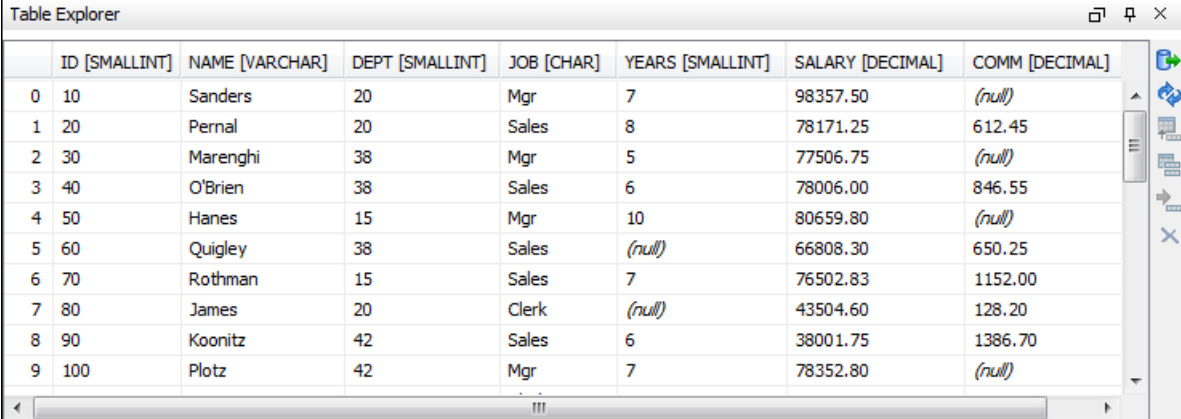
Removes the selected schema from the XML Schema Repository.

View

Opens the selected schema in Oxygen XML Author.

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 Author will try to update the database with the new cell content.

	ID [SMALLINT]	NAME [VARCHAR]	DEPT [SMALLINT]	JOB [CHAR]	YEARS [SMALLINT]	SALARY [DECIMAL]	COMM [DECIMAL]
0	10	Sanders	20	Mgr	7	98357.50	(null)
1	20	Pernal	20	Sales	8	78171.25	612.45
2	30	Marenghi	38	Mgr	5	77506.75	(null)
3	40	O'Brien	38	Sales	6	78006.00	846.55
4	50	Hanes	15	Mgr	10	80659.80	(null)
5	60	Quigley	38	Sales	(null)	66808.30	650.25
6	70	Rothman	15	Sales	7	76502.83	1152.00
7	80	James	20	Clerk	(null)	43504.60	128.20
8	90	Koonitz	42	Sales	6	38001.75	1386.70
9	100	Plotz	42	Mgr	7	78352.80	(null)

Figure 194: 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 Author's options is displayed in the **Table Explorer** view, a warning dialog will inform you that the table is only partially shown.

You will be notified if the value you have entered in a cell is not valid (and thus it cannot be updated).

- If the content of the edited cell does not belong to the data type of the column, the cell will be marked by a red square and it will remain in editing state until a correct value is inserted. For example, in the following figure `propID` contains `LONG` values. If a character or string was inserted, the cell will look like this:

Figure 195: Cell containing an invalid value.

- If the constraints of the database are not met (like primary key constraints for example), an Information dialog will appear, notifying you of the reason the database has not been updated. For example, if you'd try to set the primary key `propID` for the second record in the table to 10 also, you would get the following message:

Figure 196: 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 .

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 Author'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.

4. Select the type of statement from the popup menu.

   If you dragged a table depending on your choice, one of the following statements are inserted into the document:

   - SELECT `` `field1` ``,`` `field2` ``, ... FROM `` `catalog` ``.`` `table` `` (for this example: `SELECT `DEPT`,`DEPTNAME`,`LOCATION` FROM `camera`.`cameraDesc` `)
   - UPDATE`` `catalog` ``.`` `table` `` SET `` `field1` ``=,`` `field2` ``=,.... (for this example: `UPDATE `camera`.`cameraDesc` SET `DEPT`=, `DEPTNAME`=, `LOCATION`=)`

- INSERT INTO `catalog`.`table` (`field1`,`field2`, ...) VALUES (, ,) (for this example: `INSERT INTO` `camera`.`cameraDesc` (`DEPT`,`DEPTNAME`,`LOCATION`) VALUES (, ,))
- DELETE FROM `catalog`.`table` (for this example: `DELETE FROM` `camera`.`cameraDesc`)

If you dragged a column depending on your choice, one of the following statements are inserted into the document:

- SELECT `field` FROM `catalog`.`table` (for this example: `SELECT` `DEPT` `FROM` `camera`.`cameraDesc`)
- UPDATE `catalog`.`table` SET `field` = (for this example: `UPDATE` `camera`.`cameraDesc` `SET` `DEPT`=)
- INSERT INTO `catalog`.`table` (`field1`) VALUES () (for this example: `INSERT INTO` `camera`.`cameraDesc` (`DEPT`) VALUES ())
- DELETE FROM `catalog`.`table` (for this example: `DELETE FROM` `camera`.`cameraDesc` `WHERE` `DEPT`=)

SQL Validation

Currently, SQL validation support is offered for IBM DB2. Please note that if you choose a connection that doesn't support SQL validation you will receive a warning when trying to validate. The SQL document will be validated using the connection from the associated transformation scenario.

Executing SQL Statements

The steps for executing an SQL statement on a relational database are the following:

1. Configure a *transformation scenario* from the **Configure Transformation Scenario** button from the **Transformation** toolbar.

    A SQL transformation scenario needs a database connection. You can configure a connection from the **Preferences** button from the scenario dialog.

    The dialog that appears contains the list of existing scenarios that apply to SQL documents.

2. Set parameter values for SQL placeholders from the **Parameters** button from the scenario dialog.
    For example in `SELECT * FROM `test`.`department` where DEPT = ? or DEPTNAME = ?` two parameters can be configured for the place holders (?) in the transformation scenario.

    When the SQL statement will be executed, the first placeholder will be replaced with the value set for the first parameter in the scenario, the second placeholder will be replaced by the second parameter value and so on.

    > ⊘ **Restriction:** When a stored procedure is called in an SQL statement executed on an SQL Server database mixing in-line parameter values with values specified using the **Parameters** button of the scenario dialog is not recommended. It is due to a limitation of the SQL Server driver for Java applications. An example of stored procedure call that is not recommended is: `call dbo.Test(22, ?)`.

3. Execute the SQL scenario from the **Transform now** button of the scenario dialog.

    The result of a SQL transformation will be *displayed in a view* at the bottom of the Oxygen XML Author 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 Author.

Native XML Database (NXD) Support

Native XML databases have an XML-based internal model and their fundamental unit of storage is XML. Oxygen XML Author offers support for the following native XML databases:

- Berkeley DB XML;
- eXist;
- MarkLogic;
- Documentum xDb (X-Hive/DB) 10;
- Oracle XML DB.

To watch our video demonstration about the integration between the XML native databases and Oxygen XML Author, go to *http://www.oxygenxml.com/demo/Author_Database_XML_Native.html*.

Configuring Database Data Sources

This section describes the procedures for configuring the data sources for native databases.

How to Configure a Berkeley DB XML Data Source

Oxygen XML Author supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16.

1. *Open the **Preferences** dialog* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *Berkeley DBXML* from the **Driver type** combo box.
5. Press the **Add** button to add the Berkeley DB driver files.

   The driver files for the Berkeley DB database are the following:

   - db.jar (check for it into `[DBXML_DIR] / lib` or `[DBXML_DIR] / jar`)
   - dbxml.jar (check for it into `[DBXML_DIR] / lib` or `[DBXML_DIR] / jar`)

   Where `[DBXML_DIR]` is the Berkeley DB XML database root directory. For example on Windows it is: `C:\Program Files\Oracle\Berkeley DB XML <version>`.

6. Click the **OK** button to finish the data source configuration.

How to Configure an eXist Data Source

Oxygen XML Author supports eXist database server versions 1.3, 1.4 and 1.5.

1. *Open the **Preferences** dialog* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *eXist* from the **Driver type** 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 XML Author.

   - `exist.jar`
   - `lib/core/xmldb.jar`
   - `lib/core/xmlrpc-client-3.1.x.jar`
   - `lib/core/xmlrpc-common-3.1.x.jar`
   - `lib/core/ws-commons-util-1.0.x.jar`

> **Note:** For eXist database server version 1.5 and 2.0, the following driver files must also be added in the dialog box for setting up the datasource:
>
> - `lib/core/slf4j-api-1.x.x.jar`
> - `lib/core/slf4j-log4j12-1.x.x.jar`

The version number from the driver file names may be different for your eXist server installation.

6. Click the **OK** button to finish the connection configuration.

To watch our video demonstration about running XQuery against an eXist XML database, go to *http://www.oxygenxml.com/demo/eXist_Database.html*.

How to Configure a MarkLogic Data Source

Available in the Enterprise edition only.

> **Note:** Oxygen XML Author supports MarkLogic version 4.0 or later.

1. *Open the Preferences dialog* and go to **Data Sources**.

2. Click the **New** button in the **Data Sources** panel.

3. Enter a unique name for the data source.

4. Select *MarkLogic* from the **Type** combo box.

5. Press the **Add** button to add the MarkLogic driver file (`marklogic-xcc-{server_version}`, where *{server_version}* is the MarkLogic server version.)

You can download the driver file from: *http://community.marklogic.com/download*.

6. Click the **OK** button to finish the data source configuration.

How to Configure a Documentum xDb (X-Hive/DB) 10 Data Source

Available in the Enterprise edition only.

1. *Open the Preferences dialog* and go to **Data Sources**.

2. Click the **New** button in the **Data Sources** panel.

3. Enter a unique name for the data source.

4. Select *XHive* from the **Driver type** 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 Author 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. *Open the **Preferences** dialog* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
   a) Set the path to the Berkeley DB XML `database` directory in the **Environment home directory field**. Use a directory with write access. Do NOT use the installation directory where Berkeley DB XML is installed if you do not have write access to that directory.
   b) Select the **Verbosity** level: DEBUG, INFO, WARNING, or ERROR.
   c) Optionally, you can select the check-box **Join existing environment**.

      If checked, an attempt is made to join an existing environment in the specified home directory and all the original environment settings are preserved. If that fails, try reconfiguring the connection with this option unchecked.

6. Click the **OK** button to finish the connection configuration.

How to Configure an eXist Connection

The steps for configuring a connection to an eXist database are the following:

1. *Open the **Preferences** dialog* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** combo box.
5. Fill-in the connection details.
   a) Set the URI to the installed eXist engine in the **XML DB URI** field.
   b) Set the user name in the **User** field.
   c) Set the password in the **Password** field.
   d) Enter the start collection in the **Collection** field.

      eXist organizes all documents in hierarchical collections. Collections are like directories. They are used to group related documents together. This text field allows the user to set the default collection name.

6. Click the **OK** button to finish the connection configuration.

   To watch our video demonstration about running XQuery against an eXist XML database, go to *http://www.oxygenxml.com/demo/eXist_Database.html*.

The Create eXist-db XML connection Dialog

A quick way to create an eXist connection is to use the dedicated **Create eXist-db XML connection** dialog. *Open the **Preferences** dialog* and go to **Data Sources** and click **Create eXist-db XML connection**. After you fill in the fields in this dialog, click **OK** and go to **Window** > **Show View** > **Data Source Explorer** to view your connection.

To create an eXist connection using this dialog, Oxygen XML Author expects the `exist/webstart/exist.jnlp` path to be accessible at the **Host** and **Port** provided.

How to Configure a MarkLogic Connection

Available in the Enterprise edition only.

> **Note:** Oxygen XML Author supports MarkLogic version 4.0 or later.

The steps for configuring a connection to a MarkLogic database are the following:

1. *Open the **Preferences** dialog* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** 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 Author uses XCC connector to interact with MarkLogic XDBC server and requires the basic authentication schema to be set. Starting with version MarkLogic 4.0 the default authentication method when you create a HTTP or WebDAV Server is digest, so make sure to change it to basic.

   b) Set the port number of the MarkLogic engine in the **Port** field. A MarkLogic XDBC application server must be configured on the server on this port. This XDBC server will be used to execute XQuery expressions against the server. Later on, if you want to change the XDBC server, instead of editing the configuration just use the *Use it to execute queries* action from Data Source Explorer.

   c) Set the user name to access the MarkLogic engine in the **User** field.

   d) Set the password to access the MarkLogic engine in the **Password** field.

   e) Optionally set the URL used for browsing the MarkLogic database in the **Data Source Explorer** view in the **WebDAV URL** field.

      The **Database** field specifies the database over which the XQuery expressions are executed. In case you set this option to default, the database associated to the application server of the configured port is used.

6. Click the **OK** button to finish the connection configuration.

How to Configure an Documentum xDb (X-Hive/DB) 10 Connection

The steps for configuring a connection to a Documentum xDb (X-Hive/DB) 10 database are the following.

> 📝 **Note:** The bootstrap type of X-Hive/DB connections is not supported in Oxygen XML Author. The following procedure explains the `xhive://` protocol connection type.

1. *Open the Preferences dialog* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** 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 shows your database connections. You can connect to a database simply by expanding the connection node. The database structure can be expanded up to column level. supports multiple simultaneous database connections and the connections tree provides an easy way to browse them.
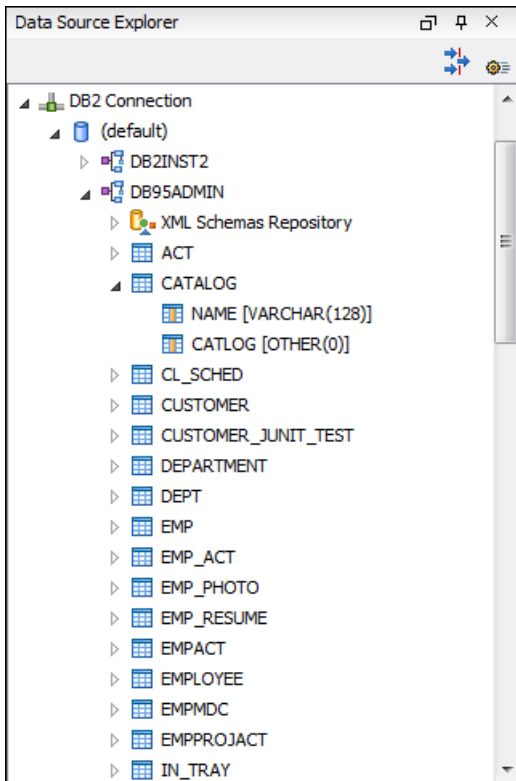
Figure 197: Data Source Explorer View

The following objects are displayed in the **Data Source Explorer** view:

- **Connection**;
- **Collection (Catalog)**;
- **XML Schema Repository**;
- **XML Schema Component**;
- **Schema**;
- **Table**;
- **System Table**;
- **Table Column**.

A   **collection** (called *catalog* in some databases) is a hierarchical container for   **resources** and further sub-collections. There are two types of resources:

- **XML resource** - an XML document or a document fragment, selected by a previously executed XPath query.
- **non XML resource**

Note: For some connections you can add or move resources into a container by dragging them from:

  - **Project view**;
  - the default file system application (Windows Explorer on Windows or Finder on Mac OS X, for example);
  - or from another database container.

The following actions are available in the view's toolbar:

Filters

Opens the **Data Sources / Table Filters** *Preferences page*, allowing you to decide which table types will be displayed in the **Data Source Explorer** view.

⚙≡ **Configure Database Sources**

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

Oracle XML DB Browser

Oracle XML DB is a feature of the Oracle database. It provides a high-performance, native XML storage and retrieval technology. Oxygen XML Author allows the user to browse the native Oracle XML Repository and perform various operations on the resources in the repository.
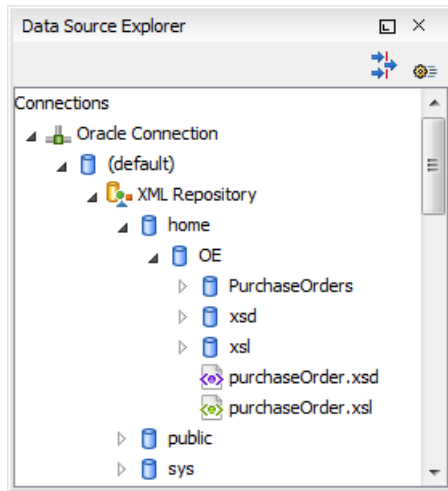
Figure 198: Browsing the Oracle XML DB Repository

The actions available at XML Repository level are the following:

Refresh

Performs a refresh of the XML Repository.

Add container

Adds a new child container to the XML Repository

Add resource

Adds a new resource to the XML Repository.

The actions available at container level are the following:

Refresh

Performs a refresh of the selected container.

Add container

Adds a new child container to the current one

Add resource

Adds a new resource to the folder.

Delete

Deletes the current container.

Properties

Shows various properties of the current container.

The actions available at resource level are the following:

Refresh

Performs a refresh of the selected resource.

Open

Opens the selected resource in the editor.

Rename

Renames the current resource.

Move

Moves the current resource to a new container (also available through drag and drop).

Delete

Deletes the current resource.

Copy location

Allows you to copy to clipboard an application specific URL for the resource which can then be used for various actions like opening or transforming the resources.

Properties

Shows various properties of the current resource.

Compare

This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

For running XQuery transformation on collections from XML Repository please see *a tutorial from Oracle*.

PostgreSQL Connection

Oxygen XML Author allows the user to browse the structure of the PostgreSQL database in the **Data Source Explorer** view and open the tables in the **Table Explorer** view.
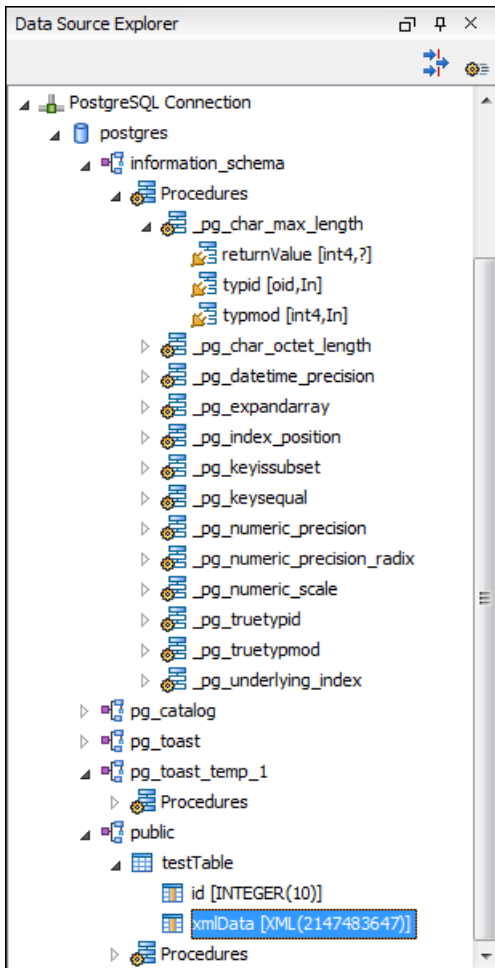
Figure 199: Browsing a PostgreSQL repository

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.

Compare

This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

Berkeley DB XML Connection

This section explains the actions that are available on a Berkeley DB XML connection.

Actions Available at Connection Level

In a Berkeley DB XML repository the actions available at connection level in the **Data Source Explorer** view are the following:

Refresh

Performs a refresh of the selected node's subtree.

Disconnect

Closes the current database connection.

Configure Database Sources

Opens *the Data Sources preferences page* where you can configure both data sources and connections.

Add container

Adds a new container in the repository with the following attributes.

- **Name** - The name of the new container.
- **Container type** - At creation time, every container must have a type defined for it. This container type identifies how XML documents are stored in the container. As such, the container type can only be determined at container creation time; you cannot change it on subsequent container opens. Containers can have one of the following types specified for them:

  - **Node container** - XML documents are stored as individual nodes in the container. That is, each record in the underlying database contains a single leaf node, its attributes and attribute values if any, and its text nodes, if any. Berkeley DB XML also keeps the information it needs to reassemble the document from the individual nodes stored in the underlying databases. This is the default, and preferred, container type.
  - **Whole document container** - The container contains entire documents. The documents are stored without any manipulation of line breaks or whitespace.

- **Allow validation** - If checked it causes documents to be validated when they are loaded into the container. The default behavior is to not validate documents.
- **Index nodes** - If checked it causes indices for the container to return nodes rather than documents. The default is to index at the document level. This property has no meaning if the container type is whole document container.

Properties

Shows a dialog containing a list of the Berkeley connection properties: version, home location, default container type, compression algorithm, etc.

Actions Available at Container Level

In a Berkeley DB XML repository the actions available at container level in the **Data Source Explorer** view are the following:

Add Resource

Adds a new XML resource to the selected container.

Rename

Allows you to specify a new name for the selected container.

Delete

Removes the selected container from the database tree.

Edit indices

Allows you to edit the indices for the selected container.

Refresh

Performs a refresh of the selected node's subtree.

Properties

Displays a dialog with a list of properties of the Berkeley container like: container type, auto indexing, page size, validate on load, compression algorithm, number of documents, etc.
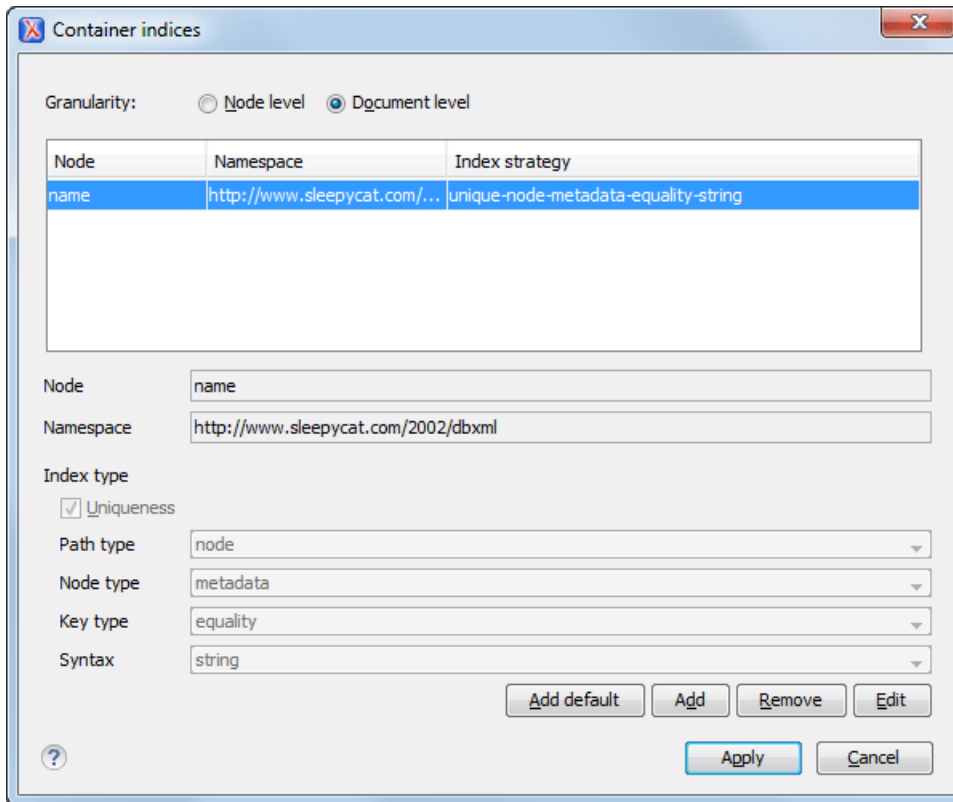
Figure 200: Container indices

The fields of the dialog are the following:

- Granularity:

    - **Document level** granularity is good for retrieving large documents.
    - **Node level** granularity is good for retrieving nodes from within documents.

- Add / Edit indices:

    - **Node** - The node name.
    - **Namespace** - The index namespace
    - Index strategy:

        - **Index type**:

            - **Uniqueness** - Indicates whether the indexed value must be unique within the container
            - **Path type**:

                - **node** - Indicates that you want to index a single node in the path
                - **edge** - Indicates that you want to index the portion of the path where two nodes meet

            - **Node type**:

                - **element** - An element node in the document content.
                - **attribute** - An attribute node in the document content.
                - **metadata** - A node found only in a document's metadata content.

            - **Key type**:

- **equality** - Improves the performances of tests that look for nodes with a specific value
- **presence** - Improves the performances of tests that look for the existence of a node regardless of its value
- **substring** - Improves the performance of tests that look for a node whose value contains a given substring

- **Syntax types** - The syntax describes what sort of data the index will contain and is mostly used to determine how indexed values are compared.

Actions Available at Resource Level

In a Berkeley DB XML repository the actions available at resource level in the **Data Source Explorer** view are the following:

Refresh

Performs a refresh of the selected resource.

Open

Opens the selected resource in the editor.

Rename

Allows you to change the name of the selected resource.

Move

Allows you to move the selected resource in a different container in the database tree (also available through drag and drop).

Delete

Removes the selected resource from the container.

Copy location

Allows you to copy to clipboard an application specific URL for the resource which can then be used for various actions like opening or transforming the resources.

Compare

This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

eXist Connection

This section explains the actions that are available on an eXist connection.

Actions Available at Connection Level

For an eXist database the actions available at connection level in the **Data Source Explorer** view are the following:

Configure Database Sources

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

Disconnect

Closes the current database connection.

Refresh

Performs a refresh of the selected node's subtree.

Actions Available at Container Level

For an eXist database the actions available at container level in the **Data Source Explorer** view are the following:

New File

Creates a file in the selected container.

New Collection

Creates a collection.

Import Folders

Adds recursively the content of specified folders from the local file system.

Import Files

Adds a set of XML resources from the local file system.

Cut

Cuts the selected containers.

Copy

Copies the selected containers.

> **Note:** You can add or move resources into container by dragging them from Project view, the default file system application (Windows Explorer on Windows or Finder on 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 OS X, for example) or from another database container.

Paste

Pastes the copied resources.

Delete

Removes the selected resource from the collection.

Copy location

Allows you to copy to clipboard an application-specific URL for the resource which can then be used for various actions like opening or transforming the resources.

Properties

Allows the user to view various useful properties associated with the resource.

Save As

Allows you to save the name of the selected binary resource as a file on disk.

Compare

This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

MarkLogic Connection

Once you configure a MarkLogic connection, you can use the **Data Source Explorer** view to display all the application servers configured on the server. You can expand each application server and view all the modules that it is configured to use. The **Data Source Explorer** view allows you to open and edit these modules.

> 📄 **Note:** To browse modules located in a database, directory properties must be associated with them. These directory properties are generated automatically if the *directory creation* property of the database is set to automatic. In case this property is set to *manual* or *manual-enforced*, add the directory properties of the modules manually, using the XQuery function `xdmp:directory-create()`.

Manually Adding Directory Properties

For two documents with the `/code/modules/main.xqy` and `/code/modules/imports/import.xqy` IDs, run this query:
```
(xdmp:directory-create('/code/modules/'),
xdmp:directory-create('/code/modules/imports/')).
```

For further information about directory properties go to: *http://blakeley.com/blogofile/2012/03/19/directory-assistance/*

When you execute or debug XQuery files opened from this view, the imported modules are better identified by the MarkLogic server. In a module, you are also able to add breakpoints that the debugger takes into account.

> 📄 **Note:** Add breakpoints in the modules of the application server that executes the debugging.

> 📄 **Note:** Open XQuery modules from the application server involved in the debugging or execution process.

In the **Requests** container of each application server Oxygen XML Author presents both the queries stopped for debugging and the queries that are still running. At the end of your session, to clean up the entire **Requests** container, right click it an use the **Cancel all running requests** action.
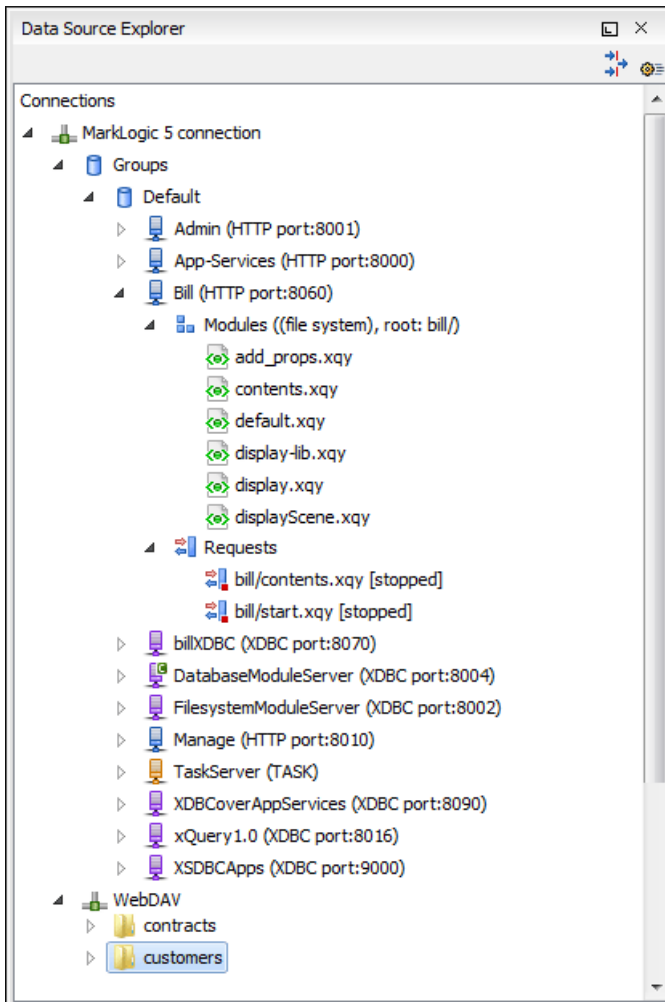
Figure 201: MarkLogic Connection in Data Source Explorer

The **Data Source Explorer** view presents all the application servers available on the MarkLogic server. To change the XDBC application server that Oxygen XML Author uses to execute XQuery expressions, select the **Use it to execute queries** option from its contextual menu.

To manage resources for a MarkLogic database through WebDAV, configure a WebDAV URL in *the MarkLogic connection*.

The following actions are available in the contextual menu of the WebDAV connection:

* connection level actions:

  Configure Database Sources...

  Opens the **Data Sources** *preferences page*. Here you can configure both data sources and connections.

  New Folder...

  Creates a new folder on the server.

  Import Files...

  Allows you to add a new file on the server.

  Refresh

  Performs a refresh of the connection.

  Find/Replace in Files...

  Allows you to find and replace text in multiple files from the server.

- folder level actions:

  New File

  Creates a new file on the server in the current folder.

  New Folder...

  Creates a new folder on the server.

  Import Folders...

  Imports folders on the server.

  Import Files - allows you to add a new file on the server in the current folder;

  Cut

  Removes the current selection and places it in the clipboard.

  Copy

  Copies the current selection.

  Rename

  Allows you to change the name of the selected folder.

  Delete

  Removes the selected folder.

  Refresh

  Refreshes the sub-tree of the selected node.

  Find/Replace in Files...

  Allows you to find and replace text in multiple files from the server.

- file level actions:

  Open

  Allows you to open the selected file in the editor;

  Cut

  Removes the current selection and places it in the clipboard.

  Copy

  Copies the current selection.

  Copy Location

  Copies an application specific URL for the selected resource to the clipboard. You can use this URL for various actions like opening or transforming the resources.

  Rename

  Allows you to change the name of the selected file.

  Delete

  Removes the selected file.

  Refresh

  Performs a refresh of the selected node.

  Properties

  Displays the properties of the current file in a dialog.

  Find/Replace in Files...

  Allows you to find and replace text in multiple files from the server.

Compare

This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

Documentum xDb (X-Hive/DB) Connection

This section explains the actions that are available on a Documentum xDb (X-Hive/DB) 10 connection.

Actions Available at Connection Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at connection level in the **Data Source Explorer** view are the following:

Refresh

Performs a refresh of the selected node's subtree.

Disconnect

Closes the current database connection.

Configure Database Sources

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

Add library

Allows you to add a new library.

Insert XML Instance

Allows you to add a new XML resource directly into the database root. See *Documentum xDb (X-Hive/DB) 10 Parser Configuration* for more details.

Insert non XML Instance

Allows you to add a new non XML resource directly into the database root.

Properties

Displays the connection properties.

Actions Available at Catalog Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at catalog level in the **Data Source Explorer** view are the following:

Refresh

Performs a refresh of the selected catalog.

Add as models

Allows you to add a new abstract schema model to the selected catalog.

Set default schema

Allows you to set a default DTD to be used for parsing. It is not possible to set a default XML Schema.

Clear default schema

Allows you to clear the default DTD. The action is available only if there is a DTD set as default.

Properties

Displays the catalog properties.

Actions Available at Schema Resource Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at schema resource level in the **Data Source Explorer** view are the following:

Refresh

Performs a refresh of the selected schema resource.

Open

Opens the selected schema resource in the editor.

Rename

Allows you to change the name of the selected schema resource.

Save As

Allows you to save the selected schema resource as a file on disk.

Delete

Removes the selected schema resource from the catalog

Copy location

Allows you to copy to clipboard the URL of the selected schema resource.

Set default schema

Allows you to set the selected DTD to be used as default for parsing. The action is available only for DTD.

Clear default schema

Allows you to unset the selected DTD. The action is available only if the selected DTD is the current default to be used for parsing.

Actions Available at Library Level

For a Documentum xDb (X-Hive/DB) 10 database the actions available at library level in the **Data Source Explorer** view are the following:

Refresh

Performs a refresh of the selected library.

Add library

Adds a new library as child of the selected library.

Add local catalog

Adds a catalog to the selected library. By default, only the root-library has a catalog, and all models would be stored there.

Insert XML Instance

Allows you to add a new XML resource to the selected library. See *Documentum xDb (X-Hive/DB) 10 Parser Configuration* for more details.

Insert non XML Instance

Allows you to add a new non XML resource to the selected library.

Rename

Allows you to specify a new name for the selected library.

Move

Allows you to move the selected library to a different one (also available through drag and drop).

Delete

Removes the selected library.

Properties

Displays the library properties.

Actions Available at Resource Level

When an XML instance document is added For a Documentum xDb (X-Hive/DB) 10 database the actions available at resource level in the **Data Source Explorer** view are the following:

Refresh

Performs a refresh of the selected resource.

Open

Opens the selected resource in the editor.

Rename

Allows you to change the name of the selected resource.

Move

Allows you to move the selected resource in a different library in the database tree (also available through drag and drop).

> **Note:** You can copy or move resources by dragging them from another database catalog.

Save As

Allows you to save the selected binary resource as a file on disk.

Delete

Removes the selected resource from the library.

Copy location

Allows you to copy to clipboard the URL of the selected resource.

Add AS model

Allows you to add an XML schema to the selected XML resource.

Set AS model

Allows you to set an active AS model for the selected XML resource.

Clear AS model

Allows you to clear the active AS model of the selected XML resource.

Properties

Displays the resource properties. Available only for XML resources.

Compare

This action is available in the contextual menu of two selected resources. Select this action to compare the resources using Diff Files.

Validation of an XML resource stored in an Documentum xDb (X-Hive/DB) 10 database is done against the schema associated with the resource in the database.

Documentum xDb (X-Hive/DB) 10 Parser Configuration for Adding XML Instances

When an XML instance document is added to a Documentum xDb (X-Hive/DB) 10 connection or library it is parsed with an internal XML parser of the database server. The following options are available for configuring this parser:

- DOM Level 3 parser configuration parameters. More about each parameter can be found here: *DOM Level 3 Configuration*.
- Documentum xDb (X-Hive/DB) 10 specific parser parameters (for more information please consult the Documentum xDb (X-Hive/DB) 10 manual):
  - **xhive-store-schema** - If checked, the corresponding DTD's or XML schemas are stored in the catalog during validated parsing.
  - **xhive-store-schema-only-internal-subset** - Stores only the internal subset of the document (not any external subset). This options modifies the **xhive-store-schema** one (only has a function when that parameter is set to true, and when DTD's are involved). Select this option this option if you only want to store the internal subset of the document (not the external subset).
  - **xhive-ignore-catalog** - Ignores the corresponding DTD's and XML schemas in the catalog during validated parsing.
  - **xhive-psvi** - Stores **psvi** information on elements and attributes. Documents parsed with this feature turned on, give access to **psvi** information and enable support of data types by XQuery queries.

- **xhive-sync-features** - Convenience setting. With this setting turned on, parameter settings of XhiveDocumentIf are synchronized with the parameter settings of LSParser. Note that parameter settings **xhive-psvi** and **schema-location** are always synchronized.

Troubleshooting

Cannot save the file. DTD factory class `org.apache.xerces.impl.dv.dtd.DTDDVFactoryImpl` *does not extend from* `DTDDVFactory`

I am able to access my XML Database in the Data Source Explorer and open files for reading but when I try to save changes to a file, back into the database, I receive the following error:"Cannot save the file. DTD factory class `org.apache.xerces.impl.dv.dtd.DTDDVFactoryImpl` does not extend from `DTDDVFactory`." How can I fix this?

Answer:

xhive.jar contains a MANIFEST.MF with a classpath:

```
Class-Path: core/antlr-runtime.jar core/aspectjrt.jar core/fastutil-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.

WebDAV Connection

This section explains how to work with a WebDAV connection in the **Data Source Explorer** view.

How to Configure a WebDAV Connection

By default Oxygen XML Author is configured to contain a WebDAV data source connection called **WebDAV (S)FTP**. Based on this data source you can create a WebDAV connection for browsing and editing data from a database that provides a WebDAV interface. The connection will be available in *the Data Source Explorer view*. The steps for configuring a WebDAV connection are the following:

1. *Open the Preferences dialog* and go to **Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the WebDAV data sources in the **Data Source** combo box.
5. Fill-in the connection details:
   a) Set the URL to the WebDAV repository in the field **WebDAV URL**.
   b) Set the user name to access the WebDAV repository in the field **User**.
   c) Set the password to access the WebDAV repository in the field **Password**.
6. Click the **OK** button.

   To watch our video demonstration about the WebDAV support in Oxygen XML Author, go to *http://www.oxygenxml.com/demo/WebDAV_Support.html*.

WebDAV Connection Actions

This section explains the actions that are available on a WebDAV connection in the **Data Source Explorer** view.

Actions Available at Connection Level

The contextual menu of a WebDAV connection in the **Data Source Explorer** view contains the following actions:

Configure Database Sources...

Opens the **Data Sources** *preferences page*. Here you can configure both data sources and connections.

Disconnect

Stops the connection.

Import Files...

Allows you to add a new file on the server.

New Folder...

Creates a new folder on the server.

Refresh

Performs a refresh of the connection.

Find/Replace in Files...

Allows you to find and replace text in multiple files from the server.

Actions Available at Folder Level

The contextual menu of a folder node in a WebDAV connection in the **Data Source Explorer** view contains the following actions:

New File

Creates a new file on the server in the current folder.

New Folder...

Creates a new folder on the server.

Import Folders...

Imports folders on the server.

Import Files

Allows you to add a new file on the server in the current folder.

Cut

Removes the current selection and places it in the clipboard.

Copy

Copies the current selection.

Paste

Pastes the copied selection.

Rename

Allows you to change the name of the selected folder.

Delete

Removes the selected folder.

Refresh

Refreshes the sub-tree of the selected node.

Find/Replace in Files...

Allows you to find and replace text in multiple files from the server.

Actions Available at File Level

The contextual menu of a file node in a WebDAV connection in the **Data Source Explorer** view contains the following actions:

Open

Allows you to open the selected file in the editor.

Cut

Removes the current selection and places it in the clipboard.

Copy

Copies the current selection.

Copy Location

Copies an application specific URL for the selected resource to the clipboard. You can use this URL for various actions like opening or transforming the resources.

Rename

Allows you to change the name of the selected file.

Delete

Removes the selected file.

Refresh

Performs a refresh of the selected node.

Properties

Displays the properties of the current file in a dialog.

Find/Replace in Files...

Allows you to find and replace text in multiple files from the server.

BaseX Support

This section explains how to configure the BaseX XML database support. The BaseX support is composed of two parts:

• Resource management in the **Data Source Explorer** view;
• XQuery execution.

Resource management

Resource management is available by creating an WebDAV connection to the BaseX server.

First of all, make sure the BaseX HTTP Server is started. For details about starting the BaseX HTTP Server, go to *http://docs.basex.org/wiki/Startup#BaseX_HTTP_Server*. The configuration file for the HTTP server is named `.basex` and is located in the BaseX installation directory. This file might help you to find out the port on which the HTTP server is running. The default port for BaseX WebDAV is 8984.

To ensure everything is functioning, open an WebDAV URL inside a browser and check if it works. For example, the following URL gets a document from a database named TEST:
`http://localhost:8984/webdav/TEST/etc/factbook.xml`.

Once you are sure that the BaseX WebDAV service is working, you can configure the WebDAV connection in Oxygen XML Author as described in *How to Configure a WebDAV Connection* on page 580. The WebDAV URL should resemble this: `http://{hostname }:{port}/webdav/`. If the BaseX server is running on your own machine and it has the default configuration, the data required by the WebDAV connection is:

• WebDAV URL: `http://localhost:8984/webdav`;

- User: `admin`;

- Password: `admin`.

Once the WebDAV connection is created you can start browsing using *the **Data Source Explorer** view*.

XQuery Execution

XQuery execution is possible through an XQJ connection.

BaseX XQJ Data Source

First of all, create an XQJ data source as described in *How to Configure an XQJ Data Source* on page 534. The BaseX XQJ API-specific files that must be added in the configuration dialog are `xqj-api-1.0.jar`, `xqj2-0.1.0.jar` and `basex-xqj-1.2.3.jar` (the version names of the JAR file may differ). These libraries can be downloaded from *xqj.net/basex/basex-xqj-1.2.3.zip*. As an alternative, you can also find the libraries in the BaseX installation directory, in the **lib** subdirectory.

BaseX XQJ Connection

The next step is to create an XQJ connection as described in *How to Configure an XQJ Connection* on page 534.

For a default BaseX configuration, the following connection details apply (please modify them when necessary):

- *Port*: 1984
- *serverName*: localhost
- *user*: admin
- *password*: admin

XQuery execution

Now that the XQJ connection is configured, open in Oxygen XML Author the XQuery file you wish to execute and create a *Transformation Scenario* as described in *XQuery Transformation* on page 505. In the **Transformer** combo box, select the name of the XQJ connection you created. Apply the transformation scenario and the XQuery will be executed.

Chapter

15

Importing Data

Topics:

This chapter describes how you can import data stored in text format, Excel sheet, or relational database tables, into XML documents.

Introduction

Computer systems and databases contain data in incompatible formats and one of the most time-consuming activities has been to exchange data between these systems. Converting the data to XML can greatly reduce complexity and create data that can be read by different types of applications.

This is why Oxygen XML Author offers support for importing text files, MS Excel files, Database Data, and HTML files into XML documents. The XML documents can be further converted into other formats using the *Transform features*.

Import from Database

This section explains how to import data from a database into Oxygen XML Author.

Import Table Content as XML Document

The steps for importing the data from a relational database table are the following:

1. Clicking this action opens a dialog with all the defined database connections:

2. Select the connection to the database that contains the data.

    Only connections configured on relational data sources can be used to import data.

3. If you want to edit, delete or add a data source or connection click on the **Configure Database Sources** button. The **Preferences**/**Data Sources** option page is opened.
4. Click **Connect**.
5. From the catalogs list, click on a schema and choose the required table.
6. Click the **OK** button.

    The **Import Criteria** dialog opens next, with a default query string in the **SQL Query** pane:

    The dialog contains the following items:

    *   **SQL Preview** - If the **SQL Preview** button is pressed, it shows the labels that are used in the XML document and the first five lines from the database into the **Import settings** panel. All data items in the input are converted by default to element content, but this can be overridden by clicking the individual column headers. Clicking once on a column header (ex **Heading0**) causes the data from this column to be used as attribute values of the row elements. Click a second time and the column's data is ignored when generating the XML file. You can cycle through these three options by continuing to click the column header. The following symbols decorate the column header to indicate the type of content that column is converted to:

        *   <> symbols for data columns converted to element content
        *   = symbol for data columns converted to attribute content
        *   *x* symbol for ignored data

    *   **Change labels** - This button opens a new dialog, allowing you to edit the names of the root and row elements, change the XML name and the conversion criterion. The XML names can be edited by double-clicking the desired item and entering the required label. The conversion criterion can also be modified by selecting from the drop-down list **ELEMENT**, **ATTRIBUTE**, or **SKIPPED**.
    *   **Save in file** - If checked, the new XML document is saved at the specified path.

        > **Note:** If only **Open in editor** is checked, the newly created document is open in the editor, but as an unsaved file.

    *   **Generate XML Schema** - Allows you to specify the path of the generated XML Schema file.

7. Click the **SQL Preview** button.

The **SQL Query** string is editable. You can specify which fields are considered.

Use aliases if the following are true:

- the query string represents a join operation of two or more tables
- columns selected from different tables have the same name

The use of aliases avoids the confusion of two columns being mapped to the same name in the result document of the importing operation.

```
select s.subcat_id,
       s.nr as s_nr,
       s.name,
       q.q_id,
       q.nr as q_nr,
       q.q_text
  from faq.subcategory s,
         faq.question q
  where  ...
```

The input data is displayed in a tabular form in the **Import Settings** panel. The **XML Import Preview** panel contains an example of what the generated XML looks like.

Convert Table Structure to XML Schema

The structure of a table from a relational database can be imported in Oxygen XML Author as an XML Schema. This feature is activated by the **Generate XML Schema** option from the **Import criteria** dialog used in *the procedure for importing table data* as an XML instance document.

Import from MS Excel Files

Oxygen XML Author offers support for importing MS Excel Files. To import Excel files, go to **File** > **Import** > **MS Excel file** and in the **Import** dialog box select the file you want to import. In the **Available Sheets** section of this dialog, the sheets of the document you are importing are presented. Select a sheet and click next to move on to the second **Import** dialog box.
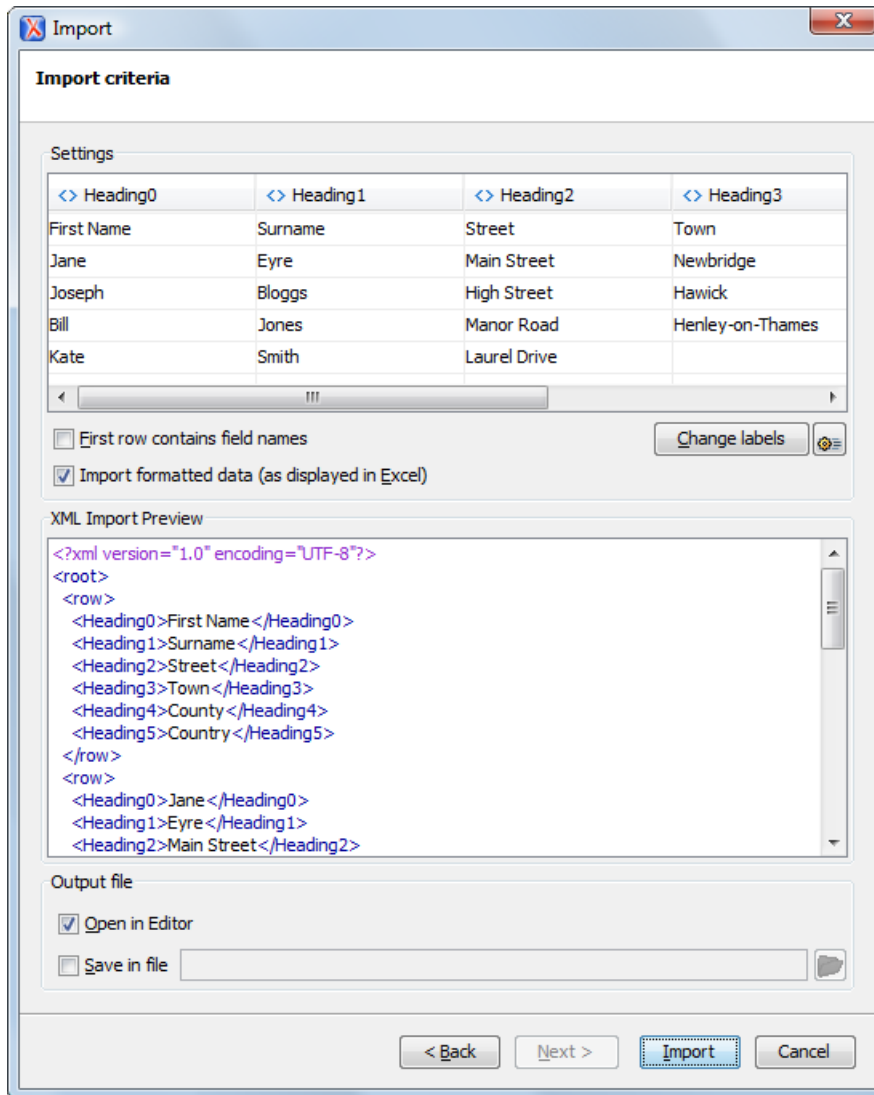
Figure 202: The "Import" Dialog Box - Import Criteria

The **Settings** section presents the data from the Excel sheet in a tabular form. It also contains the following options:

- **First row contains field names** - Uses the content from the first row to name the columns.
- **Import formatted data (as displayed in Excel)** - Keeps the Excel styling.
- **Change labels** - Opens the **Presentation Names** dialog box. The following options are available:

  - **Root Element** - allows you to edit the name of the Root element;
  - **Row Element** - allows you to edit the name of the Row element;
  - **Real Name** - contains the original name of each Heading;
  - **XML Name** - allows you to modify the names of the Headings;
  - **Criterion** - allows you to transform the Heading elements to attributes of the Root element.

- ⚙ **Import settings** - Opens the XML / Import preferences page.

The **XML Import Preview** section displays the Excel document in an XML format.

The **Output File** section contains the following options:

- **Open in Editor** - Opens the imported document in the Editor;
- **Save in File** - Saves the imported documented in the specified location.

When you finish configuring the options in these dialogs, click **Import**.

Import from MS Excel 2007-2010 (`.xlsx`)

To import XML from Excel 2007-2010 (`.xlsx`) documents, Oxygen XML Author needs additional libraries from the release 3.10 of Apache POI project. Follow these steps:

1. Download version 3.10 of the Apache POI project from *http://poi.apache.org/download.html*. If a newer version has been released, you can still find version 3.10 at *http://archive.apache.org/dist/poi/release/bin/*.
2. From the downloaded project locate and add the following `.jar` files in the `lib` directory of the installation folder of Oxygen XML Author :

    - `dom4j-1.6.1.jar`

    - `poi-ooxml-3.10-FINAL-20140208.jar`

    - `poi-ooxml-schemas-3.10-FINAL-20140208.jar`

    - `xmlbeans-2.3.0.jar`

Import from HTML Files

HTML is one of the formats that can be imported as an XML document. The steps needed are:

1. Go to menu **File** > **Import** > **HTML File ...**.
2. Enter the URL of the HTML document.
3. Select the type of the result XHTML document:

    - XHTML 1.0 Transitional
    - XHTML 1.0 Strict

4. Click the **OK** button.

The resulted document is an XHTML file containing a DOCTYPE declaration referring to the XHTML DTD definition on the Web. The parsed content of the imported file is transformed to XHTML Transitional or XHTML Strict depending on what radio button you chose when performing the import operation.

Import from Text Files

The steps for importing a text file into an XML file are the following:

1. Go to menu **File** > **Import** > **Text File...**
   The **Select text file** dialog box is displayed.
2. Select the URL of the text file.
3. Select the encoding of the text file.
4. Click the **OK** button.

    The **Import Criteria** dialog box is displayed:

    The input data is displayed in a tabular form. The **XML Import Preview** panel contains an example of what the generated XML document looks like. The names of the XML elements and the transformation of the first five lines from the text file are displayed in the **Import settings** section. All data items in the input are converted by default to element content, but this can be overridden by clicking the individual column headers. Clicking once a column header causes the data from this column to be used as attribute values of the row elements. Click the second time and the column's data is ignored when generating the XML file. You can cycle through these three options by continuing to click the column header. The following symbols decorate the column header to indicate the type of content that column is converted to:

- <> symbols for data columns converted to element content
- = symbol for data columns converted to attribute content
- *x* symbol for ignored data

5. Select the field delimiter for the import settings:

   - Comma;
   - Semicolon;
   - Tab;
   - Space;
   - Pipe.

6. Set other optional settings of the conversion.

   The dialog offers the following settings:

   - **First row contains field names** - If the option is enabled, you will notice that the table has moved up. The default column headers are replaced (where such information is available) by the content of the first row. In other words, the first row is interpreted as containing the field names. The changes are also visible in the preview of the XML document. To return to default settings (where the first row is interpreted as containing data and not fields names), simply uncheck the option.
   - **Change labels** - This button opens a new dialog, allowing you to edit the names of the root and row elements, change the XML name and the conversion criterion.

     The XML names can be edited by double-clicking the desired item and entering the required label. The conversion criterion can also be modified by selecting one of the drop-down list options: **ELEMENT**, **ATTRIBUTE**, or **SKIPPED**.

Chapter

16

Content Management System (CMS) Integration

Topics:

- *Integration with Documentum (CMS)*
- *Integration with Microsoft SharePoint*

This chapter explains how you can integrate Oxygen XML Author with a content management system (CMS), to edit the data stored in the CMS directly in Oxygen XML Author. Oxygen XML Author offers support for Documentum CMS and Microsoft SharePoint, but other CMSs can use the *plugin support* for similar integrations.

Integration with Documentum (CMS)

Oxygen XML Author 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 Author supports Documentum (CMS) version 6.5 or later with *Documentum Foundation Services 6.5* or later installed.

⚠ **Attention:**

It is recommended to use the latest 1.6.x Java version. It is possible that the Documentum (CMS) support will not work properly if you use other Java versions.

Configure Connection to Documentum Server

This section explains how to configure a connection to a Documentum server.

How to Configure a Documentum (CMS) Data Source

Available in the Enterprise edition only.

To configure a Documentum (CMS) data source you need the Documentum Foundation Services Software Development Kit (*DFS SDK*) corresponding to your server version. The *DFS SDK* can be found in the Documentum (CMS) server installation kit or it can be downloaded from *EMC Community Network*.

📄 **Note:** The *DFS SDK* can be found in the form of an archive named, for example, *emc-dfs-sdk-6.5.zip* for Documentum (CMS) 6.5.

1. *Open the Preferences dialog* and go to **Data Sources**.
   The **Preferences** dialog is opened at the **Data Sources** panel.
2. In the **Data Sources** panel click the **New** button.
3. Enter a unique name for the data source.
4. Select **Documentum (CMS)** from the driver type combo box.
5. Press the **Choose DFS SDK Folder** button.
6. Select the folder where you have unpacked the *DFS SDK* archive file.

   If you have indicated the correct folder the following Java libraries (jar files) will be added to the list (some variation of the library names is possible in future versions of the *DFS SDK*):

   * `lib/java/emc-bpm-services-remote.jar`
   * `lib/java/emc-ci-services-remote.jar`
   * `lib/java/emc-collaboration-services-remote.jar`
   * `lib/java/emc-dfs-rt-remote.jar`
   * `lib/java/emc-dfs-services-remote.jar`
   * `lib/java/emc-dfs-tools.jar`
   * `lib/java/emc-search-services-remote.jar`
   * `lib/java/ucf/client/ucf-installer.jar`
   * `lib/java/commons/*.jar` (multiple jar files)
   * `lib/java/jaxws/*.jar` (multiple jar files)
   * `lib/java/utils/*.jar` (multiple jar files)

   📄 **Note:** If for some reason the jar files are not found, you can add them manually by using the **Add Files** and **Add Recursively** buttons and navigating to the `lib/java` folder from the *DFS SDK*.

7. Click the **OK** button to finish the data source configuration.

How to Configure a Documentum (CMS) Connection

Available in the Enterprise edition only.

The steps for configuring a connection to a Documentum (CMS) server are the following:

1. *Open the **Preferences** dialog* and go to **Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the previously configured Documentum (CMS) data sources in the **Data Source** combo box.
5. Fill-in the connection details:

   - **URL** - The URL to the Documentum (CMS) server: `http://<hostname>:<port>`
   - **User** - The user name to access the Documentum (CMS) repository.
   - **Password** - The password to access the Documentum (CMS) repository.
   - **Repository** - The name of the repository to log into.

6. Click the **OK** button to finish the configuration of the connection.

Known Issues

The following are known issues with the Documentum (CMS):

1. Please note that there is a known problem in the UCF Client implementation for Mac OS X from Documentum 6.5 which prevents you from viewing or editing XML documents from the repository on Mac OS X. The UCF Client is the component responsible for file transfer between the repository and the local machine. This component is deployed automatically from the server. Documentum 6.6 and later versions do not exhibit this problem.

   > **Note:** This issue was reproduced with Documentum 6.5 SP1. In Documentum 6.6 this is no longer reproducing.

2. In order for the Documentum driver to work faster on Linux, you need to specify to the JVM to use a weaker random generator, instead of the very slow native implementation. This can be done by modifying in the Oxygen XML Author 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 Author allows you to browse the structure of a Documentum repository in the **Data Source Explorer** view and perform various operations on the repository resources.

You can drag and drop folders and resources to other folders to perform move or copy operations with ease. If the drag and drop is between resources (drag the child item to the parent item) you can create a relationship between the respective resources.
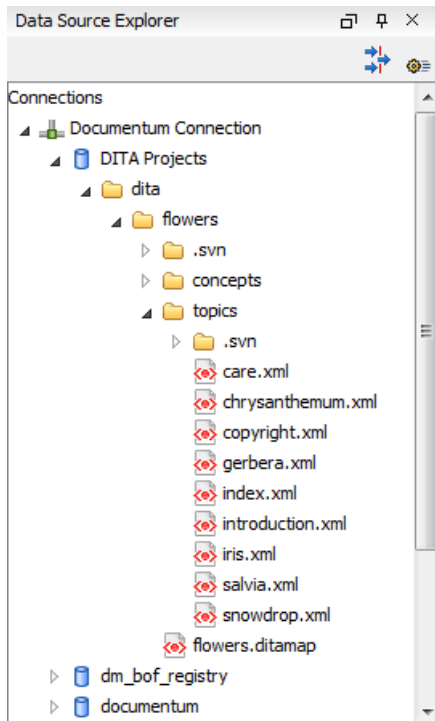
Figure 203: Browsing a Documentum repository

Actions Available on Connection

The contextual menu of a Documentum (CMS) connection in the **Data Source Explorer** view offers the following actions:

Configure Database Sources

Opens the *Data Sources preferences page* where you can configure both data sources and connections.

New Cabinet

Creates a new cabinet in the repository. The cabinet properties are:

- **Type** - The type of the new cabinet (default is **dm_cabinet**).
- **Name** - The name of the new cabinet.
- **Title** - The title property of the cabinet.
- **Subject** - The subject property of the cabinet.

Refresh

Refreshes the connection.

Actions Available on Cabinets / Folders

The actions available on a Documentum (CMS) cabinet in the **Data Source Explorer** view are the following:

New Folder

Creates a new folder in the current cabinet / folder. The folder properties are the following:

- **Path** - Shows the path where the new folder will be created.
- **Type** - The type of the new folder (default is **dm_folder**).
- **Name** - The name of the new folder.
- **Title** - The title property of the folder.
- **Subject** - The subject property of the folder.

New Document

Creates a new document in the current cabinet / folder. The document properties are the following:

- **Path** - Shows the path where the new document will be created.
- **Name** - The name of the new document.
- **Type** - The type of the new document (default is **dm_document**).
- **Format** - The document content type format.

Import

Imports local files / folders in the selected cabinet / folder of the repository. Actions available when performing an import:

- **Add Files** - Shows a file browse dialog and allows you to select files to add to the list.
- **Add Folders** - Shows a folder browse dialog that allows you to select folders to add to the list. The subfolders will be added recursively.
- **Edit** - Shows a dialog where you can change the properties of the selected file / folder from the list.
- **Remove** - Removes the selected files / folders from the list.

Rename

Changes the name of the selected cabinet / folder.

Copy

Copies the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop while holding the **(Ctrl (Meta on Mac OS))** key pressed.

Move

Moves the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop.

Delete

Deletes the selected cabinet / folder from the repository. The following options are available:

- **Folder(s)** - Allows you to delete only the selected folder or to delete recursively the folder and all subfolders and objects.
- **Version(s)** - Allows you to specify what versions of the resources will be deleted.
- **Virtual document(s)** - Here you can specify what happens when virtual documents are encountered. They can be either deleted either by themselves or together with their descendants.

Refresh

Performs a refresh of the selected node's subtree.

Properties

Displays the list of properties of the selected cabinet / folder.

Actions Available on Resources

The actions available on a Documentum (CMS) resource in the **Data Source Explorer** view are the following:

Edit

Checks out (if not already checked out) and opens the selected resource in the editor.

Edit with

Checks out (if not already checked out) and opens the selected resource in the specified editor / tool.

Open (Read-only)

Opens the selected resource in the editor. The resources are marked as read-only in the editor using a lock icon on the file tab. If you want to edit those resources, enable the *Can edit read only files* option.

Open with

Opens the selected resource in the specified editor / tool.

Check Out

Checks out the selected resource from the repository. The action is not available if the resource is already checked out.

Check In

Checks in the selected resource (commits changes) into the repository. The action is only available if the resource is checked out.
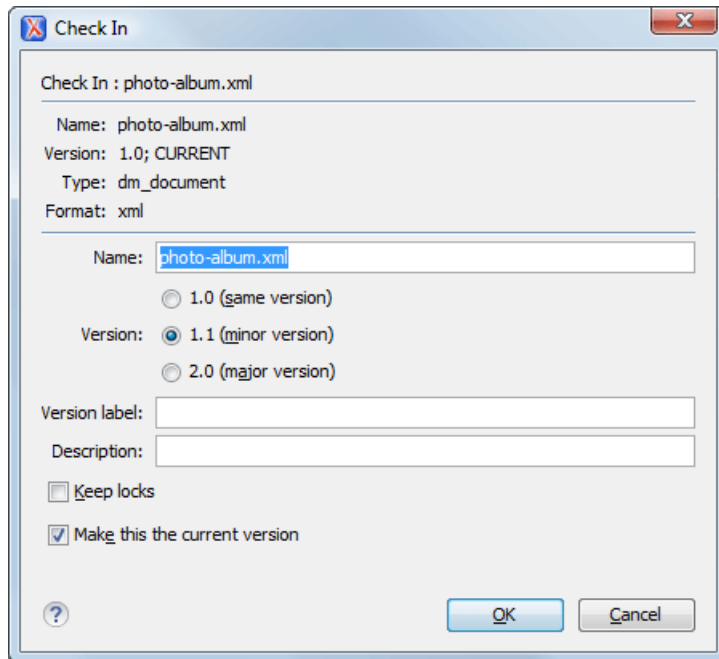
Figure 204: Check In Dialog

The following resource properties are available:

- **Name** - The resource name in the repository.
- **Version** - Allows you to choose what version the resource will have after being checked in.
- **Version label** - The label of the updated version.
- **Description** - An optional description of the resource.
- **Keep Locks** - When this option is enabled, the updated resource is checked into the repository but it also keeps it locked.
- **Make this the current version** - Makes the updated resource the current version (will have the *CURRENT* version label).

Cancel Checkout

Cancels the checkout process and loses all modifications since the checkout. Action is only available if the resource is checked out.

Export

Allows you to export the resource and save it locally.

Rename

Changes the name of the selected resource.

Copy

Copies the selected resource in a different location in the tree. Action is not available on virtual document descendants. This action can also be performed with drag and drop while holding the **Ctrl (Meta on OS X)** key pressed.

Move

Moves the selected resource in a different location in the tree. Action is not available on virtual document descendants and on checked out resources. This action can also be performed with drag and drop.

✖ **Delete**

Deletes the selected resource from the repository. Action is not available on virtual document descendants and on checked out resources.

Add Relationship

Adds a new relationship for the selected resource. This action can also be performed with drag and drop between resources.

Convert to Virtual Document

Allows you to convert a simple document to a virtual document. Action is available only if the resource is a simple document.

Convert to Simple Document

Allows you to convert a virtual document to a simple document. Action is available only if the resource is a virtual document with no descendants.

Copy location

Allows you to copy to clipboard an application-specific URL for the resource which can then be used for various actions like opening or transforming the resources.

🔄 **Refresh**

Performs a refresh of the selected resource.

📰 **Properties**

Displays the list of properties of the selected resource.

Transformations on DITA Content from Documentum (CMS)

Oxygen XML Author 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 Author or a customization of such a predefined DITA transformation.

Integration with Microsoft SharePoint

This section explains how to work with a SharePoint connection in the **Data Source Explorer** view.

📝   **Note:** The SharePoint connection is available in the Enterprise edition.

📝   **Note:** You can access documents stored on SharePoint Online for Office 365.

To watch our video demonstration about connecting to a repository located on a SharePoint server and using SharePoint, go to *http://www.oxygenxml.com/demo/SharePoint_Support.html*.and SharePoint Online for Office 365

How to Configure a SharePoint Connection

By default Oxygen XML Author contains a predefined **SharePoint** data source connection. Use this data source to create a connection to a SharePoint server which will be available in *the Data Source Explorer view*.

Follow these steps to configure a SharePoint connection:

1. *Open the Preferences dialog* and go to **Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select SharePoint in the **Data Source** combo box.
5. Fill-in the connection details:

   a) Set the URL to the SharePoint repository in the field **SharePoint URL**.

b) Set the server domain in the **Domain** field.

c) Set the user name to access the SharePoint repository in the **User** field.

d) Set the password to access the SharePoint repository in the **Password** field.

To watch our video demonstration about connecting to repository located on a SharePoint server, go to *http://www.oxygenxml.com/demo/SharePoint_Support.html*.

The SharePoint Browser View

To display the **SharePoint Browser** view, go to **Window** > **Show View** > **SharePoint Browser**. This view allows you to connect to a SharePoint repository and perform SharePoint-specific actions on the available resources.
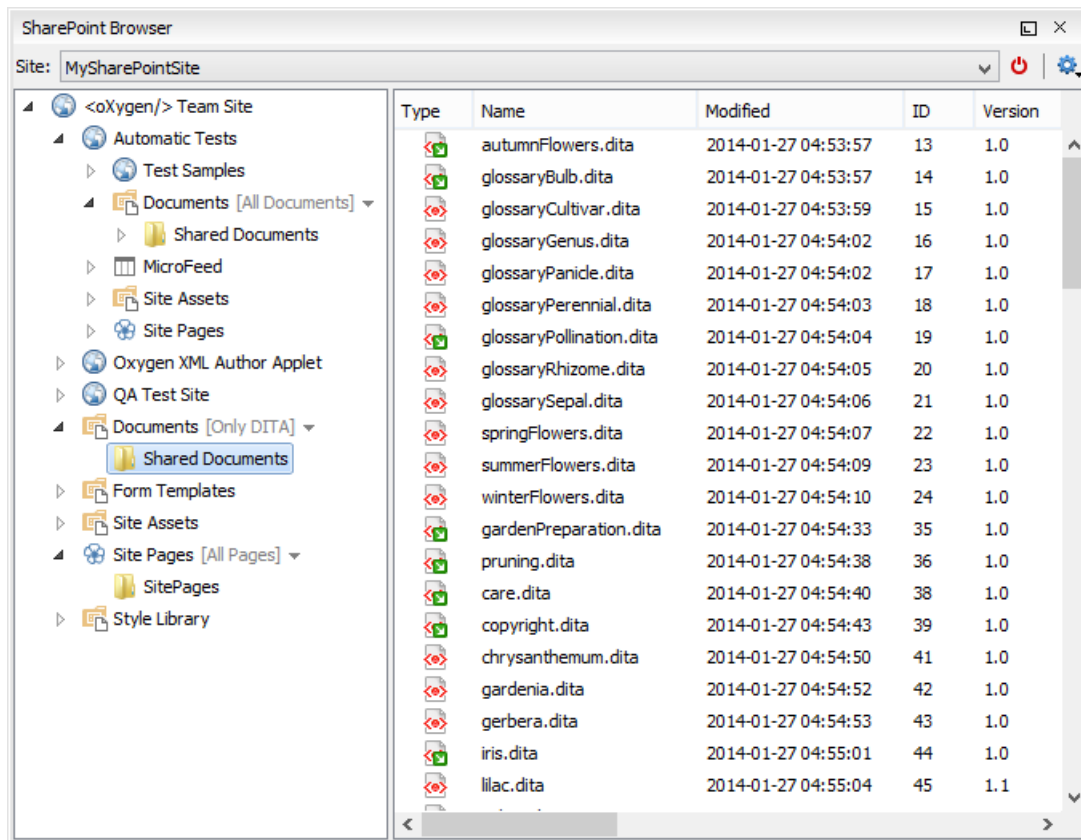
Figure 205: SharePoint Browser View

The view is split in several functional areas:

Connection Area

The following controls are available:

- The **Site** combo box allows you to select and connect to an already *defined SharePoint connection*.
- The ■ **Disconnect** action terminates the current connection.
- The ⚙ **Settings** drop-down menu contains actions that help you to quickly define a new connection or manage the existing ones from the **Data Source** options page: **New SharePoint Connection** and **Configure Database Sources**. Also, here you can choose one of the predefined view layouts.

SharePoint Site Navigation Area

If there is no connection selected in the **Site** combo box, this area is left blank and promotes the actions that allow you to quickly add SharePoint connections. Otherwise, the navigation area presents the SharePoint site structure in a tree-like fashion displaying the following node types: *sites* (⬜), *libraries*, and *folders*.

Depending on a node's type, a contextual menu offers customized actions that can be performed on that node.

> **Note:** The contextual menu of a folder allows you to create new folders, new documents, and to rename and delete the folder.

> **Note:** The rename and delete actions are not available for library root folders (folders located at first level in a SharePoint library).

Each library node display next to its name a drop down box where you can select the current library *view*. This functionality is also available on the node's contextual menu, under the **Current View** submenu.

Folder Content Area

The content of a folder is displayed in a tabular form, where each row represents the properties of a folder or document. The list of columns and the way the documents and folders are organized depends on the currently selected view of the parent library.

Action	Description	Available for	
		folders	**documents**
📂 **Open**	Displays the content of the current selected folder. Opens the current document for editing.		
Rename...	Renames the current node on server.		
✖ **Delete...**	Deletes the current node from the server.		
Copy Location	Copies to clipboard the URL of the current node.		
⬇ **Check Out**	Reserves the current document for your use so that other users cannot change it while you are editing it.		
Check In...	Commits on the server the changes you made to the document, so that other users can see them. It also makes the document available for editing to other users.		
Discard Check Out	Discards the previous checkout operation, making the file available for editing to other users.		
🔄 **Refresh**	Queries the server in order to refresh the available properties of the current node.		

You can filter and sort the displayed items. To display the available filters of a column, click the filter widget located on the column's header. You can apply multiple filters at the same time.

> **Note:** A column can be filtered or sorted only if it was configured this way on the server side.
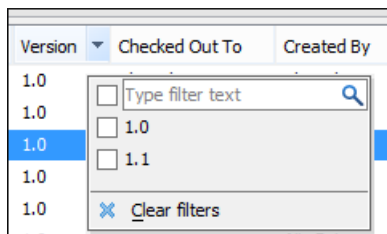
SharePoint Connection Actions

This section explains the actions that are available on a SharePoint connection in the **Data Source Explorer** view.

Actions Available at Connection Level

The contextual menu of a SharePoint connection in the **Data Source Explorer** view contains the following actions:

Configure Database Sources...

Opens the **Data Sources** *preferences page*. Here you can configure both data sources and connections.

Disconnect

Stops the connection.

New Folder...

Creates a new folder on the server.

Import Files...

Allows you to add a new file on the server.

Refresh

Performs a refresh of the connection.

Find/Replace in Files...

Allows you to find and replace text in multiple files from the server.

Actions Available at Folder Level

The contextual menu of a folder node in a SharePoint connection in the **Data Source Explorer** view contains the following actions:

New File

Creates a new file on the server in the current folder.

New Folder...

Creates a new folder on the server.

Import Folders...

Imports folders on the server.

Import Files

Allows you to add a new file on the server in the current folder.

Cut

Removes the current selection and places it in the clipboard.

Copy

Copies the current selection.

Paste

Pastes the copied selection.

Rename

Allows you to change the name of the selected folder.

Delete

Removes the selected folder.

Refresh

Refreshes the sub-tree of the selected node.

Find/Replace in Files...

Allows you to find and replace text in multiple files from the server.

Actions Available at File Level

The contextual menu of a file node in a SharePoint connection in the **Data Source Explorer** view contains the following actions:

Open

Allows you to open the selected file in the editor.

Cut

Removes the current selection and places it in the clipboard.

Copy

Copies the current selection.

Copy Location

Copies an application specific URL for the selected resource to the clipboard. You can use this URL for various actions like opening or transforming the resources.

Check Out

Checks out the selected document on the server.

Check In

Checks in the selected document on the server. This action opens the **Check In** dialog. In this dialog, the following options are available:

- **Minor Version** - increments the minor version of the file on the server
- **Major Version** - increments the major version of the file on the server
- **Overwrite** - overwrites the latest version of the file on the server
- **Comment** - allows you to comment on a file that you check in

Discard Check Out

Discards the previous checkout operation, making the file available for editing to other users.

Rename

Allows you to change the name of the selected file.

Delete

Removes the selected file.

Refresh

Performs a refresh of the selected node.

Properties

Displays the properties of the current file in a dialog.

Find/Replace in Files...

Allows you to find and replace text in multiple files from the server.

Note: The **Check In**, **Check Out**, and **Discard Check Out** options are available in the Enterprise edition only.

Chapter

17

Tools

Oxygen XML Author ships with a set of tools oriented to XML related tasks. You have access to a revision control system, a comparing and merging solution and also to other tools like a large file viewer and a hex viewer.

SVN Client

The Syncro SVN Client is a client application for the Apache Subversion™ version control system, compatible with Subversion 1.6, 1.7 and 1.8 servers. It manages files and directories that change over time and are stored in a central repository. The version control repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to access older versions of your files and examine the history of how and when your data changed.

To start Syncro SVN Client, go to **Tools** > **SVN Client**.

Main Window

This section explains the main window of Syncro SVN Client.

Views

The main window consists of the following views:

- *Repositories view* - Allows you to define and manage Apache Subversion™ repository locations.
- *Working Copy view* - Allows you to manage with ease the content of the working copy.
- *History view* - Displays information (author name, revision number, commit message) about the changes made to a resource during a specified period of time.
- *Editor view* - Allows you to edit different types of text files, with full syntax-highlight.
- *Annotations view* - Displays a list with information regarding the structure of a document (author and revision for each line of text).
- *Compare view* - Displays the differences between two revisions of a text file from the working copy.
- *Image Preview* - Allows you to preview standard image files supported by Syncro SVN Client: JPG, GIF and PNG.
- *Compare Images view* - Displays two images side by side.
- *Properties view* - Displays the SVN properties of a resource under version control.
- *Console view* - Displays information about the currently running operation, similar with the output of the Subversion command line client.
- *Help view* - Shows information about the currently selected view.

The main window's status bar presents in the left side the operation in progress or the final result of the last performed action. In the right side there is a progress bar for the running operation and a stop button to cancel the operation.

Main Menu

The main menu of the Syncro SVN Client is composed of the following menus:

- **File** menu:

  New submenu:

  New File

  This operation creates a new file as child of the selected folder from the *Repositories view* tree, or from the *Working Copy view* tree, depending on which view was last focused. For the *Working Copy view*, the file is added to version control only if the selected folder is under version control.

  New Folder (Ctrl (Command on OS X) + Shift + F)

  This operation creates a new folder as child of the selected folder from the *Repositories view* tree or from the *Working Copy view* tree, depending on which view was last focused. For the *Working Copy view*, the folder is added to version control only if the selected directory is under version control.

  New External Folder (Ctrl (Command on OS X) + Shift + W)

  This operation allows you to add a new external definition on the selected folder. An externals definition is a mapping of a local directory to the URL - and ideally a particular revision - of a versioned directory, stored in the `svn:externals` property of the selected folder.

The URLs used in the externals definition format can be relative. You can specify the repository URLs to which the external folders point to using the following relative formats:

- **../** - Relative to the URL of the directory on which the `svn:externals` property is set.
- **^/** - Relative to the root of the repository in which the `svn:externals` property is versioned.
- **//** - Relative to the scheme of the URL of the directory on which the `svn:externals` property is set.
- **/** - Relative to the root URL of the server on which the `svn:externals` property is versioned.

⚠ **Important:** To change the target URL of an external definition or to delete an external item, do the following:

- modify or delete the item's definition found in the `svn:externals` property set on the parent folder
- use the **Update** operation on the item's parent folder for the change to take effect

📑 **Note:** Syncro SVN Client does not support definitions of local relative externals.

📂 Open (Ctrl (Command on OS X) + O)

This action opens the selected file in an editor where you can modify it. The action is active only when a single item is selected. The action opens a file with the internal editor or the external application associated with that file type. In case of a folder the action opens the selected folder with the system application for folders (for example Windows Explorer on Windows, Finder on OS X, etc). Folder opening is available only for folders selected in the *Working Copy view*. This action works on any file selection from the *Repositories view*, *Working Copy view*, *History view* or *Directory Change Set view*, depending on which view was last focused when invoking it.

Open with(Ctrl (Command on OS X) + Shift + O)

Displays the **Open with** dialog for specifying the editor in which the selected file is opened. In case multiple files are selected only external applications can be used to open the files. This action works on any file selection from *Repositories view*, *Working Copy view*, *History view* or *Directory Change Set view*, depending on which view was last focused when invoking it.

Show in Explorer/Show in Finder

Opens the parent directory of the selected working copy file and selects the file.

💾 Save (Ctrl (Command on OS X) + S)

Saves the local file currently opened in the editor or the **Compare** view.

Save as

Saves locally any file selected in the **Repositories**, **History**, and **Directory Change Sets** view.

Copy URL Location (Ctrl (Command on OS X) + Alt + U)

Copies to clipboard the URL location of the resource currently selected in the **Repositories** view.

📋 Copy to

Copies to a specified location the currently selected resource(s) either in **Repositories** or **Working copy** view.

📑 **Note:** This action can also be used from **History** and **Directory Change Set** views to recover older versions of a repository item.

Move to (Ctrl (Command on OS X) + M)

Moves to a specified location the currently selected resource(s) either in **Repositories** or **Working copy** view.

Rename (F2)

Renames the resource currently selected either in **Repositories** or **Working copy** view.

✖ Delete (Delete)

Deletes the resource currently selected either in **Repositories** or **Working copy** view.

Locking:

- **Scan for locks (Ctrl (Command on OS X) + L)** - Contacts the repository and recursively obtains the list of locks for the selected resources. A dialog containing the locked files and the lock description will be displayed. Only active for resources under version control. For more details see *Scanning for locks*.

- Lock **(Ctrl (Command on OS X) + K)** - Allows you to lock certain files for which you need exclusive access. You can write a comment describing the reason for the lock and you can also force (*steal*) the lock. The action is active only on files under version control. For more details on the use of this action see *Locking a file*.

- Unlock **(Ctrl (Command on OS X) + Alt + K)** - Releases the exclusive access to a file from the repository. You can also choose to unlock it by force (*break the lock*).

  Show SVN Properties (Ctrl (Command on OS X) + Shift + P)

  Brings up the *Properties view* and displays the SVN properties for a selected resource from *Repositories view* or *Working Copy view*, depending on which view was last focused when invoking it.

  File Information (Ctrl (Command on OS X) + I)

  Provides additional information for a selected resource from the *Working Copy view*. For more details please see the section *Obtain information for a resource*.

  Exit (Ctrl (Command on OS X) + Q)

  Closes the application.

- **Edit** menu:

  Undo (Ctrl (Command on OS X) + Z)

  Undo edit changes in the local file currently opened in the editor or the **Compare** view.

  Redo (Ctrl (Command on OS X) + Y)

  Redo edit changes in the local file currently opened in the editor or the **Compare** view.

  Cut (Ctrl (Command on OS X) + X)

  Cut selection to clipboard from the local file currently opened in the editor view or the **Compare** view.

  Copy (Ctrl (Command on OS X) + C)

  Copy selection to clipboard from the local file currently opened in the editor or the **Compare** view.

  Paste (Ctrl (Command on OS X) + V)

  Paste selection from clipboard in the local file currently opened in editor or the **Compare** view.

  Find/Replace (Ctrl (Command on OS X) + F)

  Perform find / replace operations in the local file currently opened in the editor or the **Compare** view.

  Find Next (F3)

  Go to the next find match using the same find options of the last find operation. The action runs in the editor panel and in any non-editable text area, for example the **Console** view.

  Find Previous (Shift + F3)

  Go to the previous find match using the same find options of the last find operation. The action runs in the editor panel and in any non-editable text area, for example the **Console** view.

- **Repository** menu:

  New Repository Location (Ctrl+Alt+N (Command+Alt+N on OS X))

  Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.
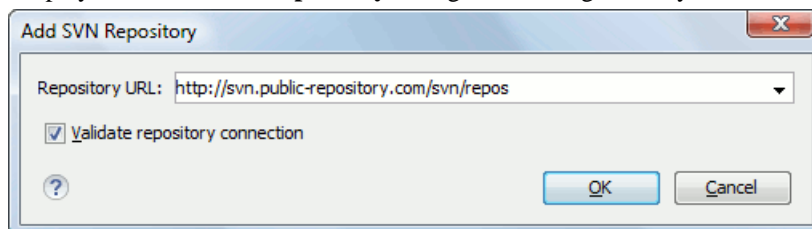
Figure 206: Add SVN Repository Dialog Box

If the **Validate repository connection** option is selected, the URL connection is validated before being added to the **Repositories** view.

Edit Repository Location (Ctrl+Alt+E (Command+Alt+E on OS X))

Context-dependent action that allows you to edit the selected repository location using the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.

Change the Revision to Browse (Ctrl+Alt+Shift+B (Command+Alt+Shift+B on OS X))

Context-dependent action that allows you to change the selected repository revision using the **Change the Revision to Browse** dialog. It is active only when a repository location root is selected.

Remove Repository Location (Ctrl+Alt+Shift+R (Command+Alt+Shift+R on OS X))

Allows you to remove the selected repository location from the view. It shows you a confirmation dialog before removal. It is active only when a repository location root is selected.

Refresh

Refreshes the resource selected in the **Repositories** view.

Check out (Ctrl+Alt+Shift+C (Command+Alt+Shift+C on OS X))

Allows you to create a working copy from a repository directory, on your local file system. To read more about this operation, see the section *Check out a working copy*.

Export

Exports a folder from the repository to the local file system.

Import sub-menu:

Import Folder Content (Ctrl+Alt+Shift+M (Command+Alt+Shift+M on OS X))

Depending on the selected folder from a repository, allows you to import the contents of a specified folder from the file system into it. To read more about this operation, see the section *Importing resources into a repository*.

> **Note:** The difference between **Import Folder Content** and **Share project** actions is that the latter one also converts the selected directory into a working copy.

Import File(s) (Ctrl+Alt+I (Command+Alt+I on OS X))

Imports the files selected from the files system into the selected folder from the repository.

• **Working Copy** menu:

(on OS X) Working Copies Manager

Opens a dialog with a list of working copies that the Apache Subversion™ client is aware of. In this dialog you can add existing working copies or remove no longer needed ones.

Switch to

Selects one of the following view modes: **All Files**, **Modified**, **Incoming**, **Outgoing**, or **Conflicts**.

Refresh (F5)

Refreshes the state of the selected resources or of the entire working copy if there is no selection.

Synchronize (Ctrl (Command on OS X) + Shift + S)

Connects to the repository and determines the working copy and repository changes made to the selected resources. The application switches to **Modified** view mode if the *Always switch to 'Modified' mode* option is selected.

Update (Ctrl (Command on OS X) + U)

Updates all the selected resources that have incoming changes to the HEAD revision. If one of the selected resources is a directory then the update for that resource will be recursive.

Update to revision/depth

Allows you to update the selected resources from the working copy to an earlier revision from the repository. You can also select the update *depth* for the current folder. You can find out more about the *depth* term in the *sparse checkouts* section.

Commit

Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what to commit by selecting or not resources. A directory will always be committed recursively. The unversioned resources will be deselected by default. In the commit dialog you can also enter a commit comment before sending your changes to the repository.

Update all (Ctrl (Command on OS X) + Shift + U)

Updates all resources from the working copy that have incoming changes. It performs a recursive update on the synchronized resources.

Commit all

Commits all the resources with outgoing changes. It is disabled when **Incoming** mode is selected or the synchronization result does not contain resources with outgoing changes. It performs a recursive commit on the synchronized resources.

Revert (Ctrl (Command on OS X) + Shift + V)

Undoes all local changes for the selected resources. It does not contact the repository, the files are obtained from Apache Subversion™ pristine copy. It is enabled only for modified resources. See *Revert your changes* for more information.

Edit conflict (Ctrl (Command on OS X) + E)

Opens the **Compare** editor, allowing you to modify the content of the currently conflicting resources. For more information on editing conflicts, see *Edit conflicts*.

Mark Resolved (Ctrl (Command on OS X) + Shift + R)

Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.

Mark as Merged (Ctrl (Command on OS X) + Shift + M)

Instructs the Subversion system that you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the *Merge conflicts* section for more information about how you can solve the pseudo-conflicts.

Override and Update

Drops any outgoing change and replaces the local resource with the HEAD revision. Action available on resources with outgoing changes, including the conflicting ones. See the *Revert your changes* section.

Override and Commit

Drops any incoming changes and sends your local version of the resource to the repository. Action available on conflicting resources. See also the section *Drop incoming modifications*.

Mark as copied

You can use this action to mark an item from the working copy as a copy of an other item under version control, when the copy operation was performed outside of an SVN client. The **Mark as copied** action is available when you select two items (both the new one and the source one) and depends on the state of the source item.

Mark as moved

You can use this action to mark an item from the working copy as being moved from another location of the working copy, when the move operation was performed outside of an SVN client. The **Mark as moved** action is available when you select two items from different locations (both the new one and the source one (usually reported as *missing*)) and depends on the state of the source item.

Mark as renamed

You can use this action to mark an item from the working copy as being renamed outside of an SVN client. The **Mark as renamed** action is available when you select two items from the same directory (both the new one and the source one (usually reported as *missing*)) and depends on the state of the source item.

Add to "svn:ignore" (Ctrl (Command on OS X) + Alt + I)

Allows you to keep inside your working copy files that should not participate to the version control operations. This action can only be performed on resources not under version control. It actually modifies the value of the `svn:ignore` property of the resource's parent directory. Read more about this in the *Ignore Resources Not Under Version Control* section.

Add to version control <u>(Ctrl (Command on OS X) + Alt + V)</u>

Allows you to schedule for addition resources that are not under version control. For further details, see *Add Resources to Version Control* section.

Remove from version control

Schedules selected items for deletion from repository upon the next commit. The items are not removed from the file system after committing.

Clean up <u>(Ctrl (Command on OS X) + Shift + C)</u>

Performs a maintenance cleanup operation to the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. Useful when you already know where the problem originated and want to fix it as quickly as possible. Only active for resources under version control.

Expand all <u>(Ctrl (Command on OS X) + Alt + X)</u>

Displays all descendants of the selected folder. You can obtain a similar behavior by double-clicking on a collapsed folder.

Collapse all <u>(Ctrl (Command on OS X) + Alt + Z)</u>

Collapses all descendants of the selected folder. The same behavior is obtained by double-clicking on a expanded folder.

- **Compare** menu:

Perform Files Differencing

Performs a comparison between the source and target files.

Next Block of Changes

Jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes in the document;

> **Note:** A change block groups one or more consecutive lines that contain at least one change.

Previous Block of Changes

Jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes in the document;

Next Change

Jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change.

Previous Change

Jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change.

Last Change

Jumps to the last change from the current file.

First Change

Jumps to the first change from the current file.

Copy All Non-Conflicting Changes from Right to Left

This action copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.

Copy Change from Right to Left <u>(Ctrl (Command on OS X) + Shift + Comma)</u>

This action copies the selected change from the right editor to the left editor.

Show Word Level Details

Provides a word-level comparison of the selected change.

Show Character Level Details

Provides a character-level comparison of the selected change.

Ignore Whitespaces

Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before the strings are compared they are first normalized and then the whitespace at the beginning and the end of the strings is trimmed.

- **History** menu:

Show History (Ctrl (Command on OS X) + H)

Displays the history for a SVN resource at a given revision. The resource can be one selected from the **Repositories** view, **Working Copy** view, or from the **Affected Paths** table from the **History** view, depending on which view was last focused when this action was invoked.

Show Annotation (Ctrl+Shift+A (Command+Shift+A on OS X))

Complex action that does the following operations:

- opens the selected resource in the **Annotations** editor
- displays corresponding annotations list in the **Annotations** view
- displays the history of the selected resource

Repositories

This operation is available for any resource selected from view, **Working Copy** view, **History** view or **Directory Change Sets** view, depending on which view was last focused when this action was invoked.

Revision Graph (Ctrl (Command on OS X) + G)

This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see the section *Revision Graph*. This operation is enabled for any resource selected into the **Repositories** view or **Working Copy** view.

- **Tools** menu:

Share project

Allows you to *share a new project* using an SVN repository. The local project is automatically converted into an SVN working copy.

Branch / Tag

Allows you to copy the selected resource from the **Repositories** view or **Working Copy** view to a branch or tag into the repository. To read more about this operation, see the section *Creating a Branch / Tag*.

Merge (Ctrl (Command on OS X) + J)

Allows you to merge the changes made on one branch back into the trunk, or vice versa, using the selected resource from the working copy. To read more about this operation, see the section *Merging*.

Switch (Ctrl (Command on OS X) + Alt + W)

Allows you to change the repository location of a working copy or only of a versioned item of the working copy within the same repository. It is available when the selected item of the working copy is a versioned resource, except an external folder. To read more about this action, see the section *Switching the Repository Location*.

Relocate

Allows you to change the base URL of the root folder of the working copy to a new URL, when the base URL of the repository changed, for example the repository itself was relocated to a different server. This operation is available for a selected item of the working copy tree that is a versioned folder. To read more about this operation, see the section *Relocate a Working Copy*.

Create patch (Ctrl (Command on OS X) + Alt + P)

Allows you to create a file containing all the differences between two resources, based on the `svn diff` command. To read more about creating patches, see *the section about patches*.

Working copy format

This submenu contains the following two operations:

Upgrade

Upgrades the format of the currently loaded working copy to the newest one known by Syncro SVN Client. This allows you to benefit of all the new features of the client.

Downgrade

Downgrades the format of the currently loaded working copy to SVN 1.7 format. This is useful in case you wish to use older SVN clients with the current working copy, or, by mistake, you have upgraded the format of an older working copy to SVN 1.8.

> **Note:** SVN 1.7 working copies cannot be downgraded to older formats.

See the section *Working Copy Format* to read more about this subject.

- **Options** menu:

  Preferences

  Opens the **Preferences** dialog.

  Menu Shortcut Keys

  Opens the **Preferences** dialog directly on the **Menu Shortcut Keys** option page, where users can configure in one place the keyboard shortcuts available for menu items available in Syncro SVN Client.

  Global Run-Time Configuration

  Allows you to configure SVN general options, that should be used by all the SVN clients you may use:

  - **Edit 'config' file** - In this file you can configure various SVN client-side behaviors.
  - **Edit 'servers' file** - In this file you can configure various server-specific protocol parameters, including HTTP proxy information and HTTP timeout settings.

  Export Options

  Allows you to export the current options to a file.

  Import Options

  Allows you to import options you have previously exported.

  Reset Options

  Resets all your options to the default ones.

  Reset Authentication

  Resets the Subversion authentication information.

- **Window** menu:

  Show View

  Allows you to select the view you want to bring to front.

  Show Toolbar

  Allows you to select the toolbar you want to be visible.

  Enable flexible layout

  Toggles between a fixed and a flexible layout. When the flexible layout is enabled, you can move and dock the internal views to adapt the application to different viewing conditions and personal requirements.

  Reset Layout

  Resets all the views to their default position.

- **Help** menu:

  Help (F1)

  Opens the **Help** dialog.

Dynamic Help

Shows the **Dynamic Help** view.

Check for New Versions

Checks the availability of new Oxygen XML Author versions.

Register

Opens the registration dialog.

Improvement Program Options

Allows you to activate or deactivate the Syncro Soft Product Improvement Program.

Report Problem

Opens a dialog that allows the user to write the description of a problem that was encountered while using the application.

Support Center

Opens the Support Center web page in a browser.

Main Toolbar

The toolbar of the Syncro SVN Client SVN Repositories window contains the following actions:

Check out

Checks out a working copy from a repository. The repository URL and the working copy format must be specified.

Synchronize

Synchronizes the current working copy with the repository.

Update All

Updates all resources of the working copy that have an older revision that repository.

Commit All

Commits all resources of working copy that have a newer version compared to that of the repository.

Refresh

Refreshes the whole content of the current working copy from disk starting from the root folder. At the end of the operation, the modified files and folders that were not committed to repository yet, are displayed in the **Working Copy** view.

Compare

The selected resource is compared with:

- the *BASE* revision, when the selected resource is:
  - locally modified and the **All Files** view mode is currently selected (no matter if there are incoming changes)
  - locally modified and there are no incoming changes when any other view mode is selected
- the remote version of the same resource, when remote information is available after a **Synchronize** operation (only when one of **Modified**, **Incoming**, **Outgoing** and **Conflicts** view modes is selected)
- the working copy revision, when the selected resource is from the **History** view

Show History

Displays the history of the selected resource (from the **Working Copy** or **Repository** views) in the **History** view.

Show Annotation

Displays the annotations of the selected resource. The selected resource can be in the **Working Copy** or the **History** views.

Revision Graph

Displays the revision graph of the selected resource. The selected resource can be in the **Working Copy** or the **Repositories** views.

Enable/Disable flexible layout

Toggles between a fixed and a flexible layout. When the flexible layout is enabled, you can move and dock the internal views to adapt the application to different viewing conditions and personal requirements.

Status Bar

The status bar of the Syncro SVN Client window displays important details of the current status of the application. This information is available only in the **Working Copy** view.

| usermanual/tasks/xquery-db-tranformation.dita | ⏸ ▬ ◪ | SVN 1.7 | 0 ⬅ | 2319 ⮞ | 0 ⬌ | Synchronizing… |
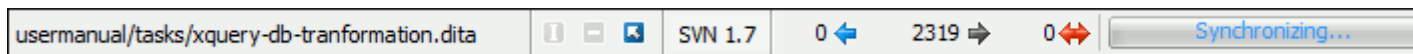
Figure 207: Status bar

The status bar is composed of the following areas:

- the path of the currently processed file from the current working copy (during an operation like **Check out** or **Synchronize**) or the result of the last operation;
- the current status of the following working copy options:

  - Show ignored files (⏸);
  - Show deleted files (▬);
  - Process svn:externals definitions (◪).

  The options for ignored and deleted files are switched on and off from *the Settings menu* of the **Working Copy** panel;
- the format of the currently loaded working copy;
- the current numbers of incoming changes (⬅), outgoing changes (⮞) and conflicting changes (⬌);
- a progress bar for the currently running SVN operation and a button (▬) that allows you to stop it.

Getting Started

This section explains the basic operations that can be done in Syncro SVN Client.

SVN Repository Location

This section explains how to add and edit the repository locations in Syncro SVN Client.

Add / Edit / Remove Repository Locations

Usually, team members do all of their work separately, in their own working copy, and must share their work. This is done using an Apache Subversion™ repository. Syncro SVN Client supports versions 1.4, 1.5, 1.6, 1.7 and 1.8 of the SVN repository format.

Before you can begin working with a Subversion repository, define a repository location in the *Repositories view*.

To create a repository location, click the **New Repository Location** toolbar button or right click inside the view and select **New Repository Location...** from the popup menu. On Windows, the context menu can be displayed on a right click with the mouse or with the keyboard by pressing the special context menu key available on Windows keyboards. This action opens the **Add SVN Repository** dialog which prompts you for the URL of the repository you want to connect to. No authentication information is requested at the time the location is defined. It is left to the Subversion client to

request the user and password information when it is needed. The main benefit of allowing Subversion to manage your password in this way is that it prompts you for a new password only when your password changes.

Once you enter the repository URL Syncro SVN Client tries to contact the server and get the content of the repository for displaying it in the *Repositories view*. If the server does not respond in the timeout interval set in preferences, an error is reported. If you do not want to wait until the timeout expires, you can end the waiting process with the ■ **Stop** button from the toolbar of the view.

To edit a repository location, click the 🔍 **Edit Repository Location** toolbar button or right click inside the view on a repository root entry and select **Edit Repository Location...** from the popup menu.

The **Edit SVN Repository** dialog works in the same way as the **Add SVN Repository** dialog. It shows the previously defined repositories URLs and it allows you to change them.

To remove a repository location, click the ✕ **Remove Repository Location** toolbar button or right click inside the view on a repository entry and select **Remove Repository Location...** from the popup menu. A confirmation dialog is displayed to make sure that you do not accidentally remove locations.

The order of the repositories can be changed in the **Repositories** view at any time with the two buttons on the toolbar of the view, the up arrow ⬆ and the down arrow ⬇. For example, pressing the up arrow once moves up the selected repository in the list with one position.

To set the reference revision number of an SVN repository right-click on the repository in the list displayed in the *Repositories view* and select the **Change the Revision to Browse...** action. The revision number of the repository is used for displaying the contents of the repository when it is viewed in the *Repositories view*. Only the files and folders that were present in the repository at the moment when this revision number was generated on the repository are displayed as contents of the repository tree. Also this revision number is used for all the file open operations executed directly from the *Repositories view*.

Authentication

Five protocols are supported: *HTTP*, *HTTPS*, *SVN*, *SVN + SSH* and *FILE*. If the repository that you are trying to access is password protected, the **Enter authentication data** dialog requests a user name and a password. If the **Store authentication data** checkbox is checked, the credentials are stored in Apache Subversion™ default directory:

- on Windows - `%HOME%\Application Data\Subversion\auth`. Example: `C:\Documents and Settings\John\Application Data\Subversion\auth`
- on Linux and OS X - `$HOME/.subversion/auth`. Example: `/home/John/.subversion/auth`

There is one file for each server that you access. If you want to make Subversion forget your credentials, you can use the **Reset authentication** command from the **Options** menu. This causes Subversion to forget all your credentials. When you reset the authentication data, restart Oxygen XML Author for the change to take effect.

ⓘ **Tip:** The *FILE* protocol is recommended if the SVN repository and Oxygen XML Author are located on the same computer as it ensures faster access to the SVN repository compared with other protocols.

For HTTPS connections where client authentication is required by your SSL server, you must choose the certificate file and enter the corresponding certificate password which is used to protect your certificate.

When using a secure HTTP (HTTPS) protocol for accessing a repository, a **Certificate Information** dialog pops up and asks you whether you accept the certificate permanently, temporarily or simply deny it.

If the repository has SVN+SSH protocol, the SSH authentication can also be made with a private key and a pass phrase.
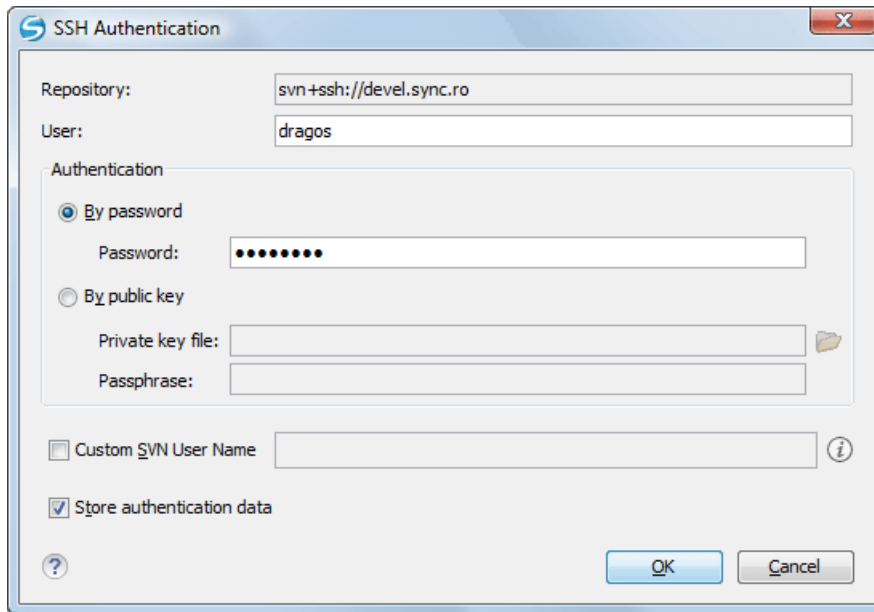
Figure 208: User & Private key authentication dialog

After the SSH authentication dialog, another dialog pops up for entering the SVN user name that accesses the SVN repository. The SVN user name is recorded as the *committer* in SVN operations.

When connecting for the first time to a Subversion repository through SVN+SSH protocol, you will be asked to confirm if you trust the SSH host. The same dialog box is also displayed when the server changed the SSH key or when the key was deleted from the local Subversion cache folder.
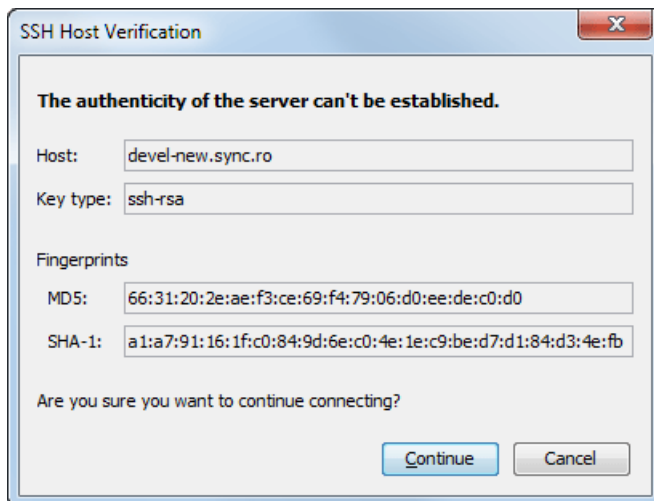
Figure 209: SSH server name and key fingerprint

Share a Project

Even if you start developing a new project or you want to migrate an existing one to Subversion, Syncro SVN Client allows you to share it easily with the rest of your team. The **Share project** action helps you do this by providing a simplified interface:
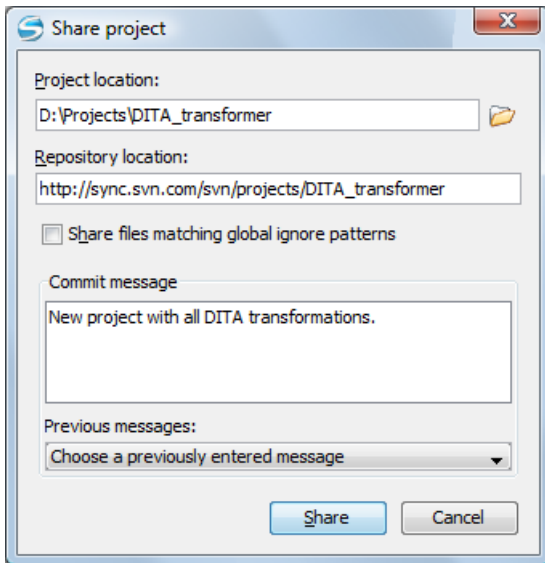
Figure 210: Share Project Dialog Box

To share a project, set the following:

*   **Project location** box - enter the location of the project on the local disk.

    📋 **Note:** The root directory of the project is not imported in the repository, only its content is imported.

*   **Repository location** box - type the repository address of the new project. If the address does not exist, it is created automatically. If it does exist, make sure that it is an empty location to avoid mixing your project content with other files.
*   **Share files matching global ignore patterns** check box - file names that match the patterns defined under the `global-ignores` property, in the Subversion clients configuration file, are also imported in the repository.

The shared project directory is automatically converted to a working copy and added under Syncro SVN Client management.

Defining a Working Copy

An Apache Subversion™ working copy is an ordinary directory tree on your local system, containing a collection of files. You can edit these files however you wish, your working copy being your private work area. In order to make your own changes available to others or incorporate other people's changes, you must explicitly tell Subversion to do so. You can even have multiple working copies of the same project.

Figure 211: Working Copy View

A Subversion working copy also contains some extra files, created and maintained by Subversion, to help it keep track of your files. In particular, each directory in your working copy contains a subdirectory named `.svn`, also known as the working copy *administrative directory*. This administrative directory contains an unaltered copy of the last updated files from the repository. This copy is usually referred to as the *pristine copy* or the *BASE revision* of the working copy. These files help Subversion recognize which files contain unpublished changes, and which files are out-of-date with respect to others' work.

A typical Subversion repository often holds the files (or source code) for several projects. Usually each project is a subdirectory in the repository's file system tree. In this arrangement, a user's working copy usually corresponds to a particular subtree of the repository.

Check Out a Working Copy

Check Out means making a copy of a project from a repository into your local file system. This copy is called a *working copy*. An Apache Subversion™ working copy is a specially formatted directory structure which contains additional `.svn` directories that store Subversion information, as well as a pristine copy of each item that is checked out.

To check out a working copy, go to the *Repositories view*. In case no connections to your repository are available, *add a new repository location* and follow these steps to check out a working copy:

1. Navigate to the desired repository directory in the **Repositories** view.
2. Right click on the directory and select **Check Out...** from the popup menu.

   The **Check Out** dialog box is displayed:
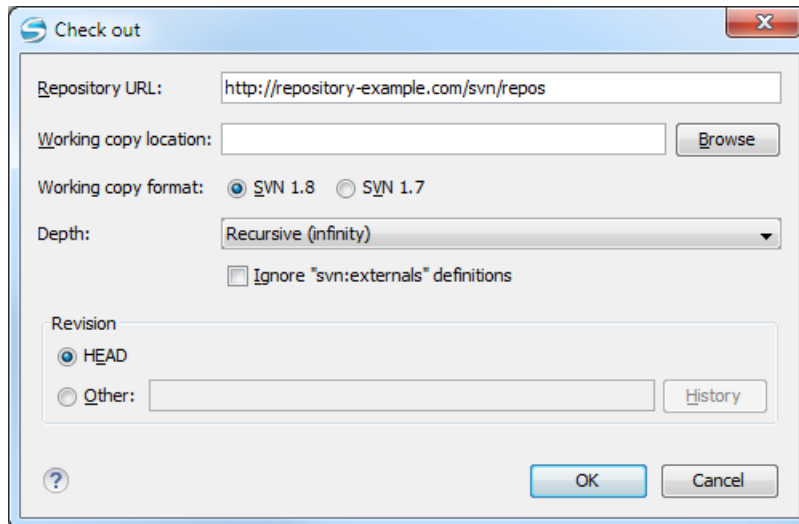
Figure 212: Check Out Dialog

3. Click the **Browse** button.
4. Select the location where the working copy is created.
5. Select the version of the working copy format.
6. Select the depth to be used when checking out the working copy;

   This allows you to specify the recursion level into child resources. The depth is used if you want to check out only a part of the selected repository directory and bring in a future update the rest of the files and subdirectories. You can find out more about the check out depth in the *sparse checkouts* section.

7. Select the revision number that is checked out.

   By default the last (HEAD) revision is checked out. If you need another revision, you have to specify the revision number or click the **History** button which opens *the History dialog*.

After a check out, the new working copy will be added to the list in the ***Working Copy** view* and loaded automatically.
The History Dialog

The **History** dialog presents a list of revisions for a resource. It is opened from the dialogs that require setting an SVN revision number like *the **Check Out** dialog* or *the **Branch / Tag** dialog* to name just a few. It presents information about revision, commit date, author, and commit comment.
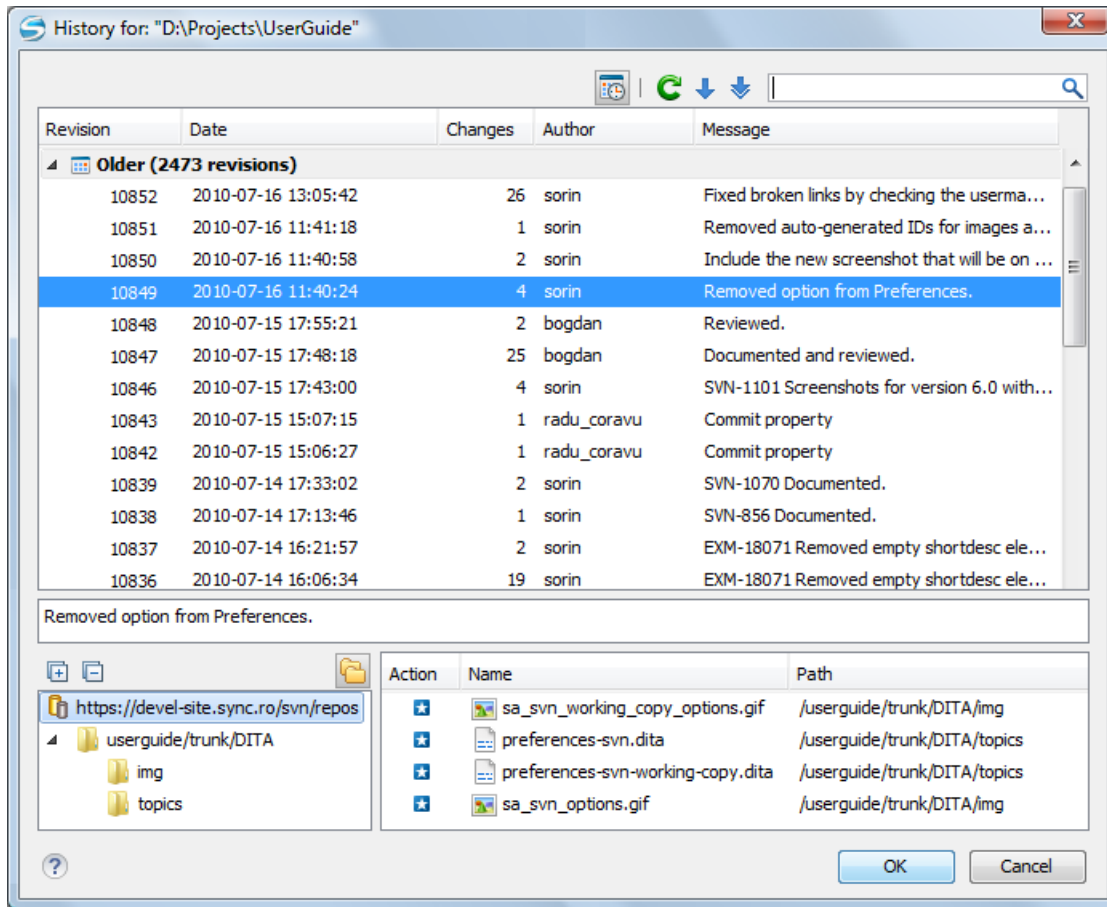
Figure 213: History Dialog

The initial number of entries in the list is 50. Additional revisions can be added to the list using the ⬇ **Get next 50** and ⬇ **Get all** buttons. The list of revisions can be refreshed at any time with the ↻ **Refresh** button. You can group revisions in predefined time frames (today, yesterday, this week, this month), by pressing the 🕘 **Group by date** button from the toolbar.

The **Affected Paths** area displays all paths affected by the commit of the revision selected in history. You can see the changes between the selected revision and the file's previous state using the **Compare with previous version** action, available in the contextual menu.

Use an Existing Working Copy

Using an existing working copy is the process of taking a working copy that exists on your file system and connecting it to Apache Subversion™ repository. If you have a brand new project that you want to import into your repository, then see the section *Import resources into the repository*. The following procedure assumes that you have an existing valid working copy on your file system.

1. Click the **Working Copies Manager** toolbar button 📁 (📁 on Mac OS X) in the *Working Copy view*.

   This action opens the **Working copies list** dialog.

2. Press the **Add** button.
3. Select the working folder copy from the file system. The name is useful to differentiate between working copies located in folders with the same name. The default name is the name of the root folder of the working copy.

   > **Note:** For SVN 1.7 and newer working copies, all the internal information is kept only in the root directory. Thus, Syncro SVN Client needs to load the whole working copy.

4. Press the **OK** button.

The selected working copy is loaded and presented in the *Working Copy view*.

⚠️ **Notice:** You can add working copies older than SVN 1.7. However, to load any of them, Syncro SVN Client will require to upgrade the working copy to SVN 1.8 format.

Manage Working Copy Resources

This section explains how to work with the resources that are displayed in the **Working Copy** view.

Edit Files

You can edit files from the *Working Copy view* by double clicking them or by right clicking them and choosing **Open** from the contextual menu.

Please note that only one file can be edited at a time. If you try to open another file, it is opened in the same editor window. The editor has syntax highlighting for known file types, meaning that a different color is used for each type of recognized token in the file. If the selected file is an image, then it is previewed in the editor, with no access to modifying it.

After modifying and saving a file from a working copy, a modified marker - an asterisk (*) - will be added to the file's icon in the *Working Copy view*. The asterisk marks the files that have local modifications that were not committed to the repository.

Add Resources to Version Control

To share new files and folders (created in your working copy), add them to version control using the **Add to version control** option from the *Working Copy view*.

You can easily spot resources not under version control by the ❓ (*unversioned*) icon displayed in the 🗒 **Local file status** column. Resources scheduled for addition (*added*) are displayed with this icon ➕ in the **Working Copy** view and are added in the repository after you commit them.

📝 **Note:** Do not make a confusion between ➕ and ➕ icons. The former icon stands for resources that are actually copies of resources already committed in the repository, meaning they are *scheduled for addition with history*.

When you use the **Add to version control** option on a directory, its entire structure is scanned and all the resources that can be added under version control are presented.

Though it is not mandatory to add resources under version control explicitly, it is recommended. If you forgot to add a resource, when you *commit your changes*, the resource is presented in the commit dialog, but not selected. When you commit and *unversioned* resource, it is automatically added under version control before starting the commit operation.
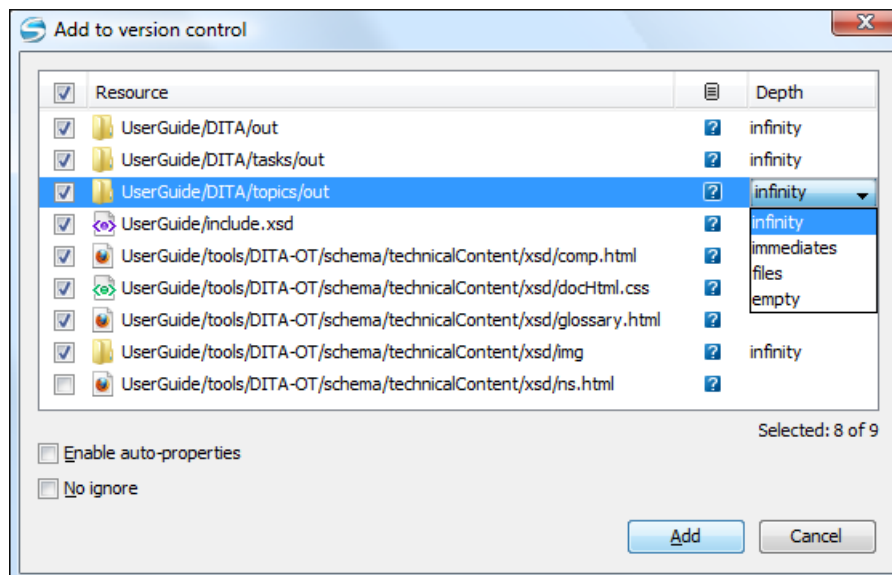
Figure 214: "Add to version control" dialog box

Note: *Ignored* (🔲) items can also be added under version control.

The **Depth** column is displayed only when directories are also presented in the dialog. For any directory, you can use one of the available values to instruct Subversion to limit the scope of the operation to a particular tree depth.

Note: The initial value of the **Depth** field can have the following values, depending on the *listing mode of the items in the working copy view*:

- *infinity* - when the working copy items are presented as a tree;
- *files* - when the working copy items are presented compressed;
- *empty* - when the working copy items are presented flat.

When you add unversioned or ignored directories, the initial value of the **Depth** field also depends on the state of the **Show unversioned directories content** and **Show ignored directories content** options. In case these options are enabled, the value is based on the listing mode of the items in the working copy view. When they are disabled, the value is *empty* .

The following options are available in this dialog box:

- **Enable auto-properties** or **Disable auto-properties** - enables or disables automatic property assignment (per runtime configuration rules), overriding the `enable-auto-props` runtime configuration directive, defined in the `config` file of the Subversion configuration directory.

  Note:

  This option is available only when there are defined properties to be applied automatically for resources newly added under version control. You can define these properties in the `config` file of the Subversion configuration directory, in the `auto-props` section.

  Based on the value of the `enable-auto-props` runtime configuration directive, the presented option is either **Enable auto-properties**, or **Disable auto-properties**.

- **No ignore** - when you enable this option, file-name patterns defined to ignore *unversioned* resources do not apply. Resources that are located inside an *unversioned* directory selected for addition, and match these patterns, are also scheduled for addition in the repository.

  Note:

  This option is available only when directories are also presented in the dialog.

  You can define file-name patterns to ignore *unversioned* resources in one of the following locations:

  - in the `config` file of the Subversion configuration directory (the `global-ignores` option from the `miscellany` section);
  - in the Oxygen XML Author options: *open the Preferences dialog* and go to **SVN** > **Working copy** > **Application global ignores**.

Each of the above two options is activated only when you select an item for which the option can be applied.

Ignore Resources Not Under Version Control

Some resources inside your working copy do not need to be subject to version control. These resources can be files created by the compiler, `*.obj`, `*.class`, `*.lst`, or output folders used to store temporary files. Whenever you *commit changes*, Apache Subversion™ shows your modified files but also the unversioned files, which fill up the file list in the commit dialog. Though the unversioned files are committed unless otherwise specified, it is difficult to see exactly what you are committing.

The best way to avoid these problems is to add the derived files to the Subversion's ignore list. That way they are never displayed in the commit dialog and only genuine unversioned files which must be committed are shown.

You can choose to ignore a resource by using the **Add to svn:ignore** action in the contextual menu of the *Working Copy view*.

In the **Add to svn:ignore** dialog you can specify the resource to be ignored by name or by a custom pattern. The custom pattern can contain the following wildcard characters:

- * - Matches any string of characters of any size, including the empty string.
- ? - Matches any single character.

For example, you can choose to ignore all text documents by using the pattern: `*.txt`.

The action **Add to svn:ignore** adds a predefined Subversion property called `svn:ignore` to the parent directory of the specified resource. In this property, there are specified all the child resources of that directory that must be ignored. The result is visible in the **Working Copy** view. The ignored resources are represented with grayed icons.

Delete Resources

The ✖ **Delete** action is available in the contextual menu of the *Working Copy view*. When you delete an item from the working copy, it is marked as *deleted* (scheduled for deletion from repository upon the next commit) and removed from the file system. Depending on the state of each item, you are prompted to confirm the operation.

If a resource is deleted from the file system without Subversion's knowledge, the resource is marked as *missing* (■) in your working copy. You can decide what you want to do with a *missing* item:

- in case of a commit, any *missing* item is first deleted automatically and then committed.

    > **Note:** Not any *missing* item can be committed as *deleted*, and removed from the repository. For example, you cannot commit an item that no longer exists on the disk and that was scheduled for addition (■) previously, since this item does not exist in the repository, but you can use the **Delete** action instead.

- in case you want to recover *missing* items, either *update* the items themselves or one of their parent directories. This fetches their latest version from the repository.

You can also delete conflicting items (file content conflicts, property conflicts, tree-conflicts) and Syncro SVN Client automatically marks them as resolved.

> **Note:** It is recommended that you resolve conflicts manually to avoid loosing any important remote modifications.

Finally, you can change your mind and *revert* the deleted items to their initial, pristine, state.

Copy Resources

You can copy several resources from different locations of the working copy. You select them in the *Working Copy view* and then use **Copy to** from the contextual menu. This is not a simple file system copy, but an Apache Subversion™ command. It will copy the resource and the copy will also have the original resource's history. This is one of Subversion's very important features, as you can keep track of where the copied resources originated.

Based on the selected items, the **Copy to** action is enabled only if it can be performed. Even if the operation would not normally be possible in SVN (due to some invalid local file states against copy), Oxygen XML Author performs the copy operation as a simple file system operation. This means no SVN versioning meta-data is affected.

> **Note:**
> - In case you copy an item to a directory that is *not under version control* (*unversioned* or *ignored*), the history of the item is not preserved. For example, when copying directories, all items inside them will also be copied without history.
> - In case you copy a directory that contains *external* items, these are not copied. This is specific for SVN 1.7 working copies only. To fetch the *external* items, use the **Update** operation on the copied directory.

In the **Copy to** dialog you can navigate through the working copy directories in order to choose a target directory, to copy inside it. If you try to copy a single resource you are also able to change that resource's name. For *versioned* items, you can select **Ignore resource history** to copy them without their history (similar to a simple file system copy).

> **Note:** The **Copy to** dialog only presents all the local directories that are a valid destination against the copy operation, based on their local file status. Also, the *working copy settings* are taken into account.

In the **Commit** dialog will appear only the directory in question without its children.

Move Resources

As in the case of the copy command, you can move several resources at once. Select the resources in the ***Working Copy view*** and choose the **Move to** action from the contextual menu. The move command actually behaves as if a copy followed by a delete command were issued. You will find the moved resources at the desired destination and also at their original location, but marked as *deleted*.

> **Note:** *External* items cannot be moved using the **Move to** action, because they cannot be deleted. Instead, you should edit the `svn:externals` property defining the *external* item and use the **Update** operation on the item's parent folder for the change to take effect.

> **Attention:** For SVN 1.8 working copies: when committing items that were moved and/or renamed, make sure you select both the source and the destination, otherwise the commit operation will fail.

Rename Resources

The **Rename** action is available in the contextual menu of the ***Working Copy view*** and can be performed on a single resource. This action acts as a move command with the destination directory being the same as the original location of the resource. A copy of the original item is created with the new name, also keeping its history. The original item is marked as *deleted*.

> **Note:** *External* items cannot be renamed using the **Rename** action because they cannot be deleted. Instead, you should edit the *svn:externals* property defining the *external* item, then use the **Update** operation on the item's parent folder for the change to take effect.

> **Attention:** For SVN 1.8 working copies: when committing items that were moved and/or renamed, make sure you select both the source and the destination, otherwise the commit operation will fail.

Lock / Unlock Resources

The idea of version control is based on the *copy-modify-merge* model of file sharing. This model states that each user contacts the repository and creates a local working copy (check out). Users can then work independently and modify their working copies as they please. When their goal has been accomplished, it is time for the users to share their work with the others, to send them to the repository (commit). When a user has modified a file that has been also modified on the repository, the two files will have to be merged. The version control system assists the user with the merging as much as it can, but in the end the user is the one that must make sure it is done correctly.

The copy-modify-merge model only works when files are contextually mergeable: this is usually the case of line-based text files (such as source code). However this is not always possible with binary formats, such as images or sounds. In these situations, the users must each have exclusive access to the file, ending up with a *lock-modify-unlock* model. Without this, one or more users could end up wasting time on changes that cannot be merged.

An SVN lock is a piece of metadata which grants exclusive access to a user. This user is called the lock owner. A lock is uniquely identified by a lock token (a string of characters). If someone else attempts to commit the file (or delete a parent of the file), the repository demands two pieces of information:

- User authentication - the user performing the commit must be the lock owner;
- Software authorization - the user's working copy must have the same lock token as the one from the repository, proving that it is the same working copy where the lock originated from.

Scanning for Locks

When starting to work on a file that is not contextually mergeable (usually a binary file), it is better to verify if someone else is not already working on that file. You can do this in the ***Working Copy view*** by selecting one or more resources, then right clicking on them and choosing the **Scan for Locks** action from the context menu.
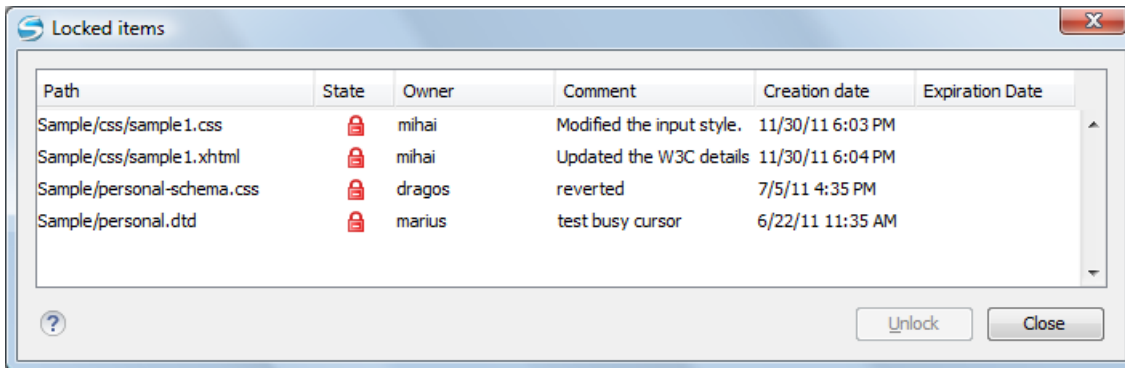
Figure 215: The locked items dialog

The **Locked items** dialog contains a table with all the resources that were found locked on the repository. For each resource there are specified: resource path, state of the lock, owner of the lock, lock comment, creation and expiration date for the lock (if any).

The state of the lock can be one of:

- 🔒 - shown when:

  - another user has locked the file in the repository;
  - the file was locked by the same user from another working copy;
  - the file was locked from the **Repositories** view.

- 🔒 - displayed after you have locked a file from the current working copy.
- 🔓 - a file already locked from your working copy is no longer locked in the repository (it was unlocked by another user).
- 🔒 - a file already locked from your working copy is being locked by another user. Now the owner of the file lock is the user who stole the lock from you.

You can unlock a resource by selecting it and pressing the **Unlock** button.

Locking a File

By locking a file you have exclusive write access over it in the repository.

You can lock a file from your working copy or directly from the **Repositories** view. Note that you can lock only files, but no directories. This is a restriction imposed by Apache Subversion™.

The **Lock** dialog allows you to write a comment when you set a lock or when you steal an existing one. Note that you should steal a lock only after you made sure that the previous owner no longer needs it, otherwise you may cause an unsolvable conflict which is exactly why the lock was put there in the first place. The Subversion server can have a policy concerning lock stealing, as it may not allow you to do this if certain conditions are not met.

The lock stays in place until you unlock the file or until someone breaks it. There is also the possibility that the lock expires after a period of time specified in the Subversion server policy.

Unlocking a File

A file can be unlocked from the contextual menu of the *Working Copy view*. A dialog will prompt you to confirm the unlocking and it will also allow you to break the lock (unlock it by force).

Synchronize with Repository

In the work cycle you will need to incorporate other people's changes (update) and to make your own work available to others (commit). This is what the **Incoming** and **Outgoing** modes of *the Working Copy view* was designed for, to help you send and receive modifications from the repository.

The **Incoming** and **Outgoing** modes of this view focus on incoming and outgoing changes. The incoming changes are the changes that other users have committed in the repository since you last updated your working copy. The outgoing changes are the modifications you made to your working copy as a result of editing, removing or adding resources.

The view presents the status of the working copy resources against the BASE revision after a **Refresh** operation. You can view the state of the resources versus a repository HEAD revision by using the **Synchronize** action from *the Working Copy view*.

View Differences

One of the most common requirements in project development is to see what changes have been made to the files from your Working Copy or to the files from the repository. You can examine these changes after a synchronize operation with the repository, by using the **Open in compare editor** action from the contextual menu.

The text files are compared using a built-in *Compare view* which uses a line differencing algorithm or a specified external diff application if such an application is *set in the SVN preferences*. When a file with outgoing status is involved, the compare is performed between the file from the working copy and the BASE revision of the file. When a file with incoming or conflict status is involved, the differences are computed using a three-way algorithm which means that the local file and the repository file are each compared with the BASE revision of the file. The results are displayed in the same view. The differences obtained from the local file comparison are considered outgoing changes and the ones obtained from the repository file comparison are considered incoming changes. If any of the incoming changes overlap outgoing changes then they are in conflict.

A special case of difference is a *diff pseudo-conflict*. This is the case when the left and the right sections are identical but the BASE revision does not contain the changes in that section. By default this type of changes are ignored. If you want to change this you can go to *SVN Preferences* and change the corresponding option.

The right editor of the internal compare view presents either the BASE revision or a revision from the repository of the file so its content cannot be modified. By default when opening a synchronized file in the **Compare** view, a compare is automatically performed. After modifying and saving the content of the local file presented in the left editor, another compare is performed. You will also see the new refreshed status in the *Working Copy view*.
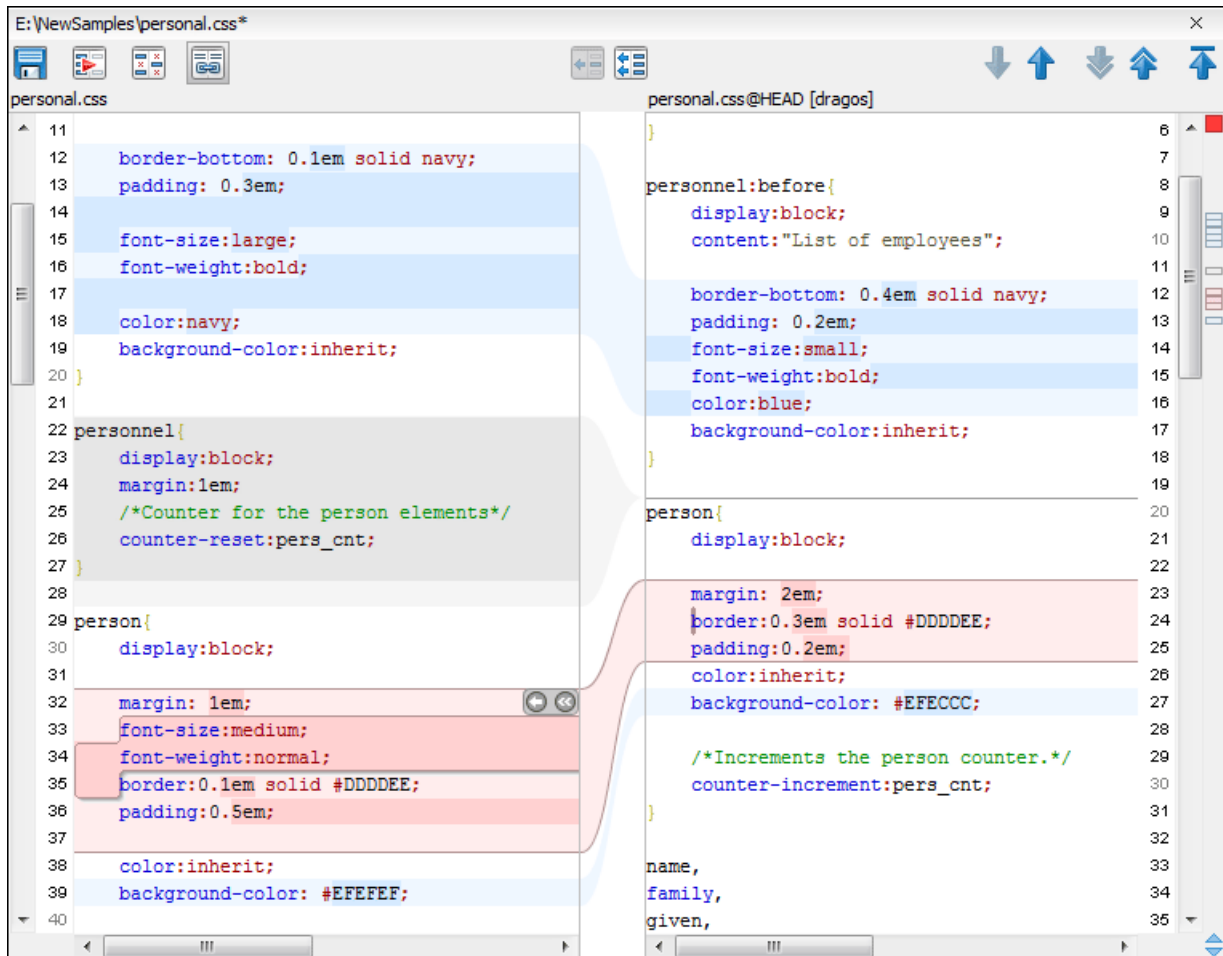
Figure 216: Compare View

At the top of each of the two editors, there are presented the name of the opened file, the corresponding SVN revision number (for remote resources) and the author who committed the associated revision.

There are three types of differences:

- incoming changes - Changes committed by other users and not present yet in your working copy file. They are marked with a blue highlight and on the middle divider the arrows point from right to left.
- outgoing changes - Changes you have done in the content of the working copy file. They are marked with a gray highlight and the arrows on the divider are pointing from left to right.
- conflicting changes - This is the case when the same section of text which you already modified in the local file has been modified and committed by some other person. They are marked with a red highlight and red diamonds on the divider.

There are numerous actions and options available in the *Compare View toolbar* or in the **Compare** menu from the main menu. You can decide that some changes need adjusting or that new ones must be made. After you perform the adjustments, you may want to perform a new compare between the files. For this case there is an action called **Perform files differencing**. After each files differencing operation the first found change will be selected. You can navigate from one change to another by using the actions **Go to first**, **Go to previous**, **Go to next** and **Go to last modification**. If you decide that some incoming change needs to be present in your working file you can use the action **Copy change from right to left**. This is useful also when you want to override the outgoing modifications contained in a conflicting section. The action **Copy all non-conflicting changes from right to left** copies all incoming changes which are not contained inside a conflicting section in your local file.

Let us assume that only a few words or letters are changed. Considering that the differences are performed taking into account whole lines of text, the change will contain all the lines involved. For finding exactly what words or letters have

changed there are available two dialogs which present a more detailed compare result when you double click on the middle divider of a difference: **Word Details** and **Character Details**.

When you want to examine only the changes in the real text content of the files disregarding the changes in the number of white spaces between words or lines there is available an option in the *SVN Preferences* which allows you to enable or disable the white space ignoring feature of the compare algorithm.

Conflicts

A file conflict occurs when two or more developers have changed the same few lines of a file or the properties of the same file. As Subversion knows nothing of your project, it leaves resolving the conflicts to the developers. Whenever a conflict is reported, you should open the file in question, and try to analyse and resolve the conflicting situation.

Real Conflicts vs Mergeable Conflicts

There are two types of conflicts:

- *real conflict* (decorator in *Name* column) - Syncro SVN Client considers the following resource states to be real conflicts:

  - *conflicted* state - a file reported by SVN as being in this state is obtained after it was updated/merged while having incoming and outgoing content or property changes at the same time, changes which could not be merged. A content conflict (symbol in *Local file status* column) is reported when the modified file has binary content or it is a text file and both local and remote changes were found on the same line. A properties conflict (symbol in *Local properties status* column) is reported when a property's value was modified both locally and remotely;

  - *tree conflicted* state (symbol in *Local file status* column) - obtained after an update or merge operation, while having changes at the directory structure level (for example, file is locally modified and remotely deleted or locally scheduled for deletion and remotely modified);

  - *obstructed* state (symbol in *Local file status* column) - obtained after a resource was versioned as one kind of object (file, directory, symbolic link), but has been replaced outside Syncro SVN Client by a different kind of object.

- *pseudo-conflict* (decorator in *Name* column) - a file is considered to be in *pseudo-conflict* when it contains both incoming and outgoing changes. When incoming and outgoing changes do not intersect, an update operation may automatically merge the incoming file content into the existing locally one. In this case, the *pseudo-conflict* marker is removed. This marker is used only as a warning which should prevent you to run into a real conflict.

> **Note:**
>
> - A conflicting resource cannot be committed to repository. You have to resolve it first, by using **Mark Resolved** action (after manually editing/merging file contents) or by using **Mark as Merged** action (for pseudo-conflicts).
>
> - and decorators are presented only when one of the following view modes is selected: **Modified**, **Incoming**, **Outgoing**, **Conflicts**.
>
> - The marker is used also for folders to signal that they contain a file in real conflict or pseudo-conflict state.

Content Conflicts vs Property Conflicts

A *Content conflict* appears in the content of a file. A merge occurs for every inbound change to a file which is also modified in the working copy. In some cases, if the local change and the incoming change intersect each other, Apache Subversion™ cannot merge these changes without intervention. So if the conflict is real when updating the file in question the conflicting area is marked like this:

```
<<<<<<< filename
your changes
=======
code merged from repository
>>>>>>> revision
```

Also, for every conflicted file Subversion places three additional temporary files in your directory:

- `filename.ext.mine` - This is your file as it existed in your working copy before you updated your working copy, that is without conflict markers. This file has your latest changes in it and nothing else.
- `filename.ext.rOLDREV` - This is the file that was the BASE revision before you updated your working copy, that is the file revision that you updated before you made your latest edits.
- `filename.ext.rNEWREV` - This is the file that Subversion client just received from the server when you updated your working copy. This file corresponds to the HEAD revision of the repository.

OLDREV and NEWREV are revision numbers. If you have conflicts with binary files, Subversion does not attempt to merge the files by itself. The local file remains unchanged (exactly as you last changed it) and you will get `filename.ext.r*` files also.

A *Property conflict* is obtained when two people modify the same property of the same file or folder. When updating such a resource a file named `filename.ext.prej` is created in your working copy containing the nature of the conflict. Your local file property that is in conflict will not be changed. After resolving the conflict you should use the **Mark resolved** action in order to be able to commit the file. Note that the **Mark resolved** action does not really resolve the conflict. It just removes the conflicted flag of the file and deletes the temporary files.

Edit Real Content Conflicts

The conflicts of a file in the conflicted state (a file with the red double arrow icon) can be edited visually with the **Compare** view (the built-in file diff tool) or with an *external diff application*. Resolving the conflict means deciding for each conflict if the local version of the change will remain or the remote one instead of the special conflict markers inserted in the file by the SVN server.

The **Compare** view (or the external diff application *set in Preferences*) is opened with the action **Edit Conflict** which is available on the contextual menus of *the Working Copy view* and is enabled only for files in the conflicted state (an update operation was executed but the differences could not be merged without conflicts). The external diff application is called with 3 parameters because it is a 3-way diff operation between the local version of the file from the working copy and the HEAD version from the SVN repository with the BASE version from the working copy as common ancestor.

If *the option Show warning dialog when edit conflicts is enabled* you will be warned at the beginning of the operation that the operation will overwrite the conflict version of the file received from the SVN server (the version which contains the conflict markers <<<<<<<, =======, >>>>>>>) with the original local version of the file that preceded the update operation. If you press the OK button the visual conflict editing will proceed and a backup file of the conflict version received from the SVN server is created in the same working copy folder as the file with the edited conflicts. The name of the backup file is obtained by appending the extension `.sync.bak` to the file as stored on the SVN server. If you press the **Cancel** button the visual editing will be aborted.

The usual actions on the differences between two versions of a file are available on the toolbar of this view:

Save

Saves the modifications of the local version of the file displayed in the left side of the view.

Perform Files Differencing

Applies the diff operation on the two versions of the file displayed in the view. It is useful after modifying the local version displayed in the left side of the view.

Go to First Modification

Scrolls the view to the topmost difference.

Go to Previous Modification

Scrolls the view to the previous difference. The current difference is painted with a darker color than the other ones.

Go to Next Modification

Scrolls the view to the next difference. The current difference is painted with a darker color than the other ones.

Go to Last Modification

Scrolls the view to the last difference.

Copy All Non Conflicting Changes from Left to Right

Not applicable for editing conflicts so it is disabled.

Copy Change from Left to Right

Not applicable for editing conflicts so it is disabled.

Copy Change from Right to Left

Copies the current difference from the left side to the right side by replacing the highlighted text of the current difference from the left side with the one from the right side.

Copy All Non Conflicting Changes from Right to Left

Applies the previous operation for all the differences.

Show Modification Details at Word Level

Displays a more detailed version of the current difference computed at word level.

Show Modification Details at Char Level

Displays a more detailed version of the current difference computed at character level.

Ignore Whitespaces

The text nodes are normalized before computing the difference so that if two text nodes differ only in whitespace characters they are reported as equal.

The operation begins by overwriting the conflict version of the file received from the SVN server (the version which contains the conflict markers <<<<<<<, =======, >>>>>>>) with the original local version of the file before running the update action which created the conflict. After that the differences between this original local version and the repository version are displayed in the **Compare** view.

If you want to edit the conflict version of the file directly in a text editor instead of the visual editing offered by the **Compare** view you should work on the local working copy file after the update operation without running the action **Edit Conflict**. If you decide that you want to edit the conflict version directly after running the action **Edit Conflict** you have to work on the `.sync.bak` file.

If you did not finish editing the conflicts in a file at the first run of the action **Edit Conflict** you can run the action again and you will be prompted to choose between resuming the editing where the previous run left it and starting again from the conflict file received from the SVN server.

After the conflicts are edited and saved in the local version of the file you should run:

- either the action **Mark Resolved** on the file so that the result of the conflict editing process can be committed to the SVN repository,
- or the action **Revert** so that the repository version overwrites all the local modifications.

Both actions remove the backup file and other temporary files created with the conflict version of the local file.

Revert Your Changes

If you want to undo changes made in your working copy, since the last update, select the items you are interested in, right click to display the contextual menu and select **Revert**. A dialog will pop up showing you the files and folders that you have changed and can be reverted. Select those you want to revert and click the **OK** button. Revert will undo only your local changes. It does not undo any changes which have already been committed. If you choose to revert a conflicting item to its pristine copy, then the eventual conflict is solved by losing your outgoing modifications. If you try to revert a resource not under version control, the resource will be deleted from the file system.

> **Note:** By default, a directory will be recursively reverted (including any other modified item it contains). However, if the directory has only property changes, you need to explicitly choose if the operation will include any modified items found inside it.

If you want some of your outgoing changes to be overridden you must first open the file in *Compare view* and choose the sections to be replaced with ones from the repository file. This can be achieved either by editing directly the file or by using the action **Copy change from right to left** from the *Compare view toolbar*. After editing the conflicting file you have to run the action **Mark as merged** before committing it.

If you want to drop all local changes and, at the same time, bring all incoming changes into your working copy resource, you can use the **Override and update** action which discards the changes in the local file and updates it from the repository. A dialog will show you the files that will be affected.
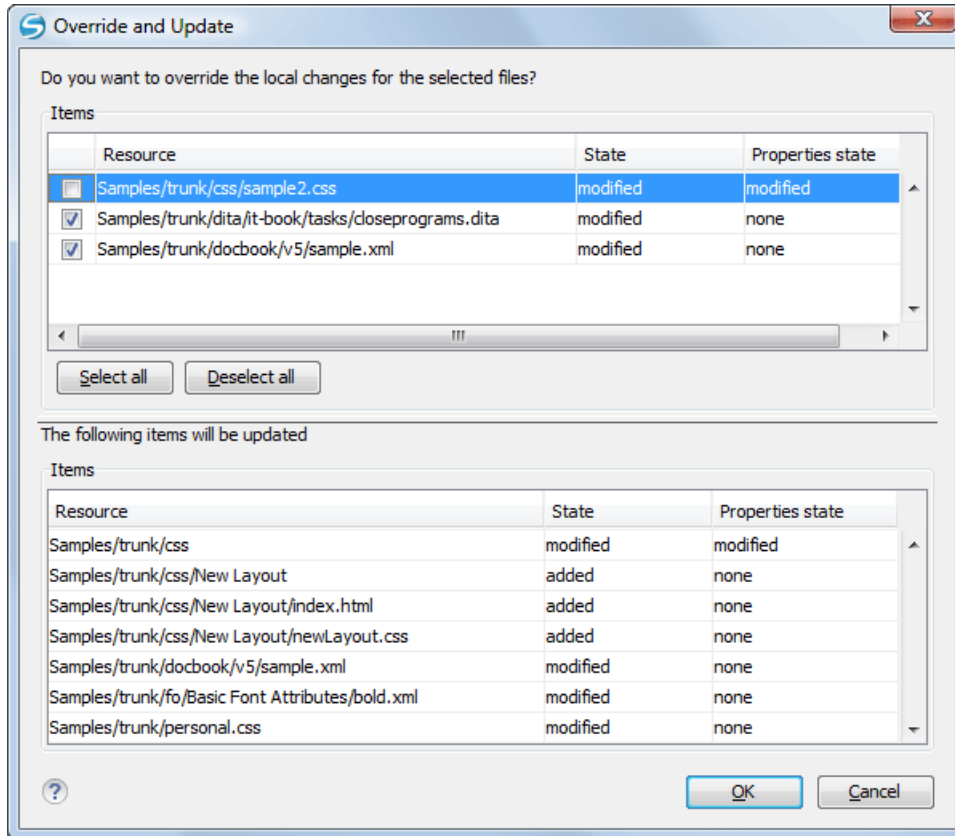
Figure 217: Override and update dialog

In the first table of the dialog you will be able to see the resources that will be overridden. In the second table you will find the list of resources that will be updated. Only resources that have an incoming status are updated.

> 🛈 **Tip:** If you want to roll-back out of your working copy changes that have already been committed to the repository, see *Merge Revisions*.

Merge Conflicted Resources

Before you can safely commit your changes to the repository you must first resolve all conflicts. In the case of pseudo-conflicts they can be resolved in most cases with an update operation which will merge the incoming modifications into your working copy resource. In the case of real conflicts, conflicts that persist after an update operation, it is necessary to resolve the conflict using the built-in compare view and editor or, in the case of properties conflict, the *Properties view*. Before you can commit you must *mark as resolved* the affected files.

Both pseudo and real conflicts can be resolved without an update. You should open the file in the compare editor and decide which incoming changes need to be copied locally and which outgoing changes must be overridden or modified. After saving your local file you have to use the *Mark as merged* action from the contextual menu before committing.

Drop Incoming Modifications

In the situation when your file is in conflict but you decide that your working copy file and its content is the correct one, you can decide to drop some or all of the incoming changes and commit afterwards. The action **Mark as merged** proves to be useful in this case too. After opening the conflicting files with *Compare view*, *Editor* or editing their properties in the **Properties** view and deciding that your file can be committed in the repository replacing the existing one, you should use the **Mark as merged** action. When you want to override completely the remote file with the local file you should run the action **Override and commit** which drops any remote changes and commits your file.

In general it is much safer to analyze all incoming and outgoing changes using the **Compare** view and only after to update and commit.

Tree Conflicts

A *tree conflict* is a conflict at the directory tree structure level and occurs when the user runs an update action on a resource that:

- it is locally modified and the same resource was deleted from the repository (or deleted as a result of being renamed or moved);
- it was locally deleted (or deleted as a result of being renamed or moved) and the same resource is incoming as modified from the repository.

The same conflict situation can occur after a merge or a switch action. The action ends with an error and the folder containing the file that is now in the tree conflict state is also marked with a conflict icon.

Such a conflict can be resolved in one of the following ways which are available when the user double clicks on the conflicting resource or when running the **Edit conflict** action:
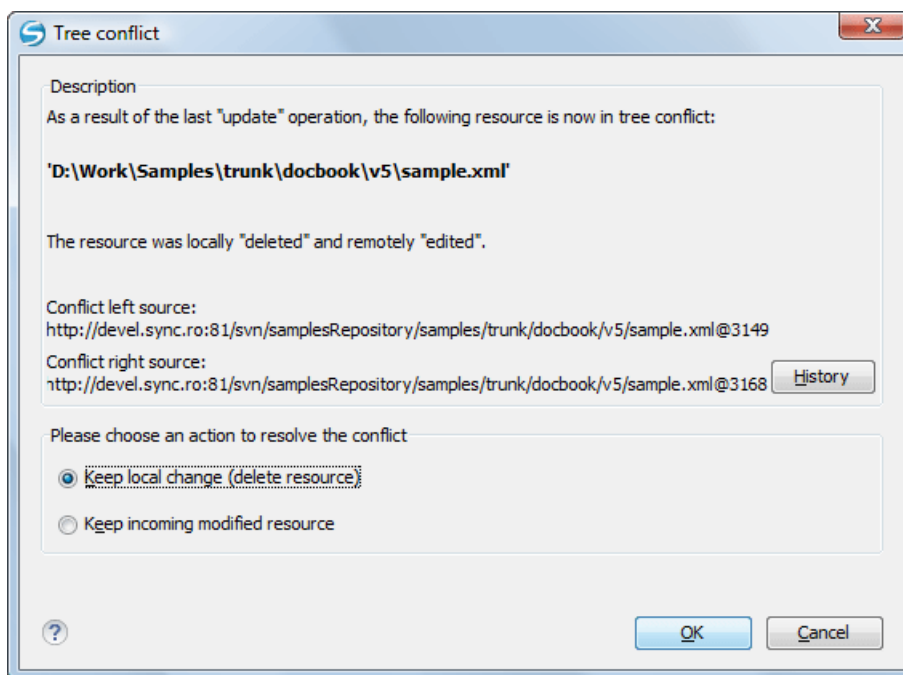
Figure 218: Resolve a tree conflict

- Keep the local modified file - If there is a renamed version of the file committed by other user that will be added to the working copy too.
- Delete the local modified file - Keeps the incoming change that comes from the repository.

Update the Working Copy

While you are working on a project, other members of your team may be committing changes to the project repository. To get these changes, you have to *update* your working copy. Updating may be done on single files, a set of selected files, or recursively on entire directory hierarchies. The update operation can be performed from *Working Copy view*. It updates the selected resources to the last synchronized revision (if remote information is available) or to the *HEAD* revision of the repository.

There are three different kinds of incoming changes:

- *Non-conflicting* - A non-conflicting change occurs when a file has been changed remotely but has not been modified locally.

- *Conflicting, but auto-mergeable* - An auto-mergeable conflicting change occurs when a text file has been changed both remotely and locally (i.e. has non-committed local changes) but the changes are on different lines of text. Not applicable to binary resources (for example multimedia files, PDFs, executable program files)
- *Conflicting* - A conflicting change occurs when one or more of the same lines of a text file have been changed both remotely and locally.

If the resource contains only incoming changes or the outgoing changes do not intersect with incoming ones then the update will end normally and the Subversion system will merge incoming changes into the local file. In the case of a conflicting situation the update will have as result a file with conflict status.

The Oxygen XML Author allows you to update your working copy files to a specific revision, not only the most recent one. This can be done by using the **Update to revision/depth** action from the **Working Copy** view (**All Files** view mode) or the **Update to revision** action from the *History view* contextual menu.

If you select multiple files and folders and then you perform an **Update** operation, all of those files and folders are updated one by one. The Subversion client makes sure that all files and folders belonging to the same repository are updated to the exact same revision, even if between those updates another commit occurred.

When the update fails with a message saying that there is already a local file with the same name Subversion tried to check out a newly versioned file, and found that an unversioned file with the same name already exists in your working folder. Subversion will never overwrite an unversioned file unless you specifically do this with an **Override and update** action. If you get this error message, the solution is simply to rename the local unversioned file. After completing the update, you can check whether the renamed file is still needed.

Send Your Changes to the Repository

Sending the changes you made to your working copy is known as *committing* the changes. If your working copy is up-to-date and there are no conflicts, you are ready to commit your changes.

The **Commit** action sends the changes from your local working copy to the repository. The **Commit** dialog box presents all the items that you are able to commit.
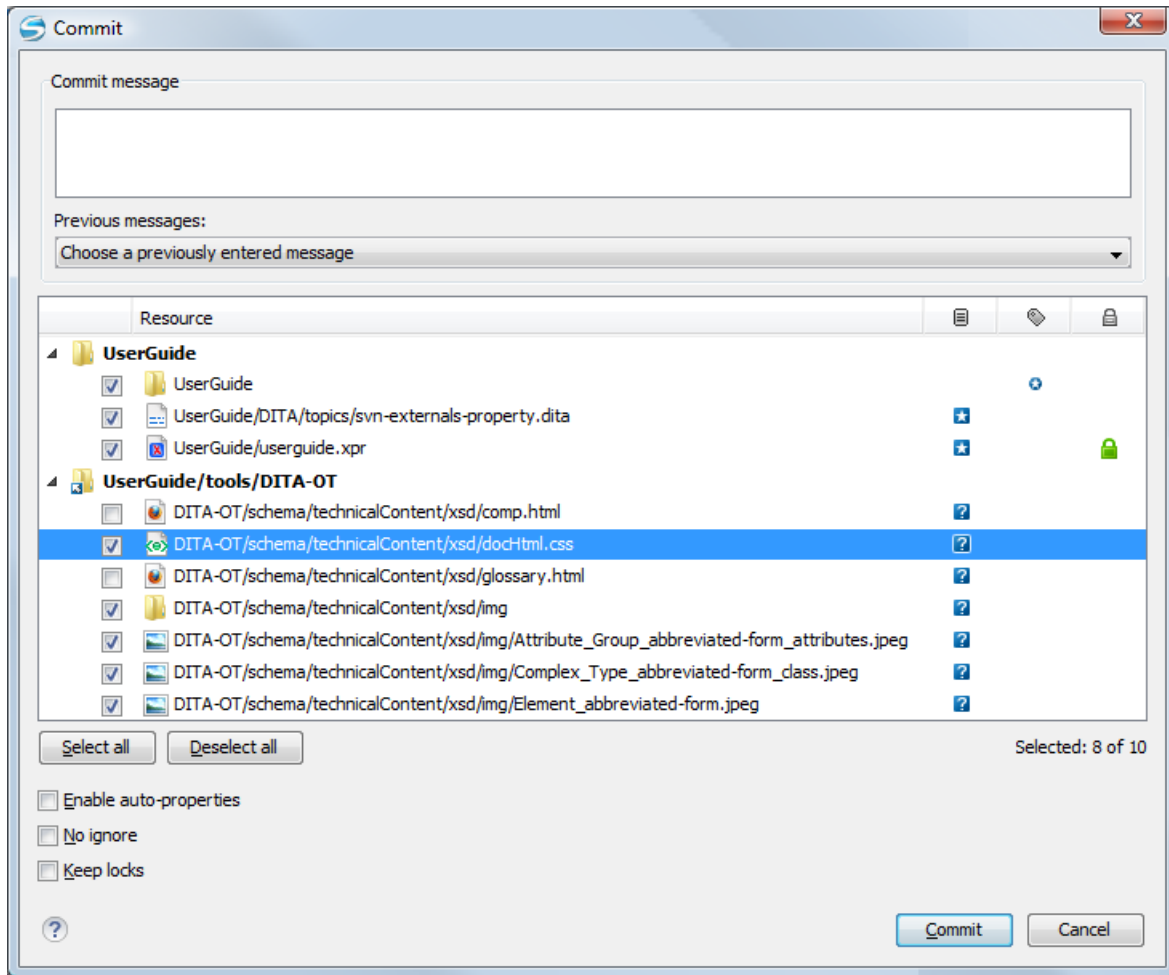
Figure 219: Commit dialog box

Enter a message to associate with the commit, or choose a previous message from the **Previous messages** list (the last 10 commit messages will be remembered even after restarting the SVN client application).

An item that can be committed has one of the following states: *added*, *modified* (content or properties), *replaced*, and *deleted*. All items that have one of these states are selected in the dialog box by default. If you do not want to commit one of the items, uncheck it.

⚠ **Attention:** For SVN 1.8 working copies: when committing items that were moved and/or renamed, make sure you select both the source and the destination, otherwise the commit operation will fail.

Besides the items that have one of the mentioned states, Syncro SVN Client also includes the files being *unversioned* or *missing*. In order to be committed, these items are handled automatically:

• *unversioned* items are added under version control;
• *missing* items are deleted.

📄 **Note:** In case the **Show unversioned directories content** is disabled, the **Commit** dialog box does not display the items inside an *unversioned* directory.

Unversioned or *missing* items are not selected by default in the **Commit** dialog box, unless you have selected them explicitly when issuing the commit command.

Note: In some cases, items that have one of the above states are not presented in the **Commit** dialog.

For example:

- items that have been *added* or *replaced* previously, but now are presented as *missing* after being removed from the file system, outside of an SVN client. Such items do not exist in the repository and you should use the **Delete** action to remove them from your working copy;
- items that have incoming changes from the repository, after a synchronization. You need to have your working copy up-to-date before committing your changes;
- files that, after a synchronization, appear as locked by other users or from other locations than the current working copy.

Note: Due to dependencies between items, when you select or clear an *unversioned* (▨) or *added* (▣) item in the **Commit** dialog box, other items with one of these states can be selected or cleared automatically.

The modifications that will be committed for each file can be reviewed in the compare editor window by double clicking a file in the **Commit** dialog box, or by right clicking and selecting the **Show Modifications** action from the contextual menu. This option is available to review only file content changes, not property changes.

The ▤ **Local file status** column indicates the actual state of the items and the ◉ **Local properties status** column indicates whether the properties of an item are modified.

The ▣ **Lock information** column is displayed in case at least one of the files in the **Commit** dialog box has lock information associated with it, valid against the commit operation.

The following options are available in this dialog box:

- **Enable auto-properties** or **Disable auto-properties** - enables or disables automatic property assignment (per runtime configuration rules), overriding the enable-auto-props runtime configuration directive, defined in the config file of the Subversion configuration directory.

  Note:

  This option is available only when there are defined properties to be applied automatically for resources newly added under version control. You can define these properties in the config file of the Subversion configuration directory, in the auto-props section.

  Based on the value of the enable-auto-props runtime configuration directive, the presented option is either **Enable auto-properties**, or **Disable auto-properties**.

- **Keep locks** - selecting the **Keep locks** option preserves any locks you set on various files.

  Note: This option is available only when files that you locked are presented in the dialog box.

Each of the above options is activated only when you select an item for which the option can be applied.

Your working copy must be up-to-date with respect to the resources you commit. This is ensured by using the **Update** action prior to committing, resolving conflicts and re-testing as needed. If your working copy resources you are trying to commit are out of date you will get an appropriate error message.

Committing to Multiple Locations

Although Subversion does not support committing to different locations at once, Syncro SVN Client offers this functionality regarding *external* items.

If items to be committed belong to different *external* definitions found in the working copy, they are grouped under the corresponding item that indicates their repository origin. Each parent item is rendered bold and its corresponding repository location is presented when hovering it. Parent items are decorated with a small arrow (▣) if they are *external* definitions. The working copy root directory is never decorated and is not presented if there are no *external* items listed (all items belong to the main working copy). Each child item is presented relative to the parent item.

📄 **Note:** When an *external* directory has modifications of its own, it is presented both as a parent item and as an item that you can select and commit. This is always the case for *external* files.

The sets of items belonging to *external* definitions from the same repository are committed together, resulting a single revision. So, the number of revisions can be smaller than the number of *externals*. External definitions are considered from the same repository if they have the same protocol, server address, port, and repository address within the server.

📄 **Note:** *External* files are always from the same repository as the parent directory which defines them, so they are always committed together with the changes from their parent directory.

Integration with Bug Tracking Tools

Users of bug tracking systems can associate the changes they make in the repository resources with a specific ID in their bug tracking system. The only requirement is that the user includes the bug ID in the commit message that he enters in the **Commit** dialog. The format and the location of the ID in the commit message are configured with SVN properties.

To make the integration possible Syncro SVN Client needs some data about the bug tracking tool used in the project. You can configure this using the following *SVN properties* which must be set on the folder containing resources associated with the bug tracking system. Usually they are set recursively on the root folder of the working copy.

- **bugtraq:message** - A string property. If it is set *the Commit dialog* will display a text field for entering the bug ID. It must contain the string *%BUGID%*, which is replaced with the bug number on commit.
- **bugtraq:label** - A string property that sets the label for the text field configured with the **bugtraq:message** property.
- **bugtraq:url** - A string property that is the URL pointing to the bug tracking tool. The URL string should contain the substring *%BUGID%* which Syncro SVN Client replaces with the issue number. That way the resulting URL will point directly to the correct issue.
- **bugtraq:warnifnoissue** - A boolean property with the values *true/yes* or *false/no*. If set to *true*, the Syncro SVN Client will warn you if the bug ID text field is left empty. The warning will not block the commit, only give you a chance to enter an issue number.
- **bugtraq:number** - A boolean property with the value *true* or *false*. If this property is set to *false*, then any character can be entered in the bug ID text field. If the property is set to *true* or is missing then only numbers are allowed as the bug ID.
- **bugtraq:append** - A boolean property. If set to *false*, then the bug ID is inserted at the beginning of the commit message. If *yes* or not set, then it's appended to the commit message.
- **bugtraq:logregex** - This property contains one or two regular expressions, separated by a newline. If only one expression is set, then the bug ID's must be matched in the groups of the regular expression string, for example `[Ii]ssue #?(\d+)` If two expressions are set, then the first expression is used to find a string which relates to a bug ID but may contain more than just the bug ID (e.g. `Issue #123` or `resolves issue 123`). The second expression is then used to extract the bug ID from the string extracted with the first expression. An example: if you want to catch every pattern `issue #XXX` and `issue #890, #789` inside a log message you could use the following strings:
  - `[Ii]ssue #?(\d+)(,? ?#?(\d+))+`
  - `(\d+)`

The data configured with these SVN properties is stored on the repository when a revision is committed. A bug tracking system or a statistics tools can retrieve from the SVN server the revisions that affected a bug and present the commits related to that bug to the user of the bug tracking system.

If the **bugtraq:url** property was filled in with the URL of the bug tracking system and this URL includes the *%BUGID%* substring as specified above in the description of the **bugtraq:url** property then *the History view* presents the bug ID as a hyperlink in the commit message. A click on such a hyperlink in the commit message of a revision opens a Web browser at the page corresponding to the bug affected by that commit.

Obtain Information for a Resource

This section explains how to obtain information for a SVN resource:

Request Status Information for a Resource

While you are working you often need to know which files you have changed, added, removed or renamed, or even which files got changed and committed by others. That's where the **Synchronize** action from *Working Copy view* comes in handy. The **Working Copy** view will show you every file that has changed in any way in your working copy, as well as any unversioned files you may have.

If you want more detailed information about a given resource you can use the **Information** action from the **Working Copy** view *contextual menu*. A dialog called **SVN Information** will pop up showing remote and local information regarding the resource, such as:

- local path and repository location
- revision number
- last change author, revision and date
- commit comment
- information about locks
- local file status
- local properties status
- remote file status
- remote properties status
- file size, etc.

The value of a property of the resource displayed in the dialog can be copied by right clicking on the property and selecting the **Copy** action.

A less detailed list of information is also presented when you hover with the mouse pointer over a resource and the tooltip window is displayed.

Request History for a Resource

In Apache Subversion™, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision you can use the **History view** that can be accessed from *Repositories view*, *Working Copy view*, *Revision Graph*, or *Directory Change Set view*. From the **Working copy view** you can display the history of local versioned resources.

Management of SVN Properties

In the *Properties view* you can read and set the Apache Subversion™ properties of a file or folder. There is a set of predefined properties with special meaning to Subversion. For more information about properties in Subversion see the SVN Subversion specification. Subversion properties are revision dependent. After you change, add or delete a property for a resource, you have to commit your changes to the repository.

If you want to change the properties of a given resource you need to select that resource from the *Working Copy view* and run the **Show properties** action from the contextual menu. The **Properties** view will show the local properties for the resource in the working copy. Once the Properties view is visible, it will always present the properties of the currently selected resource. In the **Properties** view *toolbar* there are available actions which allow you to add, change and delete the properties.

If you choose the **Add a new property** action, a new dialog will pop-up containing:

- **Name** - Combo box which allows you to enter the name of the property. The drop down list of the combo box presents the predefined Subversion properties such as **svn:ignore**, **svn:externals**, **svn:needs-lock**, etc.
- **Current value** - Text area which allows you to enter the value of the new property.

If the selected item is a directory, you can also set the property recursively on its children by checking the **Set property recursively** checkbox.

If you want to change the value for a previously set property you can use the **Edit property** action which will display a dialog where you can set:

- **Name** - Property name (cannot be changed).
- **Current value** - Presents the current value and allows you to change it.

• **Base value** - The value of the property, if any, from the resource in the pristine copy. It cannot be modified.

If you want to completely remove a property previously set you can choose the **Remove property** action. It will display a confirmation dialog in which you can choose also if the property will be removed recursively.

In the *Properties view* there is a **Refresh** action which can be used when the properties have been changed from outside the view. This can happen, for example, when the view was already presenting the properties of a resource and they have been changed after an **Update** operation.

Branches and Tags

One of the fundamental features of version control systems is the ability to create a new line of development from the main one. This new line of development will always share a common history with the main line if you look far enough back in time. This line is known as a *branch*. Branches are mostly used to try out features or fixes. When the feature or fix is finished, the branch can be merged back into the main branch (*trunk*).

Another feature of version control systems is the ability to take a snapshot of a particular revision, so you can at any time recreate a certain build or environment. This is known as *tagging*. Tagging is especially useful when making release versions.

In Apache Subversion™, there is no difference between a *tag* and a *branch*. On the repository, both are ordinary directories that are created by copying. The trick is that they are cheap copies instead of physical copies. Cheap copies are similar to hard links in Unix, which means that they merely link to a specific tree and revision without making a physical copy. As a result, branches and tags occupy little space on the repository and are created very quickly.

As long as nobody ever commits to the directory in question, it remains a tag. If people start committing to it, it becomes a branch.

Create a Branch / Tag

In the *Working Copy view* or in the *Repositories view*, select the resource which you want to copy to a branch or tag, then select the command *Branch / Tag...* from the **Tools** menu.
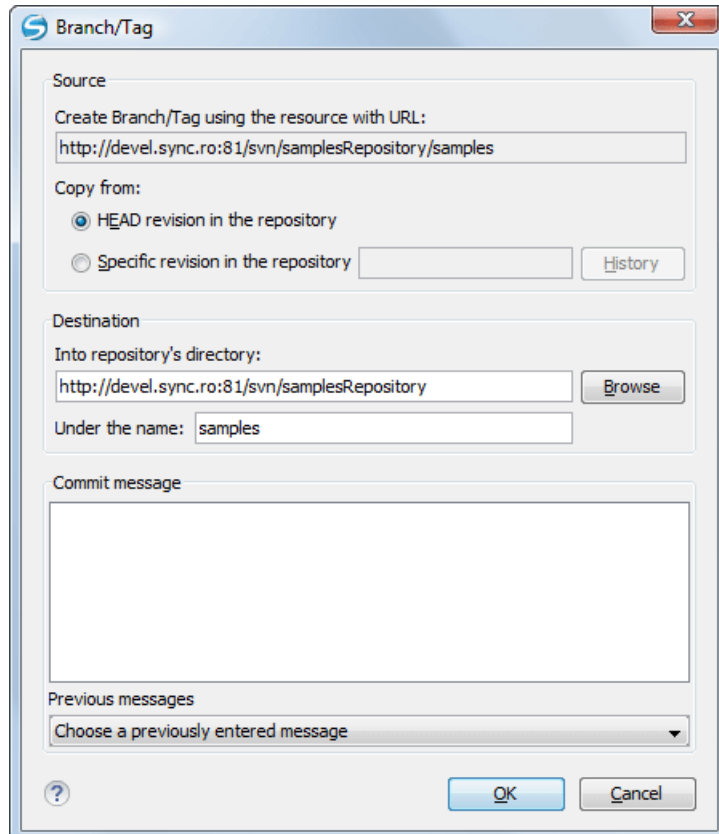
Figure 220: The Branch / Tag dialog

The default target URL for the new branch / tag will be the repository URL of the selected resource from your working copy, divided in two: the URL of the parent and the selected resource's name. You may specify other resource name if you want to make a branch / tag using a different name than the one of the selected resource, by modifying the field labeled **Under the name:**. The new branch / tag will be created as child of the specified repository directory URL and having the new provided name. To change the parent directory URL to the new path for your branch / tag, click on the **Browse** button and choose a repository target directory for your resource.

You can also specify the source of the copy. There are three options:

- **HEAD revision in the repository** - The new branch / tag will be copied in the repository from the HEAD revision. The branch will be created very quickly as the repository will make a cheap copy.
- **Specific revision in the repository** - The new branch will be copied in the repository but you can specify exactly the desired revision. This is useful for example if you forgot to make a branch / tag when you released your application. If you click on the **History** button on the right you can select the revision number from *the History dialog*. This type of branch will also be created very quickly.
- **Working copy** - The new branch will be a copy of your local working copy. If you have updated some files to an older revision in your working copy, or if you have made local changes, that is exactly what goes into the copy. This involves transferring some data from your working copy back to the repository, more exactly the locally modified files.

When you are ready to create the new branch / tag, write a commit comment in the corresponding field and press the **OK** button.

Merging

At some stage during the development process, you will want to merge the changes made on a branch back into the trunk, or vice-versa. The merge is accomplished by comparing two points (branches or revisions) in the repository and applying the obtained differences to your working copy. This process is closely related to the *diff* concept.

ⓡ **Remember:** It is a good idea to perform a merge into an unmodified working copy. If you have made changes to your working copy, commit them first. If the merge does not go as you expect, you may want to revert the changes and revert cannot recover your uncommitted modifications.

📝 **Note:** The above recommendation becomes mandatory when *reintegrating a branch*.

The **Merge** action can be found in the **Tools** menu of Syncro SVN Client. The item selected when you issued the command will be the one receiving the generated changes. If there is no item selected, the merge operation will be performed on the entire working copy.

There are four common types of merging, which are handled in different ways:

- **Merge revisions** - port changes from one branch to another. Note that the *trunk* can also be considered a branch, in this context;
- **Synchronize a branch** - fetch all the changes made on a parent branch (or the *trunk*) to a child branch;
- **Reintegrate a branch** - merge back a branch to its parent branch (which can also be the *trunk*);
- **Merge two different trees** - integrate the changes done on a branch to a different branch.

ⓡ **Remember:** A *branch* is a line of development that exists independently of another line, yet still shares a common history if you look far enough back in time. A *branch* always begins life as *a copy of something* (i.e. trunk, another branch, or tag), and moves on from there, generating its own history.

Merge Revisions

This is the case when you have made one or more changes to a branch and you want to port them across a different branch. An example of such operation can be the following: we know that a problem has been fixed by committing revisions 17, 20 and 25 on branch B1. These changes are also needed in branch B2, so, in order to merge them, we need a working copy of the B2 branch.

1. Go to menu **Tools** > **Merge**
   The **Merge** wizard is opened.
2. Select the **Merge revisions** option.

3. Press the **Next** button.

The second page of the **Merge** wizard is displayed.

4. In the **Merge from (URL)** field enter the URL of the branch or tag containing the changes that you want to port into your working copy. In our example, it is the URL of the B1 branch.

You may also click the **Browse** button to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

> **Note:** If the URL belongs to a different repository than the working copy, **Ignore ancestry / Disable merge tracking** option will be selected automatically and you cannot change this, because *Subversion cannot track changes between different repositories*.

> **Note:** You can also specify a *peg revision* at the end of the URL (like `URL@revNo`), where `revNo` is the number of this revision. The peg revision does not affect the merge range you select. By default, the `HEAD` revision is assumed.

5. Select the revisions to merge. Choose between **all revisions** and **specific revision(s)** options.

If the **Ignore ancestry / Disable merge tracking** option, found on the wizard's *Options page*, is not selected, **all revisions** means *all eligible revisions* that were not yet merged.

If you choose **specific revision(s)**, you can specify one or multiple single revisions and/or revision ranges. Also, you can mix *forward* (ascending) and *backward* (descending) revision ranges and subtract specific revisions from a range.

For example, the construction `1-5, -3, 7, 10-HEAD, 20-15`, includes changes done in the following revisions:

- 1 to 5, without 3;
- 7;
- 10 to 14;
- 21 to the latest revision (`HEAD`).

> **Note:** If using the Subversion command-line client, a revision range of the form `1-5` means all changes starting from revision 2 up to revision 5 (the changes necessary to reach revision 5, committed after revision 1). Unlike the Subversion command-line client, in Syncro SVN Client the revision ranges are inclusive, meaning that it will process all revisions, starting with revision 1, up to revision 5. This behaviour is dictated by the application's interface, given that it is easier to choose explicitly the changes to be merged. The **History** button opens *the History dialog* which allows selecting the revisions to be merged (one or more).

> **Attention:** The `HEAD` revision is the only non-numerical revision allowed, and it can be used only when specifying revision ranges, as one of the ends of the range (the other end must to be an explicit revision number). But be careful when using it, as it might not refer to the revision you think it does, if someone else made a commit in the meantime.

> **Tip:** If you want to roll-back out of your working copy changes that have already been committed to the repository, use the *negative revisions* notation (like `-3`) and/or *backward revision ranges* (like `20-10`) to perform a *reverse merge*.

6. Press the **Next** button.

The *Options page* of the wizard is displayed, where you can select the options you need for a fine-grained control of the operation.

7. Press the **Merge** button.

The merge operation is performed.

8. Optionally, resolve the conflicts that were created by the merge operation.

After the merge operation is finished, it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are

such conflicts, the following dialog will appear, presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.
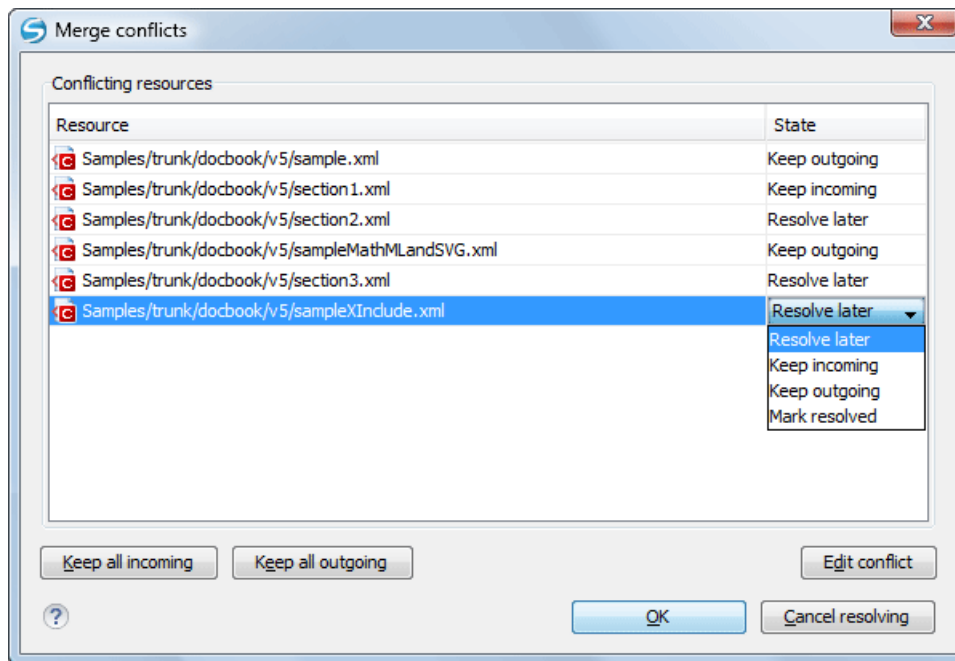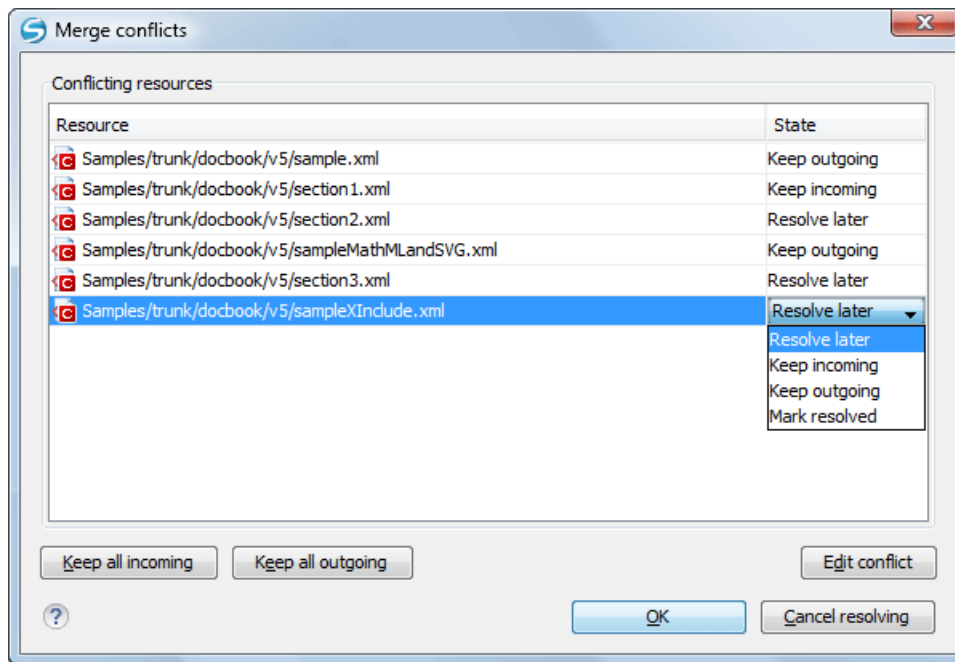

Figure 221: Merge Conflicts Dialog

The options to resolve a conflict are:

- **Resolve later** - Used for leaving the conflict as it is, to manually resolve it later.
- **Keep incoming** - This option keeps all the incoming modifications, discarding all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy, discarding all incoming ones.
- **Mark resolved** - You should chose this option after you have manually edited the conflict. To do this, use the **Edit conflict** button, that will display a dialog presenting the conflicting resource's contents (for the working copy version versus the incoming version). After manually resolving the conflict, the resource will be marked as resolved.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It's a good idea to look at the result of the merge, in the working copy, to review the changes and see if it meets your expectations. Because merging is sometimes complicated, *conflicts may appear* when there are major changes.

📝 **Note:** The merge result is only in your local working copy and needs to be committed to the repository, so that it can be seen by others.

Synchronize a Branch

While working on your own branch, other people on your team continue to make important changes in it's parent branch (which can be the *trunk* itself or any other branch). It's in your best interest to replicate those changes to your branch, to make sure they mesh well with your changes. This is done by performing a *synchronize merge*, that will bring your branch up-to-date with any changes made to its ancestral parent branch, since your branch was created. Subversion is aware of the history of your branch and knows when it split away from the parent branch.

Frequently keeping your branch in sync with the parent branch helps you prevent "surprise" conflicts when the time comes for you to hold your changes back into the parent. The synchronization uses merge tracking to skip all those

revisions that have already been merged, so a sync merge can be repeated periodically to fetch all the parent's latest changes, in order to stay up-to-date with it.

> ⚠ **Important:** It is recommended to synchronize the whole working copy of the child branch (the root of the working copy), not just a part of it.

After running the prior example, your branch working copy now contains new local modifications, and these edits are duplications of all of the changes that have happened on the trunk since you first created your branch:

At this point, your private branch is now "in sync" with the trunk, so you can rest easier knowing that as you continue to work in isolation, you're not drifting too far away from what everyone else is doing.

1. Go to **Tools** > **Merge**
   The **Merge** wizard is opened.

2. Select the **Synchronize branch** option.

3. Press the **Next** button.
   The second page of the **Merge** wizard is displayed.

4. In the **Parent branch (URL)** field enter the URL of the branch from which you created your branch. This means that the URL must belong to the same repository as your working copy from the child branch.

   You may also click the **Browse** button to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

   > 📄 **Note:** You can also specify a *peg revision* at the end of the URL (like `URL@revNo`), where `revNo` is the number of this revision. The peg revision specifies both the peg revision of the URL and the latest revision that will be considered for merging. By default, the `HEAD` revision is assumed.

5. Press the **Next** button.
   The *Options page* of the wizard is displayed, where you can select the options you need for a fine-grained control of the operation.

   > 📄 **Note: Ignore ancestry / Disable merge tracking** is not available for this merge type, since a synchronization merge should always be recorded in the destination branch.

6. Press the **Merge** button.
   The merge operation is performed.

7. Optionally, resolve the conflicts that were created by the merge operation.

   After the merge operation is finished, it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are such conflicts, the following dialog will appear, presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.
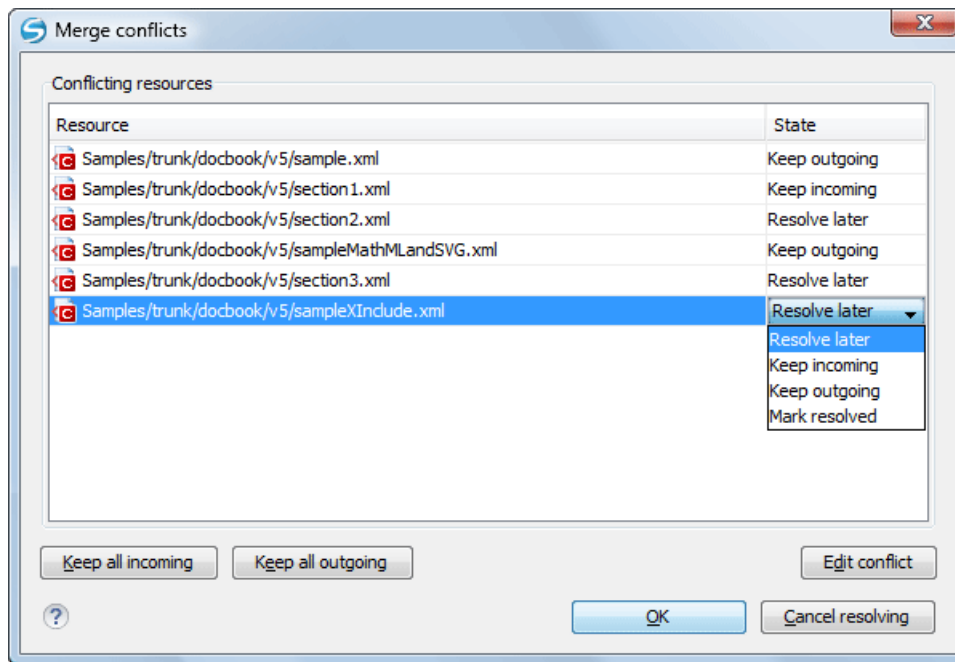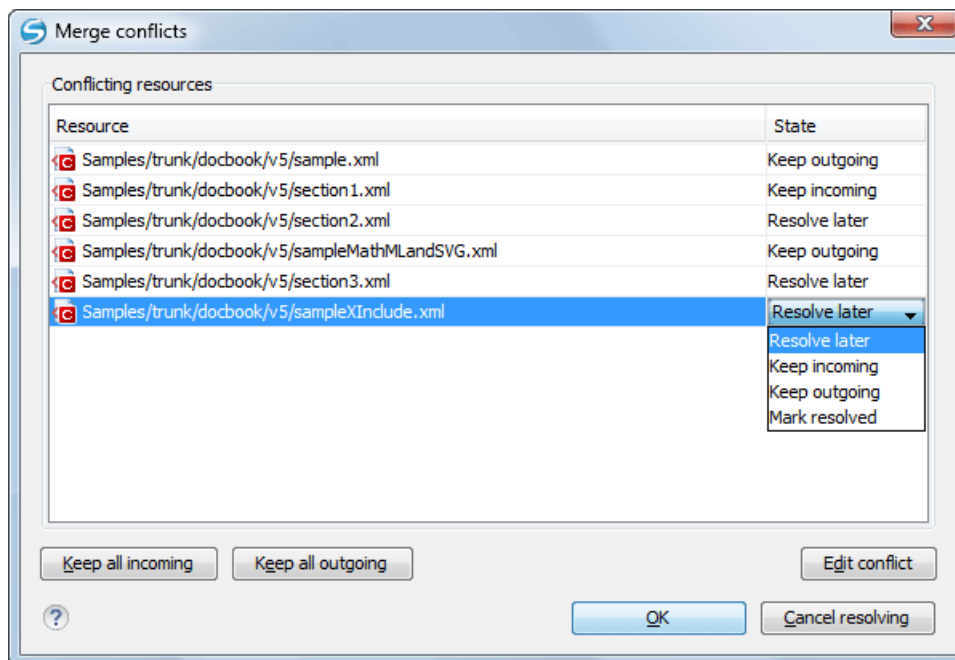
Figure 222: Merge Conflicts Dialog

The options to resolve a conflict are:

- **Resolve later** - Used for leaving the conflict as it is, to manually resolve it later.
- **Keep incoming** - This option keeps all the incoming modifications, discarding all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy, discarding all incoming ones.
- **Mark resolved** - You should chose this option after you have manually edited the conflict. To do this, use the **Edit conflict** button, that will display a dialog presenting the conflicting resource's contents (for the working copy version versus the incoming version). After manually resolving the conflict, the resource will be marked as resolved.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It's a good idea to look at the result of the merge, in the working copy, to review the changes and see if it meets your expectations. Because merging is sometimes complicated, *conflicts may appear* when there are major changes.

> **Note:** The merge result is only in your local working copy and needs to be committed to the repository, so that it can be seen by others.

Reintegrate a Branch

There are some conditions that apply to a reintegrate merge:

- the server must support merge tracking;
- the source branch (to be reintegrated) must be coherently synchronized with its parent branch;

  This means that all revisions between the branching point and the last revision merged from the parent branch to the child branch must be merged to the latter one (there must be no missing revisions in-between).

- the working copy **must not**:

  - have any local modifications;
  - contain a mixture of revisions (all items must point to the same revision);
  - have any sparse directories (all directories must be of depth *infinity*);

- contain any switched items;

This method covers the case when you have a feature branch on which the development has concluded and it should be merged back into its parent branch. Since you have kept the feature branch synchronized with its parent, the latest versions of them will be absolutely identical except for your feature branch changes. These changes can be reintegrated into the parent branch by using a working copy of it and the **Reintegrate a branch** option.

This method uses the merge-tracking features of Apache Subversion™ to automatically calculate the correct revision ranges and to perform additional checks which ensure that the branch to be reintegrated has been fully updated with its parent changes. This ensures that you don't accidentally undo work that others have committed to the parent branch, since you last synchronized the child branch with it. After the merge ends, all branch development will be completely merged back into the parent branch, so the child branch will be redundant and can be deleted from the repository.

> **Tip:** It is recommended to do a last synchronization of the child branch with its parent branch, before reintegrating it, to be sure you avoid any possible conflicts.

1. Go to menu **Tools** > **Merge**
   The **Merge** wizard is opened.
2. Select the **Reintegrate a branch** option.

   This option is disabled if the selected working copy item, or any of the items inside it (in case of directories), has any type of modification, because it is mandatory for the target item to have no modifications.

3. Press the **Next** button.
   The second page of the **Merge** wizard is displayed.
4. In the **Child branch (URL)** field enter the URL of the child branch to be reintegrated. This means that the URL must belong to the same repository as your working copy from the parent branch.

   You may also click the **Browse** button to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

   > **Note:** You can also specify a *peg revision* at the end of the URL (like URL@revNo), where revNo is the number of this revision. By default, the HEAD revision is assumed.

5. Press the **Next** button.
   The *Options page* of the wizard is displayed, where you can select the options you need for a fine-grained control of the operation.

   > **Note:** Because a *reintegrate merge* is quite specialized for the use case described above, most of the merge options are not applicable, except those in the **File Comparison** category.

6. Press the **Merge** button.
   The merge operation is performed.
7. Optionally, resolve the conflicts that were created by the merge operation.

   After the merge operation is finished, it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are such conflicts, the following dialog will appear, presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.
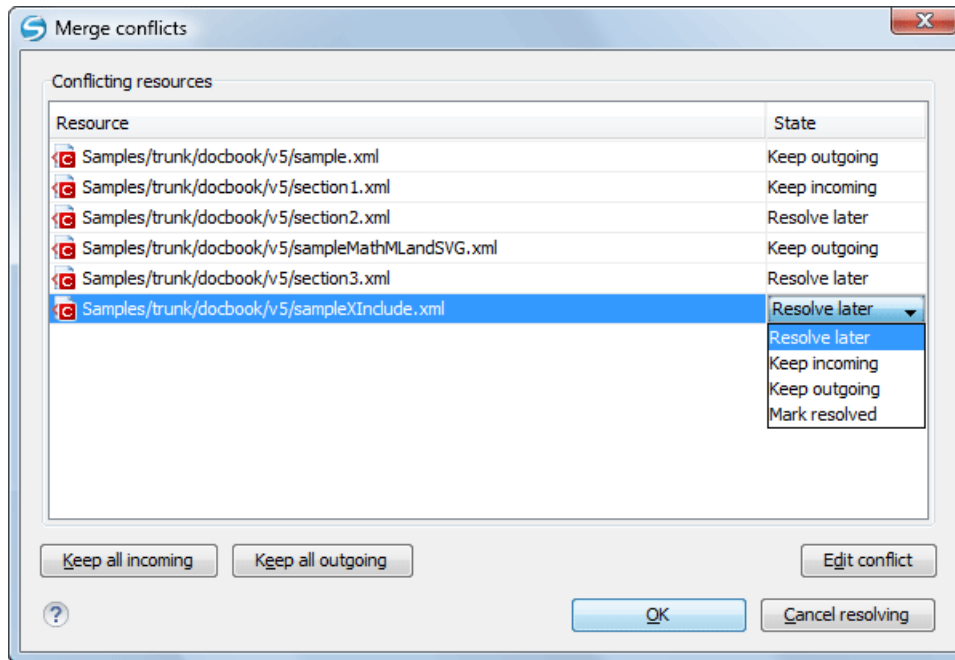
Figure 223: Merge Conflicts Dialog

The options to resolve a conflict are:

- **Resolve later** - Used for leaving the conflict as it is, to manually resolve it later.
- **Keep incoming** - This option keeps all the incoming modifications, discarding all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy, discarding all incoming ones.
- **Mark resolved** - You should chose this option after you have manually edited the conflict. To do this, use the **Edit conflict** button, that will display a dialog presenting the conflicting resource's contents (for the working copy version versus the incoming version). After manually resolving the conflict, the resource will be marked as resolved.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It's a good idea to look at the result of the merge, in the working copy, to review the changes and see if it meets your expectations. Because merging is sometimes complicated, *conflicts may appear* when there are major changes.

> **Note:** The merge result is only in your local working copy and needs to be committed to the repository, so that it can be seen by others.

Merge Two Different Trees

This is a general case of the reintegrate method. You can consider the following example: calculate the changes necessary to get from the HEAD revision of a parent branch (or the *trunk*) to the HEAD revision of one of its child branches, and apply those changes to my working copy of the parent branch. The result is that the parent branch will be identical with the child branch.

> **Note:** If the server does not support merge-tracking, then this is the only way to merge a branch back to its parent.

1. Go to menu **Tools** > **Merge**
   The **Merge** wizard is opened.
2. Select the option **Merge two different trees**.

3. Press the **Next** button.
The second page of the **Merge** wizard is displayed.

4. In the **From (starting URL and revision)** field enter the URL of the first tree.

You may also click the **Browse** button to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

> **Tip:** If you are using this method to merge a feature branch back to its parent branch, you need to start the merge wizard from within a working copy of the parent. In the field enter the full URL of the parent branch. This may sound wrong, but remember that the parent is the start point to which you want to add the branch changes.

> **Note:** If the URL belongs to a different repository than the working copy, **Ignore ancestry / Disable merge tracking** option will be selected automatically and you cannot change this, because *Subversion cannot track changes between different repositories*.

> **Note:** You can also specify a *peg revision* at the end of the URL (like `URL@revNo`), where `revNo` is the number of this revision. By default, the `HEAD` revision is assumed.

5. Select a revision to compute changes starting from it.

Enter the last revision number at which the two trees were synchronized. If you are sure no one else commits, you can use the **HEAD revision** (selected by default). If there is a chance that someone else might have made a commit since that synchronization, use the **other revision** field to input a specific revision number, to avoid losing more recent commits. You can use the **History** button to see a list of all revisions.

6. In the **To (ending URL and revision)** field enter the URL of the second tree.

You may also click the **Browse** button to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

> **Tip:** If you are using this method to merge a feature branch back to its parent branch, enter the URL of the feature branch. This way, only the changes unique to this branch will be merged, since the branch should have been periodically synchronized with its parent.

> **Note:** The URL must point to the same repository as the one in the **From (starting URL and revision)** field, else the operation will not be allowed, because Subversion cannot compute changes between items from different repositories.

> **Note:** You can also specify a *peg revision* at the end of the URL (like `URL@revNo`), where `revNo` is the number of this revision. By default, the `HEAD` revision is assumed.

7. Select a revision to compute all changes committed up to that point inclusively.

By default, the **HEAD revision** is selected. If you want a previous revision, select the **other revision** option and press *the History button* to see a list of all revisions.

8. Press the **Next** button.
The *Options page* of the wizard is displayed, where you can select the options you need for a fine-grained control of the operation.

9. Press the **Merge** button.
The merge operation is performed.

10. Optionally, resolve the conflicts that were created by the merge operation.

After the merge operation is finished, it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are such conflicts, the following dialog will appear, presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.

Figure 224: Merge Conflicts Dialog

The options to resolve a conflict are:

- **Resolve later** - Used for leaving the conflict as it is, to manually resolve it later.
- **Keep incoming** - This option keeps all the incoming modifications, discarding all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy, discarding all incoming ones.
- **Mark resolved** - You should chose this option after you have manually edited the conflict. To do this, use the **Edit conflict** button, that will display a dialog presenting the conflicting resource's contents (for the working copy version versus the incoming version). After manually resolving the conflict, the resource will be marked as resolved.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It's a good idea to look at the result of the merge, in the working copy, to review the changes and see if it meets your expectations. Because merging is sometimes complicated, *conflicts may appear* when there are major changes.

> **Note:** The merge result is only in your local working copy and needs to be committed to the repository, so that it can be seen by others.

Merge Options

Here is the list of options that can be used when merging:
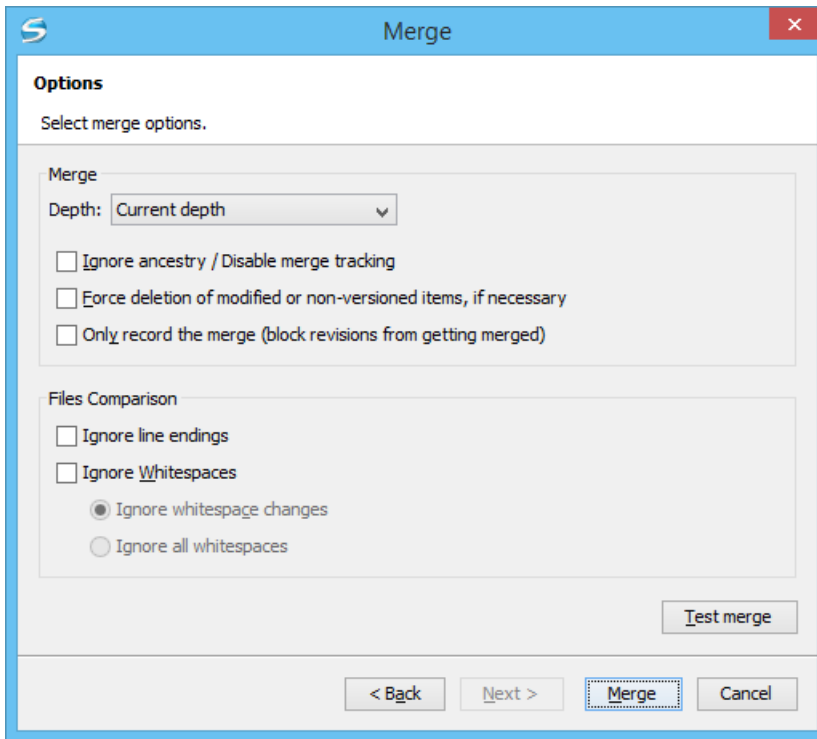
Figure 225: The Merge Wizard - Advanced Options

- **Depth** - sets the depth of the merge operation. You can specify how far down into your working copy the merge should go by selecting one of the following values:

  - Current depth
  - Recursive (*infinity*)
  - Immediate children (*immediates*)
  - File children only (*files*)
  - This folder only (*empty*)

  The *depth* term is described in the *Sparse checkouts* section. The default depth is the current depth of the working copy item receiving the merge.

- **Ignore ancestry / Disable merge tracking** - changes the way two items are merged if they do not share a common ancestry. Most merges involve comparing items that are ancestrally related to one another. Occasionally, however, you may want to merge unrelated items, so, the first item will be replaced with the second item, if this option is not enabled. In such situations, you would want the merge to do a path-based comparison only, ignoring any relations between the items. For example, in case of two different files, having the same name and being in the same relative location, this makes the difference between replacing one of the files with the other one and merging their contents.

  > **Note:** This option is selected automatically (and you cannot change this), if the URL of the merge source belongs to a different repository than the URL of the target working copy item (the one receiving the changes), because *Subversion cannot track changes between different repositories*.

- **Force deletion of modified or non-versioned items, if necessary** - When the merge operation involves deleting locally modified or non-versioned items, it will fail if this option is not enabled. This is done in order to prevent data loss.

- **Only record the merge (block revisions from getting merged)** - Enables a special mode of the merge operation, that just records it in the local merge tracking information, without actually performing it (does not modify any file contents or the structure of your working copy), when the merge tracking support is enabled. You might want to do so for two possible reasons:

- you made (or you will make) the merge manually, so you need to mark the revisions as having been merged to make the merge tracking system aware of them. This way, they will be excluded from future merges;
- you want to prevent one or more particular changes from being fetched in subsequent merges.

- **Ignore line endings**, **Ignore whitespaces** - allow you to specify how the line ending and whitespace changes should be handled. By default, all such changes are treated as real content changes, but you can ignore them if you select these options. **Ignore whitespace changes** excludes changes in the amount of whitespaces or to their type (for example, when changing the indentation or changing tabs to spaces). Adding whitespaces where there were none before, or removing a whitespace, is still considered to be a change. **Ignore all whitespaces** ignores all types of whitespace changes.
- **Test merge** - performs a dry run of the merge operation, allowing to "preview" it without actually performing the merge. In the **Console** view you will see a list of the working copy items that will be affected and how they will be affected. This is helpful in detecting if a merge will be successful or not, and where conflicts may occur.

Resolve Merge Conflicts

Optionally, resolve the conflicts that were created by the merge operation.

After the merge operation is finished, it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are such conflicts, the following dialog will appear, presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.

Figure 226: Merge Conflicts Dialog

The options to resolve a conflict are:

- **Resolve later** - Used for leaving the conflict as it is, to manually resolve it later.
- **Keep incoming** - This option keeps all the incoming modifications, discarding all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy, discarding all incoming ones.
- **Mark resolved** - You should chose this option after you have manually edited the conflict. To do this, use the **Edit conflict** button, that will display a dialog presenting the conflicting resource's contents (for the working copy version versus the incoming version). After manually resolving the conflict, the resource will be marked as resolved.

Final Words on Merge

Subtree Merges

It is recommended to perform a merge on the whole working copy (select its root directory when triggering the operation) to avoid *subtree mergeinfo*. *Subtree mergeinfo* is the mergeinfo recorded to describe a *subtree merge* - that is, a merge done directly to some child of a branch root, which might be needed in some situations. There is nothing special about *subtree merges* or *subtree mergeinfo* except only one important point to keep in mind: the complete record of merges to a branch may not be contained solely in the mergeinfo on the branch root and you may have to look to any subtree mergeinfo to get a full accounting. Fortunately Subversion does this for you and rarely will you need to concern yourself with it.

Merging from Foreign Repositories

Subversion does support merging from foreign repositories. While all merge source URLs must point to the same repository, the merge target working copy may come from a different repository than the source. However, there are some caveats. Most notably, copies made in the merge source will be transformed into plain additions in the merge target. Also, merge-tracking is not supported for merges from foreign repositories.

> **Note:** When performing merges from other repositories than the one of the destination working copy item, the **Ignore ancestry / Disable merge tracking** option will be selected automatically.

General Merge Recommendations

As a recommendation, merge only into clean working copies, that **do not**:

- contain modifications;
- have any sparse directories (all directories must be of depth *infinity*);
- contain any switched items;

> **Important:** While the above is only a recommendation, it becomes mandatory when performing a *reintegrate merge*. Also, trying to merge to mixed-revision working copies will fail.

> **Remember:** The merge result is only in your local working copy and needs to be committed to the repository, so that it can be seen by others.

Switch the Repository Location

The **Switch** action is useful when the repository location of a working copy or only of a versioned item of the working copy must be changed within the same repository. The action is available on the **Tools** menu when a versioned resource is selected in the current working copy that is displayed in *the Working Copy view*.

Relocate a Working Copy

Sometimes the base URL of the repository is changed after a working copy is checked out from that URL, for example the repository itself is relocated to a different server. In such cases you do not have to check out again a working copy from the new repository location. It is easier to change the base URL of the root folder of the working copy to the new URL of the repository. This action is called **Relocate** and is available on the **Tools** menu when the selected item in *the Working Copy view* is a versioned folder.

If the selected item is not the root folder of the working copy then the effect is the same as for *the Switch action* applied on the same selected item.

Patches

This section explains how to work with patches in Syncro SVN Client.

What Is a Patch

Let's suppose you are working to a set of XML files, that you distribute to other people. From time to time you are tagging the project and distribute the releases. If you continue working for a period correcting problems, you may find yourself in the situations to notify your users that you have corrected a problem. In this case you may prefer to distribute

them a patch, a collection of differences that applied over the last distribution would correct the problem. The SVN client creates the patch in *the Unified Diff format*.

Creating a patch in Apache Subversion™ implies the access to two states (revisions) of a project:

- the current working copy and a revision from the repository - if you have not committed yet your current working copy and prefer not to do it, you create a patch between the current working copy and a revision from the repository
- two repository revisions - if both states are two revisions already committed to the repository

Create a Patch Between Working Copy and Repository Revision

When the changes that must be included in a patch were not committed to the repository the patch should be created with the differences between the selected item(s) of the working copy and a repository revision. The steps are the following:

1. Go to menu **Tools** > **Create patch**.
   This opens the **Create patch** wizard:
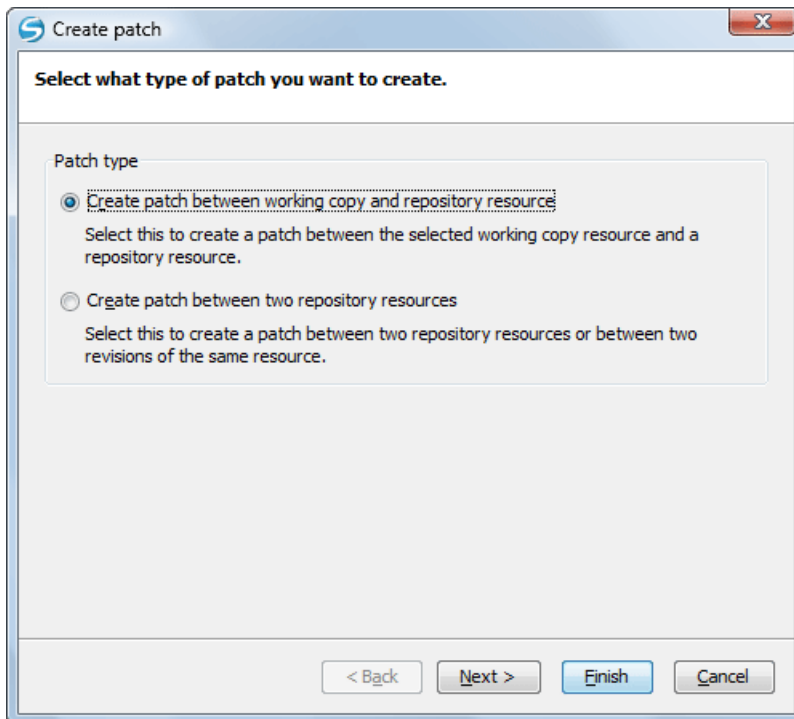
Figure 227: The Create Patch Wizard - Patch Type

2. Select the first option in the dialog.
3. Press the **Next** button.
   The second step of the wizard is opened:
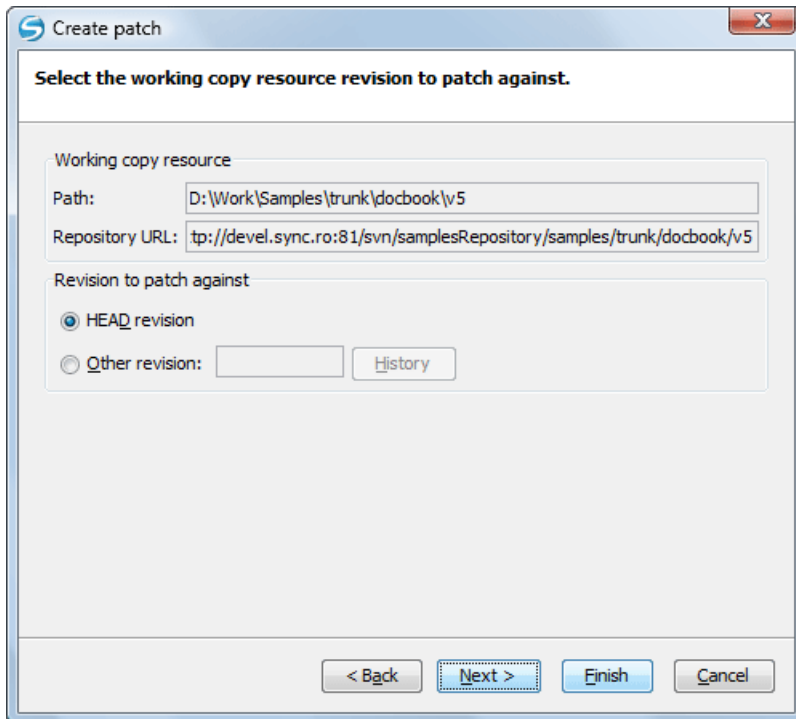
Figure 228: The Create Patch Wizard - Step 2

4. Specify the working copy resource.

    a) Specify the local file path of the working copy resource.

    b) Specify the repository URL corresponding to the working copy resource.

5. Select the repository revision.

You have the option of choosing between the HEAD revision and a specific revision number. For the second option you should press *the History button* to display a list of the repository revisions.

6. Press the **Next** button.
The next step of the wizard is displayed:
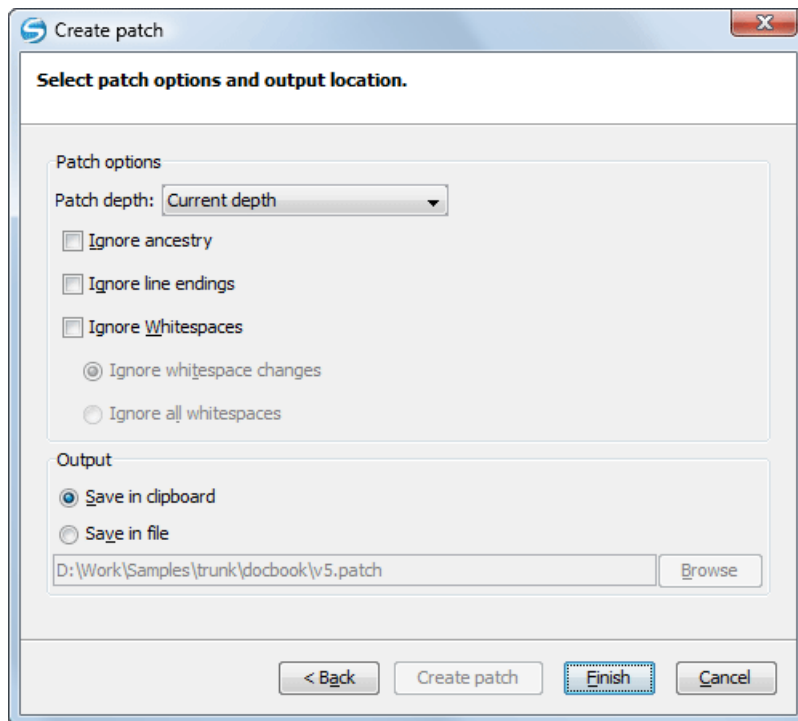
Figure 229: The Create Patch Wizard - Options

7. Select the depth of the patch operation.

In case of a file resource the depth is always zero. In case of a folder resource the depth has one of the following values:

- **Current depth** - The depth of going into the folder for creating the patch is the same as the depth of that folder in the working copy.
- **Recursive (infinity)** - The patch is created on all the files and folders contained in the selected folder.
- **Immediate children (immediates)** - The patch is created only on the child files and folders without going in subfolders.
- **File children only (files)** - The patch is created only on the child files.
- **This folder only (empty)** - The patch is created only on the selected folder (that is no child file or folder is included in the patch).

8. Select the **Ignore ancestry** checkbox (optional).

If checked, the SVN ancestry that exists when the two URLs have a common SVN history is ignored.

9. Select the **Ignore line endings** checkbox (optional).

If checked, the differences in line endings are ignored when the patch is created.

10. Select the **Ignore whitespaces** checkbox (optional).

If checked, the differences in white spaces are ignored when the patch is created.

11. Select the output location.

- **Save in clipboard** - The patch will be created and saved in clipboard. This is useful when you do not want to save the patch in a file on disk.
- **Save in file** - The patch will be created and saved in the specified file.

12. Press the **Next** button.
This will go to the final step of the wizard:

Figure 230: The Patch Wizard - Add Unversioned Resources

13. Select the unversioned files that will be included in the patch.

If the patch is applied on a folder of the working copy and that folder contains unversioned files this step of the wizard offers the option of selecting the ones that will be included in the patch.

14. Press the OK button.
The patch is created by applying all the specified options.

Create a Patch Between Two Repository Revisions

When a patch must include the differences between two repository revisions, in the same repository or in two different repositories, the steps for creating the patch are the following:

1. Go to menu **Tools** > **Create patch**.
This opens the **Create patch** wizard:
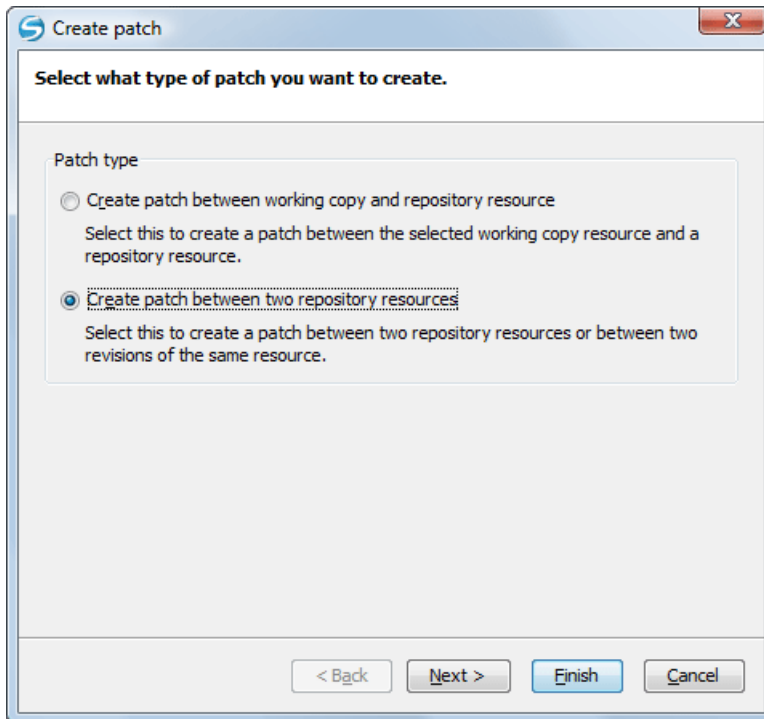
Figure 231: The Create Patch Wizard - Patch Type

2. Select the second option in the dialog.
3. Press the **Next** button.
   The second step of the wizard is opened:
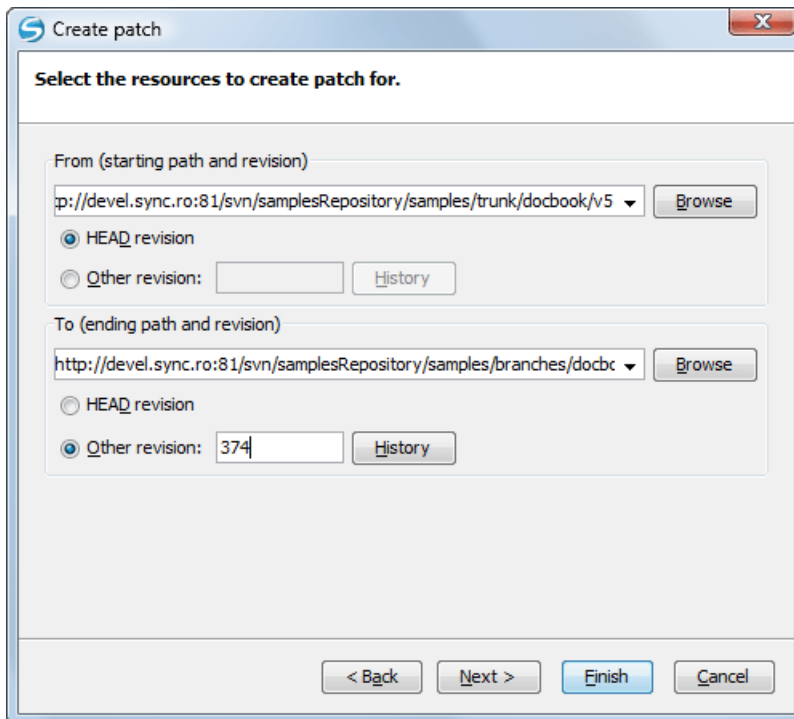
Figure 232: The Create Patch Wizard - Step 2

4. Select the first repository revision in the **From** panel.
5. Select the second repository revision in the **To** panel.

For both revisions you have the option of choosing between the HEAD revision and a specific revision number. For a specific number you should press *the History button* to display a list of the repository revisions.

6. Press the **Next** button.

The next step of the wizard is displayed:
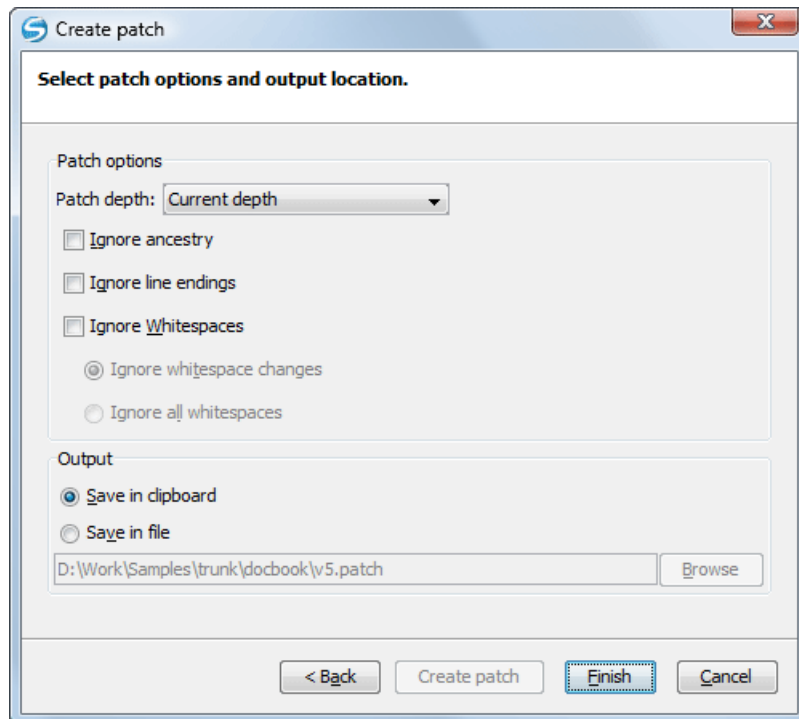
Figure 233: The Create Patch Wizard - Options

7. Select the depth of the patch operation.

In case of a file resource the depth is always zero. In case of a folder resource the depth has one of the following values:

- **Current depth** - The depth of going into the folder for creating the patch is the same as the depth of that folder in the working copy.
- **Recursive (infinity)** - The patch is created on all the files and folders contained in the selected folder.
- **Immediate children (immediates)** - The patch is created only on the child files and folders without going in subfolders.
- **File children only (files)** - The patch is created only on the child files.
- **This folder only (empty)** - The patch is created only on the selected folder (that is no child file or folder is included in the patch).

8. Select the **Ignore ancestry** checkbox (optional).

If checked, the SVN ancestry that exists when the two URLs have a common SVN history is ignored.

9. Select the **Ignore line endings** checkbox (optional).

If checked, the differences in line endings are ignored when the patch is created.

10. Select the **Ignore whitespaces** checkbox (optional).

If checked, the differences in white spaces are ignored when the patch is created.

11. Select the output location.

- **Save in clipboard** - The patch will be created and saved in clipboard. This is useful when you do not want to save the patch in a file on disk.
- **Save in file** - The patch will be created and saved in the specified file.

12. Press the **Next** button.

This will go to the final step of the wizard:

Figure 234: The Patch Wizard - Add Unversioned Resources

13. Select the unversioned files that will be included in the patch.

If the patch is applied on a folder of the working copy and that folder contains unversioned files this step of the wizard offers the option of selecting the ones that will be included in the patch.

14. Press the OK button.

The patch is created by applying all the specified options.

Working with Repositories

This section explains how to locate and browse SVN repositories in Syncro SVN Client.

Importing Resources Into a Repository

This is the process of taking a project and importing it into a repository so that it can be managed by an Apache Subversion™ server. If you have already been using Subversion and you have an existing working copy you want to use, then you will likely want to follow the procedure for *using an existing working copy*.

This process is started from menu **Repository** > **Import** > **Import Folder Content**. The same action is available in the **Repositories** view contextual menu. A dialog will ask you to select a directory that will be imported into the selected repository location. The complete directory tree will be imported into the repository including all files. The name of the imported folder will not appear in the repository, but only the contents of the folder will.

Exporting Resources From a Repository

This is the process of taking a resource from the repository and saving it locally in a clean form, with no version control information. This is very useful when you need a clean build for an installation kit.

The export dialog is very similar to the check out dialog:
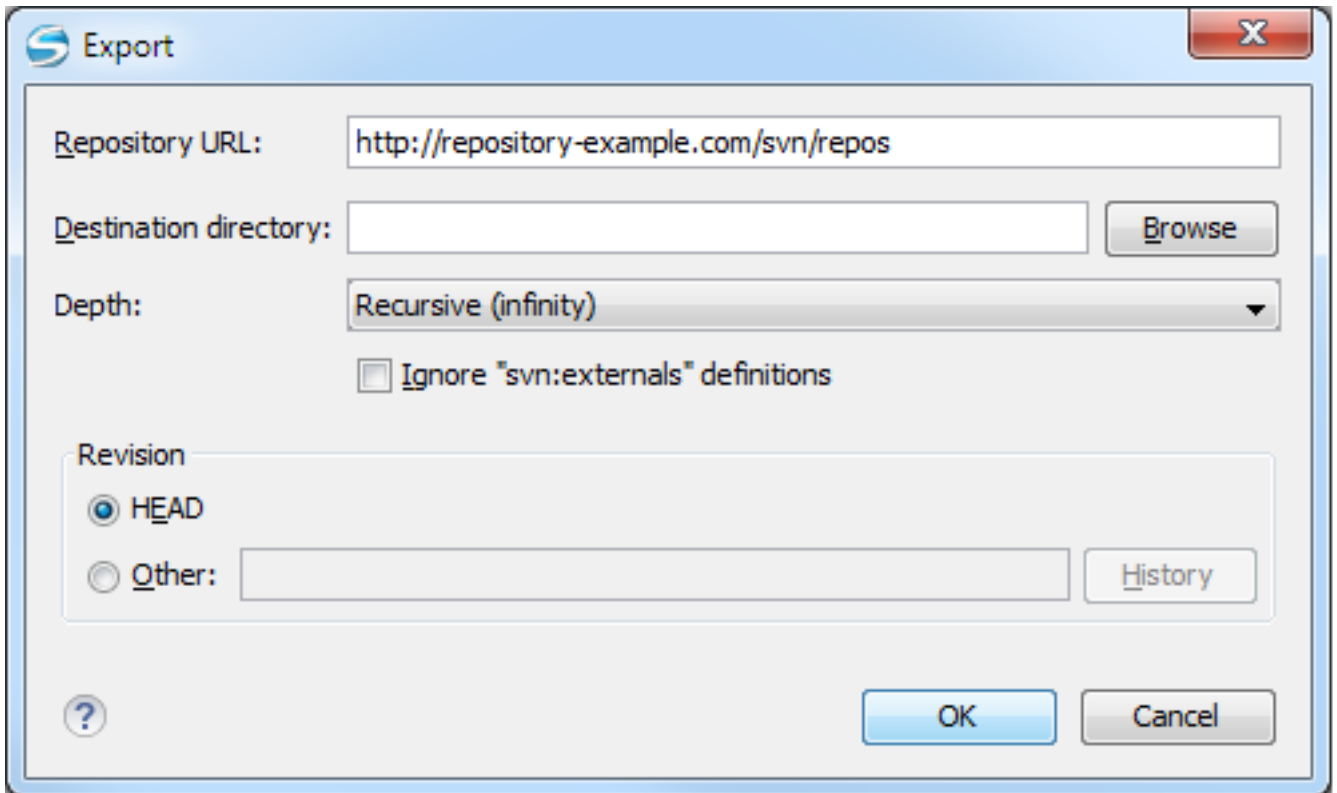
Figure 235: Export from Repository

You can choose the target directory from the file system by pressing the **Browse** button. If you need to export a specific revision, you can select the **Revision** radio button and then click on the **History** button and choose a revision from the new dialog. Or you could simply type the revision number in the corresponding text field.

Please note that the content of the selected directory from the repository and not the directory itself will be exported to the file system.

Copy / Move / Delete Resources From a Repository

Once you have a location defined in the *Repositories view*, you can execute commands like copy, move and delete directly on the repository. The commands correspond to the following actions in the contextual menu:

The **Copy to** and **Move to** action allows you to copy and move individual or multiple resources to a specific directory from the *HEAD* revision of the repository.
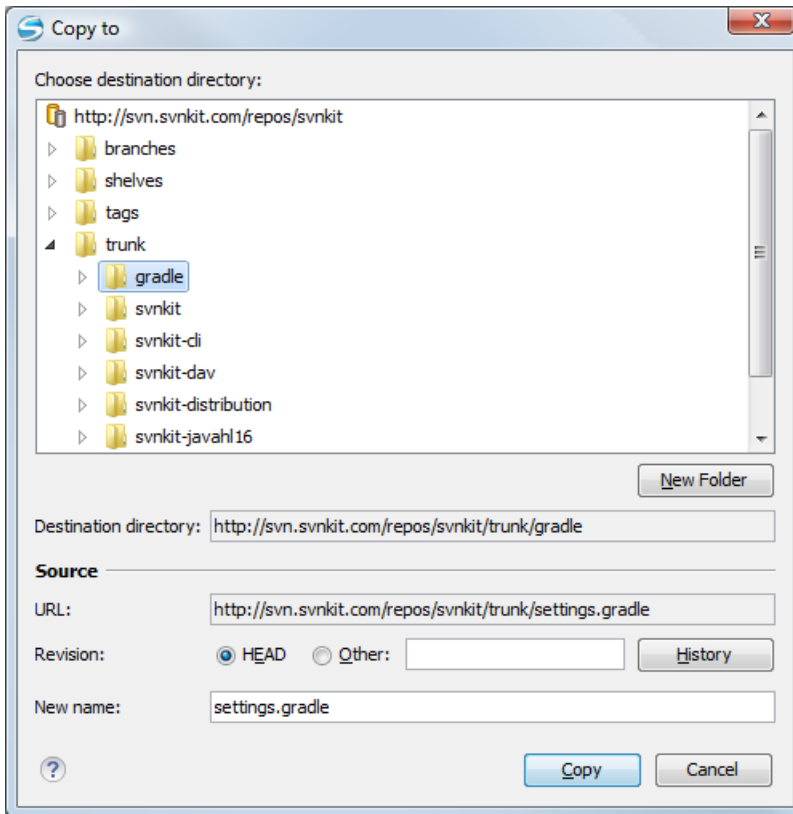
Figure 236: Copy/Move Items in Repository

The dialog box used to copy or move items allows you to browse the *HEAD* revision of the repository and select the destination of the items, presenting its repository URL below the tree view.

The **Source** section presents relevant options regarding the item(s) that you move or copy:

• **URL** - this field is displayed only if you copy/move a single item;
• **Revision** - presents the revision from which you copy one or more items, allowing you to also choose another revision;

> **Note:** Since only items from the HEAD revision can be moved, the **Revision** options are not presented for the **Move to** action.

> **Note:** When you copy a single item while browsing a revision other than *HEAD*, the **Revision** options present this revision but does not allow you to change it. The same applies if copying multiple items.

• **New name** - This option is presented when you copy or move a single item, allowing you to also rename it.

Another useful action is **Delete**, allowing you to erase resources directly from the repository.

All three actions are commit operations and you will be prompted with the **Commit message** dialog.

Sparse Checkout

Sometimes you need to check out only certain parts of a directory tree. For this you can check out the top directory (*the action Check out from the Repositories view*) and then update recursively only the needed directories (*the action Update from the Working Copy view*). Now, each directory has a depth set to it, which has four possible values:

• **Recursive (*infinity*)** - Updates all descendant directories and files recursively.
• **Immediate children (*immediates*)** - Updates the directory, including direct child directories and files, but does not populate the child directories.
• **File children only (*files*)** - Updates the directory, including only child files without the child directories.

- **This folder only (*empty*)** - Updates only the selected directory, without updating any children.

For some operations, you can use as depth the current depth registered on the directories from the working copy (the value **Current depth**). This is the depth value defined in a previous check out or update operation.

The sparse checked out directories are presented in the ***Working Copy view*** with a marker corresponding to each depth value, in the top left corner, as follows:

- ⬛ - **Recursive (*infinity*)** - This is the default value and it is has no mark. The directory has no limiting depth.
- ⬛ - **Immediate children (*immediates*)** - The directory is limited to direct child directories (without contents) and files.
- ⬛ - **File children only (*files*)** - The directory is limited to direct child files only.
- ⬛ - **This folder only (*empty*)** - The directory has *empty* depth set.

A depth set on a directory means that some operations process only items within the specified depth range. For example, **Synchronize** on a working copy directory reports the repository modified items within the depth set on the directory and those existing in the working copy outside of this depth.

The depth information is also presented in the **SVN Information** dialog box and in the tool tip displayed when hovering a directory in the **Working Copy** view.

Syncro SVN Client Views

The main working area occupies the center of the application window, which contains the most important views:

- *Repositories View*
- *Working Copy View*
- *History View*
- *Console View*

The other views that support the main working area are also presented in this section.

Repositories View

The **Repositories** view allows you to define and manage Apache Subversion™ repository locations and browse repositories. Repository files and folders are presented in a tree view with the repository locations at the first level, where each location represents a connection to a specific repository. More information about each resource is displayed in a tabular form:

- **Date** - Date when the resource was last modified;
- **Revision** - The revision number at which the resource was last time modified;
- **Author** - Name of the person who made the last modification on the resource;
- **Size** - Resource size on disk;
- 🔒 **Lock information** - Information about the lock status of a file. When a repository file is locked by a user the 🔒 icon is displayed in this column. If no icon is displayed the file is not locked. The tooltip of this column displays the details about lock:

  - owner - the name of the user who created the lock;
  - date - the date when the user locked the file;
  - expires on - date when the lock expires. Lock expiry policy is set in the repository options, on the server side;
  - comment - the message attached when the file was locked.

- **Type** - Contains the resource type or file extension.

Figure 237: Repositories View

Toolbar

The **Repositories** view's toolbar contains the following buttons:

- **New Repository Location** - Allows you to enter a new repository location by means of the **Add SVN Repository** dialog.

- **Move Up** - Move the selected repository up one position in the list of repositories in the **Repositories** view.

- **Move Down** - Move the selected repository down one position in the list of repositories in the **Repositories** view.

- **Collapse all** - Collapses all repository trees.

- **Stop** - Stops the current repository browsing operation executed when a repository node is expanded. This is useful when the operation takes too long or the server is not responding.

- **Settings** - Allows you to configure the resource table appearance.

Contextual Menu Actions

The **Repositories** view contextual menu contains different actions depending on the selected item. If a repository location is selected, the following management actions are available:

New Repository Location (Ctrl+Alt+N (Command+Alt+N on OS X))

Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.
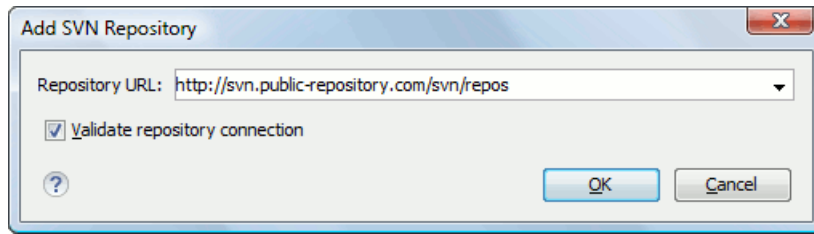
Figure 238: Add SVN Repository Dialog Box

If the **Validate repository connection** option is selected, the URL connection is validated before being added to the **Repositories** view.

Edit Repository Location (Ctrl+Alt+E (Command+Alt+E on OS X))

Context-dependent action that allows you to edit the selected repository location using the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.

Change the Revision to Browse (Ctrl+Alt+Shift+B (Command+Alt+Shift+B on OS X))

Context-dependent action that allows you to change the selected repository revision using the **Change the Revision to Browse** dialog. It is active only when a repository location root is selected.

Remove Repository Location (Ctrl+Alt+Shift+R (Command+Alt+Shift+R on OS X))

Allows you to remove the selected repository location from the view. It shows you a confirmation dialog before removal. It is active only when a repository location root is selected.

The following actions are common to all repository resources:

Open

Opens the selected file in the Editor view in read-only mode.

Open with

Displays the **Open with...** dialog to specify the editor in which the selected file is opened. In case multiple files are selected, only external applications can be used to open the files.

Save as

Saves the selected files locally, as they are in the browsed revision.

Refresh

Refreshes the resource selected in the **Repositories** view.

Check out (Ctrl+Alt+Shift+C (Command+Alt+Shift+C on OS X))

Allows you to create a working copy from a repository directory, on your local file system. To read more about this operation, see the section *Check out a working copy*.

Branch/Tag

Allows you to create a branch or a tag from the selected folder in the repository. To read more about how to create a branch/tag, see the *Creation and management of Branches/Tags* section.

Share project

Allows you to *share a new project* using an SVN repository. The local project is automatically converted into an SVN working copy.

Import sub-menu:

Import Folder Content (Ctrl+Alt+Shift+M (Command+Alt+Shift+M on OS X))

Depending on the selected folder from a repository, allows you to import the contents of a specified folder from the file system into it. To read more about this operation, see the section *Importing resources into a repository*.

> **Note:** The difference between **Import Folder Content** and **Share project** actions is that the latter one also converts the selected directory into a working copy.

Import File(s) (Ctrl+Alt+I (Command+Alt+I on OS X))

Imports the files selected from the files system into the selected folder from the repository.

Export

Exports a folder from the repository to the local file system.

Show History (Ctrl+H (Command+H on OS X))

Displays the history of the selected resource. At the start of the operation, you can set filtering options.

Show Annotation (Ctrl+Shift+A (Command+Shift+A on OS X))

Complex action that does the following operations:

- opens the selected resource in the **Annotations** editor
- displays corresponding annotations list in the **Annotations** view
- displays the history of the selected resource

Revision Graph (Ctrl+Shift+G (Command+Shift+G on OS X))

This action allows you to see the graphical representation of a resource history. For more details about a resource revision graph see the section *Revision Graph*. This operation is enabled for any resource selected into the **Repositories** view or **Working Copy** view.

Copy URL Location (Ctrl+Alt+U (Command+Alt+U on OS X))

Copies to clipboard the URL location of the selected resource.

Copy to

Copies to a specified location the currently selected resource(s). This action is also available when you browse other revisions than the latest one (*HEAD*), to allow restoring previous versions of an item.

Move to (Ctrl+M (Command+M on OS X))

Moves to a specified location the currently selected resource(s).

Rename ((F2))

Renames the selected resource.

Delete ((Delete))

Deletes selected items from the repository via an immediate commit.

New Folder

Allows you to create a folder in the selected repository path (available only for folders).

Locking

(available only for files):

Lock (Ctrl+K (Command+K on OS X))

Allows you to lock certain files for which you need exclusive access. For more details on the use of this action, see *Locking a file*.

Unlock (Ctrl+Shift+K (Command+Shift+K on OS X))

Releases the exclusive access to a file from the repository. You can also choose to unlock it by force (*break the lock*).

Show SVN Properties (Ctrl+Shift+P (Command+Shift+P on OS X))

Brings up the *Properties view* displaying the SVN properties for the selected resource. This view does not allow adding, editing, or removing SVN properties of a repository resource. These operations are allowed only for working copy resources.

File Information (Ctrl+I (Command+I on OS X))

Provides additional information for the selected resource. For more details, see the section *Information view*.

Assistant Actions

When there is no repository configured, the **Repositories** view mode lists the following two actions:
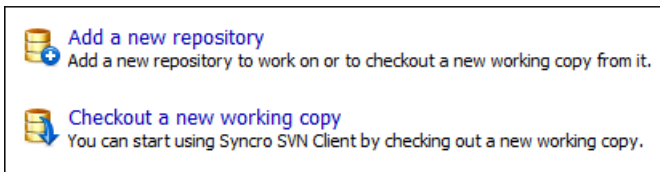
Drag and Drop Operations

The structure of the files tree can be changed with drag and drop operations inside the **Repositories** view. These operations behave in the same way with the **Copy to**/**Move to** operations.

Working Copy View

The **Working Copy** view allows you to manage the content of an SVN working copy.

The toolbar contains:

- the list of defined working copies
- a set of view modes that allow you to filter the content of the working copy based on the resource status (like incoming or outgoing changes)
- **Settings** menu
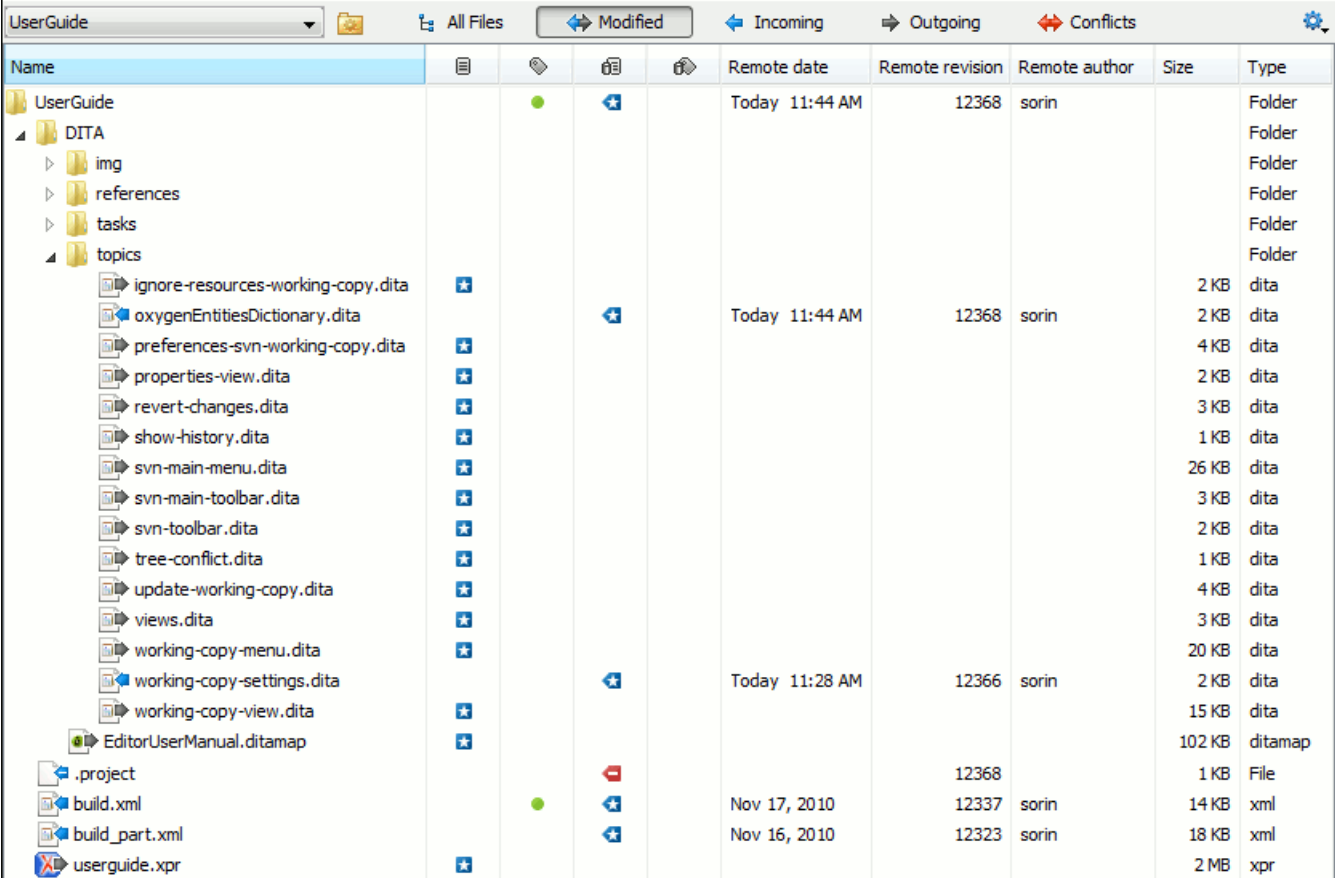
If you click any of the view modes (**All Files**, **Modified**, **Incoming**, **Outgoing**, **Conflicts**), the information displayed changes as follows:

- **All Files** - Resources (files and folders) are presented in a hierarchical structure with the root of the tree representing the location of the working copy on the file system. Each resource has an icon representation which describes the type of resource and also depicts the state of that resource with a small overlay icon.

Figure 239: Working Copy View - All Files View Mode

- **Modified** - The resource tree presents resources modified locally (including those with conflicting content) and remotely. Decorator icons are used to differentiate between various resource states:

  - - incoming modification from repository:

    - - file content or properties modified remotely;

    - - new file added remotely;

    - - file deleted remotely;

  - - outgoing modification to repository:

    - - file content or properties modified locally;

    - - new file added locally;

    - - file deleted locally;

  - - pseudo-conflict state - a resource being locally and remotely modified at the same time, or a parent directory of such a resource.

  - - real conflict state - a resource that had both incoming and outgoing changes and not all the differences could be merged automatically through the update operation (manually editing the local file is necessary for resolving the conflict).

Figure 240: Working Copy View - Modified View Mode

- **Incoming** - The resource tree presents only incoming changes.
- **Outgoing** - The resource tree presents only outgoing changes.

- ⬌ **Conflicts** - The resource tree presents only conflicting changes (real conflicts and pseudo-conflicts).

The following columns provide information about the resources:

- **Name** - Resource name. Resource icons can have the following decorator icons:

  - Additional status information:

    - 📁 **Propagated modification marker** - A folder marked with this icon indicates that the folder itself presents some changes (like modified properties) or a child resource has been modified.

    - 📁 **External** - This indicates a mapping of a local directory to the URL of a versioned resource. It is declared with a `svn:externals` property in the parent folder and it indicates a working copy not directly related with the parent working copy that defines it.

    - 📁 **Switched** - This indicates a resource that has been switched from the initial repository location to a new location within the same repository. The resource goes to this state as a result of *the Switch action* executed from the contextual menu of the Working Copy view.

    - 🔧 *Grayed* - A resource with a grayed icon but no overlaid icon is an ignored resource. It is obtained with the **Add to svn:ignore** action.

  - Current SVN depth of a folder:

    - 📁 **Immediate children (immediates)** (a variant of *sparse checkout*) - The directory contains only direct file and folder children. Child folders ignore their content.

    - 📁 **File children only (files)** (a variant of *sparse checkout*) - The directory contains only direct file children, disregarding any child folders.

    - 📁 **This folder only (empty)** (a variant of *sparse checkout*) - The directory discards any child resource.

    - 📝 **Note:**

      - Any folder not marked with one of the depth icons, has recursive depth (*infinity*) set by default (presents all levels of child resources).
      - Although folders not under version control can have no depth set, Oxygen XML Author presents *unversioned* and *ignored* folders with *empty* depth when **Show unversioned directories content** or **Show ignored directories content** options are disabled.

- 📄 **Local file status** - Shows the changes of working copy resources that were not committed to the repository yet. The following icons are used to mark resource status:

  - 🔵 - Resource is *not under version control* (*unversioned*).
  - ⬜ - Resource is being *ignored* because it is not under version control and its name matches a file name pattern defined in one of the following places:

    - *global-ignores* section in the SVN client-side *config file*;

      - ⚠️ **Attention:** If you don't explicitly set the `global-ignores` runtime configuration option - either to your preferred set of patterns or to an empty string - Subversion uses the default value.

    - *Application global ignores option* of Oxygen XML Author;
    - the value of a `svn:ignore property` set on the parent folder of the resource being ignored.

  - ➕ - Marks a newly created resource, *scheduled for addition* to the version control system.
  - ➕ - Marks a resource *scheduled for addition*, created by copying a resource already under version control and inheriting all its SVN history.
  - ➕ - The content of the resource has been *modified*.

- **R** - Resource has been *replaced* in your working copy (the file was scheduled for deletion, and then a new file with the same name was scheduled for addition in its place).
- **−** - Resource is *deleted*(scheduled for deletion from **Repository** upon the next commit).
- **!** - The resource is *incomplete* (as a result of an interrupted *check out* or *update* operation).
- **!** - The resource is *missing* because it was moved or deleted without using an SVN-aware application.
- **C** - The contents of the resource is in *real conflict state*.
- **tc** - Resource is in *tree conflict* state after an update operation because:

    - Resource was locally modified and incoming deleted from repository;
    - Resource was locally scheduled for deletion and incoming modified.

- **O** - Resource is *obstructed* (versioned as one kind of object: file, directory, or symbolic link, but has been replaced outside Syncro SVN Client by a different kind of object).

- **Local properties status** - Marks the resources that have SVN properties, with the following possible states:

    - **●** - The resource has SVN properties set.
    - **◉** - The resource properties have been modified.
    - **◉** - Properties for this resource are in *real conflict* with property updates received from the repository.

- **Revision** - The current revision number of the resource.
- **Date** - Date when the resource was last time modified on the disk.
- **BASE Revision** - The revision number of the pristine version of the resource.
- **BASE Date** - Date when the pristine version of the resource was last time committed in the repository.
- **Author** - Name of the person who made the last modification on the pristine version of the resource.
- **Remote file status** - Shows changes of resources recently modified in the repository. The following icons are used to mark incoming resource status:

    - **◁** - Resource is newly added in repository.
    - **◁** - The content of the resource has been modified in repository.
    - **◁** - Resource was replaced in repository.
    - **◁** - Resource was deleted from repository.

- **Remote properties status** - Resources marked with the ◉ icon have incoming modified properties from the repository.
- **Remote revision** - Revision number of the resource latest committed modification.
- **Remote date** - Date of the resource latest modification committed on the repository.
- **Remote author** - Name of the author who committed the latest modification on the repository.
- **Lock information** - Shows the lock state of a resource. The lock mechanism is a convention intended to help you signal other users that you are working with a particular set of files. It minimizes the time and effort wasted in solving possible conflicts generated by clashing commits. A lock gives you exclusive rights over a file, only if other users follow this convention and they do not try to bypass the lock state of a file.

    A folder can be locked only by the SVN client application, completely transparent to the user, if an operation in progress was interrupted unexpectedly. As a result, folders affected by the operation are marked with the 🔒 symbol. To clear the locked state of a folder, use the **Clean up** action.

    📄 **Note:** Users can lock only files.

    The following lock states are displayed:

    - *no lock* - the file is not locked. This is the default state of a file in the SVN repository;
    - *remotely locked* (🔒) - shown when:

        - another user has locked the file in the repository;
        - the file was locked by the same user from another working copy;
        - the file was locked from the **Repositories** view.

If you try to commit a new revision of the file to the repository, the server does not allow you to bypass the file lock.

> **Note:** To commit a new revision, you need to wait for the file to be unlocked. Ultimately, you might try to *break* or *steal* the lock, but this is not what other users expect. Use these actions carefully, especially when you are not the file lock owner.

- *locked* (🔒) - displayed after you have locked a file from the current working copy. Now you have exclusive rights over the corresponding file, being the only one who can commit changes to the file in the repository.

  > **Note:** Working copies keep track of their locked files, so the locks are presented between different sessions of the application. Synchronize your working copy with the repository to make sure that the locks are still valid (not *stolen* or *broken*).

- *stolen* (🔒) - a file already locked from your working copy is being locked by another user. Now the owner of the file lock is the user who stole the lock from you.
- *broken* (🔓) - a file already locked from your working copy is no longer locked in the repository (it was unlocked by another user).

  > **Note:** To remove the *stolen* or *broken* states from your working copy files, you have to **Update** them.

If one of your working copy files is locked, hover the mouse pointer over the lock icon to see more information:

- lock type - current file lock state;
- owner - the name of the user who created the lock;
- date - the date when the user locked the file;
- expires on - date when the lock expires. Lock expiry policy is set in the repository options, on the server side;
- comment - the message attached when the file was locked.

- **Size** - Resource size on disk;
- **Type** - Contains the resource type or file extension.

> **Note:** The working copy table allows you to show or hide any of its columns and also to sort its contents by any of the displayed columns. The table header provides a contextual menu which allows you to customize the displayed information.

The toolbar allows you to switch between two working copies:

- Drop down list - Contains all the working copies Oxygen XML Author is aware of. When you select another working copy from the list, the newly selected working copy content is scanned and displayed in the **Working Copy** view.

- 📁 (📁 on Mac OS X) **Working Copies Manager** - opens a dialog box that displays the working copies Oxygen XML Author is aware of. In this dialog box, you can add existing working copies or remove those you no longer need. If you try to add a folder which is not a valid Subversion working copy, Oxygen XML Author warns you that the selected directory is not under version control.

  > **Note:** Removing a working copy from this dialog does NOT remove it from your file system; you will have to do that manually.

Working Copy Settings

The **Settings** button from the toolbar of the **Working Copy** view provides the following options:

- **Show unversioned directories content** - displays the content of unversioned directories;

  > **Note:** In case this option is disabled, it will be ignored for items that, after a synchronize, are reported as incoming from the repository. This applies for all working copy modes, except **All Files**.

- **Show ignored items** - displays the ignored resource when **All Files** mode is selected;

- **Show ignored directories content** - displays the content of ignored directories when **All Files** mode is selected;

> **Note:** Although *ignored* items are not presented in the **Modified**, **Incoming**, and **Conflicts** modes, they will be if, after a synchronize, they are reported as incoming from the repository.

- **Show deleted items** - displays the deleted resource when **All Files** mode is selected. All other modes always display deleted resources, disregarding this option;
- **Tree** / **Compressed** / **Flat** - affect the way information is displayed inside the **Modified**, **Incoming**, **Outgoing**, and **Conflicts** view modes;
- **Configure columns** - allows you to customize the structure of the **Working Copy** view data. This action opens the following dialog box:
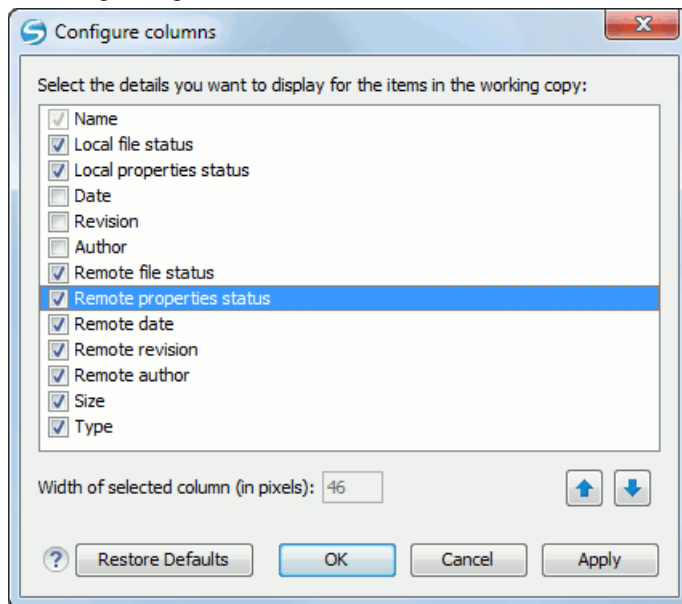
Figure 241: Configure Columns of Working Copy View

The order of the columns can be changed with the two arrow buttons. The column size can be edited in the **Width of selected column** field. The **Restore Defaults** button reverts all columns to the default order, width and enabled/disabled state from the installation of the application.

Working Copy Format

When an SVN working copy is loaded, Syncro SVN Client first checks the format of the working copy:

- if the format is older than SVN 1.7, you are prompted to upgrade it to SVN 1.8 in order to load it;
- if the format is 1.7, Syncro SVN Client takes into account the state of the *When loading an old format working copy* option.

To change how working copy formats are handled, *open the Preferences dialog* and go to **SVN** > **Working copy**, in the *Administrative area* section.

> **Note:**
>
> - The format of the working copy can be downgraded or upgraded at any time with the **Upgrade** and **Downgrade** actions available in the **Tools** menu. These actions allow switching between SVN 1.7 and SVN 1.8 working copy formats.
>
> - SVN 1.7 working copies cannot be downgraded to older formats.

Refresh a Working Copy

A refresh is a frequent operation triggered automatically when you switch between two working copies using the toolbar selector of the **Working Copy** view and when you switch between Oxygen XML Author and other applications.

The **Working Copy** view features a fast refresh mechanism: the content is cached locally when loading the working copy for the first time. Later on, when the same working copy is displayed again, the application uses this cache to detect the changes between the cached content and the current content found on disk. The refresh operation is run on these changes only, thus improving the response time. improvement is noticeable especially when working with large working copies.

Contextual Menu Actions

The contextual menu in the **Working Copy** view contains the following actions:

Edit conflict (Ctrl (Command on OS X) + E)

> Opens the **Compare** editor, allowing you to modify the content of the currently conflicting resources. For more information on editing conflicts, see *Edit conflicts*.

Open in Compare Editor (Ctrl (Command on OS X) + Alt + C)

> Displays changes made in the currently selected file.

Open (Ctrl (Command on OS X) + O)

> Opens the selected resource from the working copy. Files are opened with an internal editor or an external application associated with that file type, while folders are opened with the default file system browsing application (e.g. Windows Explorer on Windows, Finder on OS X, etc).

Open with

> Submenu that allows you to open the selected resource either with Oxygen XML Author or with another application.

Show in Explorer/Show in Finder

> Opens the parent directory of the selected working copy file and selects the file.

Expand all (Ctrl (Command on OS X) + Alt + X)

> Displays all descendants of the selected folder. You can obtain a similar behavior by double-clicking on a collapsed folder.

Refresh (F5)

> Re-scans the selected resources recursively and refreshes their status in the working copy view.

Synchronize (Ctrl (Command on OS X) + Shift + S)

> Connects to the repository and determines the working copy and repository changes made to the selected resources. The application switches to **Modified** view mode if the *Always switch to 'Modified' mode* option is selected.

Update (Ctrl (Command on OS X)+ U)

> Updates the selected resources to the *HEAD* revision (latest modifications) from the repository. If the selection contains a directory, it will be updated depending on its depth.

Update to revision/depth

> Allows you to update the selected resources from the working copy to an earlier revision from the repository. You can also select the update *depth* for the current folder. You can find out more about the *depth* term in the *sparse checkouts* section.

Commit

> Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what to commit by selecting or not resources. A directory will always be committed recursively. The unversioned resources will be deselected by default. In the commit dialog you can also enter a commit comment before sending your changes to the repository.

Revert (Ctrl (Command on OS X) + Shift + V)

> Undoes all local changes for the selected resources. It does not contact the repository, the files are obtained from Apache Subversion™ pristine copy. It is enabled only for modified resources. See *Revert your changes* for more information.

Override and Update

> Drops any outgoing change and replaces the local resource with the HEAD revision. Action available on resources with outgoing changes, including the conflicting ones. See the *Revert your changes* section.

Override and Commit

Drops any incoming changes and sends your local version of the resource to the repository. Action available on conflicting resources. See also the section *Drop incoming modifications*.

✔ **Mark Resolved (Ctrl (Command on OS X) + Shift + R)**

Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.

✔ **Mark as Merged (Ctrl (Command on OS X) + Shift + M)**

Instructs the Subversion system that you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the *Merge conflicts* section for more information about how you can solve the pseudo-conflicts.

✖ **Create patch (Ctrl (Command on OS X) + Alt + P)**

Allows you to create a file containing all the differences between two resources, based on the svn diff command. To read more about creating patches, see *the section about patches*.

Compare with:

- **Latest from HEAD (Ctrl (Command on OS X) + Alt + H)** - Performs a 3-way diff operation between the selected file and the *HEAD* revision from the repository and displays the result in the **Compare view**. The common ancestor of the 3-way diff operation is the *BASE* version of the file from the local working copy.

- **BASE revision (Ctrl (Command on OS X) + Alt + C)** - Compares the working copy file with the BASE revision file (the so-called *pristine copy*).

- **Revision (Ctrl (Command on OS X) + Alt + R)** - Shows the **History view** containing the log history of that resource.

- **Branch/Tag** - Compares the working copy file with a revision of the file from a branch or tag. The revision is specified by URL (selected with a repository browser dialog) and revision number (selected with a revision browser dialog).

- **Each other** - Compares two selected files with each other.

These *compare* actions are enabled only if the selected resource is a file.

Replace with:

- **Latest from HEAD** - Replaces the selected resources with their versions from the *HEAD* revision of the repository.

- **BASE revision** - Replace the selected resources with their versions from the pristine copy (the BASE revision).

📄 **Note:** In some cases it is impossible to replace the current selected resources with their versions from the *BASE/HEAD* revision:

- for **Replace with BASE revision** action, the resources being unversioned or added have no *BASE* revision, so they cannot be replaced. However, they will be deleted if the action is invoked on a parent folder. The action will never work for missing folders or for obstructing files (folders being obstructed by a file), because you cannot recover a tree of folders

- for **Replace with latest from HEAD** action, you must be aware that there are cases when resources will be completely deleted or reverted to BASE revision and after that updated to HEAD revision, in order to avoid conflicts. These cases are:

  - the resource is *unversioned*, *added*, *obstructed* or *modified*
  - the resource is affected by a svn:ignore or svn:externals property which is locally added on the parent folder and not yet committed to the repository

Show History (Ctrl (Command on OS X) + H)

Displays the **History view** where the log history for the selected resource will be presented. For more details about resource history see the sections about *the resource history view* and *requesting the history for a resource*.

Show Annotation (Ctrl (Command on OS X) + Shift + A)

It will display the **Annotations view** where all the users that modified the selected resource will be presented together with the specific lines and revision numbers modified by each user. For more details about resource annotations see *Annotations View*.

Revision Graph (Ctrl (Command on OS X) + G)

This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see *Revision Graph*.

Copy URL Location (Ctrl (Command on OS X) + Alt + U)

Copies the encoded URL of the selected resource from the Working Copy to the clipboard.

Mark as copied

You can use this action to mark an item from the working copy as a copy of an other item under version control, when the copy operation was performed outside of an SVN client. The **Mark as copied** action is available when you select two items (both the new one and the source one) and depends on the state of the source item.

Mark as moved

You can use this action to mark an item from the working copy as being moved from another location of the working copy, when the move operation was performed outside of an SVN client. The **Mark as moved** action is available when you select two items from different locations (both the new one and the source one (usually reported as *missing*)) and depends on the state of the source item.

Mark as renamed

You can use this action to mark an item from the working copy as being renamed outside of an SVN client. The **Mark as renamed** action is available when you select two items from the same directory (both the new one and the source one (usually reported as *missing*)) and depends on the state of the source item.

Copy to

Copies to a specified location the currently selected resource(s).

Move to Ctrl+M (Command+M on OS X)

Moves to a specified location the currently selected resource(s).

Rename (F2)

You can only rename one resource at a time. As for the move command, a copy of the original resource will be made with the new name and the original will be marked as deleted.

Delete (Delete)

Schedules selected items for deletion upon the next commit, removing them from the disk. Depending on the state of each item, you are prompted to confirm the operation.

New:

> **New File**
>
> Creates a new file inside the selected folder. The newly created file will be added under version control only if the parent folder is already versioned.
>
> **New Folder (Ctrl (Command on OS X)+ Shift + F)**
>
> Creates a child folder inside the selected folder. The newly created folder will be added under version control only if its parent is already versioned.
>
> **New External Folder (Ctrl (Command on OS X) + Shift + W)**
>
> This operation allows you to add a new external definition on the selected folder. An externals definition is a mapping of a local directory to the URL - and ideally a particular revision - of a versioned directory, stored in the svn:externals property of the selected folder.

The URLs used in the externals definition format can be relative. You can specify the repository URLs to which the external folders point to using the following relative formats:

- **../** - Relative to the URL of the directory on which the `svn:externals` property is set.
- **^/** - Relative to the root of the repository in which the `svn:externals` property is versioned.
- **//** - Relative to the scheme of the URL of the directory on which the `svn:externals` property is set.
- **/** - Relative to the root URL of the server on which the `svn:externals` property is versioned.

> **Important:** To change the target URL of an external definition or to delete an external item, do the following:
>
> - modify or delete the item's definition found in the `svn:externals` property set on the parent folder
> - use the **Update** operation on the item's parent folder for the change to take effect

> **Note:** Syncro SVN Client does not support definitions of local relative externals.

Add to version control (Ctrl (Command on OS X) + Alt + V)

Allows you to schedule for addition resources that are not under version control. For further details, see *Add Resources to Version Control* section.

Remove from version control

Schedules selected items for deletion from repository upon the next commit. The items are not removed from the file system after committing.

Add to "svn:ignore" (Ctrl (Command on OS X) + Alt + I)

Allows you to keep inside your working copy files that should not participate to the version control operations. This action can only be performed on resources not under version control. It actually modifies the value of the `svn:ignore` property of the resource's parent directory. Read more about this in the *Ignore Resources Not Under Version Control* section.

Clean up (Ctrl (Command on OS X) + Shift + C)

Performs a maintenance cleanup operation to the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. Useful when you already know where the problem originated and want to fix it as quickly as possible. Only active for resources under version control.

Locking:

- **Scan for locks (Ctrl (Command on OS X) + L)** - Contacts the repository and recursively obtains the list of locks for the selected resources. A dialog containing the locked files and the lock description will be displayed. Only active for resources under version control. For more details see *Scanning for locks*.

- **Lock (Ctrl (Command on OS X) + K)** - Allows you to lock certain files for which you need exclusive access. You can write a comment describing the reason for the lock and you can also force (*steal*) the lock. The action is active only on files under version control. For more details on the use of this action see *Locking a file*.

- **Unlock (Ctrl (Command on OS X) + Alt + K)** - Releases the exclusive access to a file from the repository. You can also choose to unlock it by force (*break the lock*).

Show SVN Properties (Ctrl (Command on OS X) + P)

Brings up the *Properties view* and displays the SVN properties for the selected resource.

File Information (Ctrl (Command on OS X) + I)

Provides additional information for the selected resource from the working copy. For more details please see *Obtain information for a resource*.

Drag and Drop Operations

The structure of the files tree can be changed with drag and drop operations inside the **Working Copy** view. These operations behave in the same way with the **Copy to**/**Move to** operations.

Also, files and folders can be added to the file tree of the view as *unversioned* resources by drag and drop operations from other applications (for example from Windows Explorer or Mac OS X Finder). In this case, the items from the file system are only copied, without removing them from their original location.

⚠ **Attention:** When you drag items from the working copy to a different application, the performed operation is controlled by that application. This means that the moved items are left as *missing* in the working copy (items are moved in the file system only, but no SVN versioning meta-data is changed).

Assistant Actions

To ensure a continuous and productive work flow, when a view mode has no files to present, it offers a set of guiding actions with some possible paths to follow.

Initially, when there is no working copy configured the **All Files** view mode lists the following two actions:

Check out a new working copy
You can start using Syncro SVN Client by checking out a new working copy.

Add a working copy
If you already have a working copy on disk, you can add it to Syncro SVN Client and start to work.
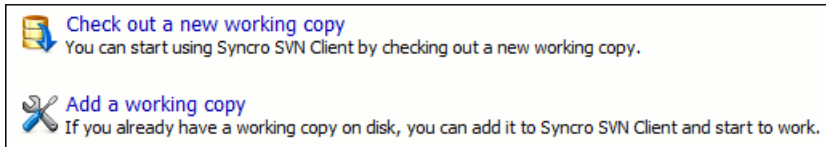
Figure 242: All Files Panel

For **Modified**, **Incoming**, **Outgoing**, **Conflicts** view modes, the following actions may be available, depending on the current working copy state in different contexts:

- ℹ Information message - Informs you why there are no resources presented in the currently selected view mode;

- ⟳ **Synchronize with Repository** - Available only when there is nothing to present in the **Modified** and **Incoming** view modes;

- ⬅ **Switch to Incoming** - Selects the **Incoming** view mode.

- ➡ **Switch to Outgoing** - Selects the **Outgoing** view mode.

- ⬌ **Switch to Conflicts** - Selects the **Conflicts** view mode.

- ⬆ **Show all changes/incoming/outgoing/conflicts** - Depending on the currently selected view mode, this action presents the corresponding resources after a synchronize operation was executed only on a part of the working copy resources.

History View

In Apache Subversion™, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision you can use the **History view** that can be accessed from *Repositories view*, *Working Copy view*, *Revision Graph*, or *Directory Change Set view*. From the **Working copy view** you can display the history of local versioned resources.

The view consists of four distinct areas:

- The table showing details about each revision, like: revision number, commit date and time, number of changes (more details available in the tooltip), author's name, and a fragment of the commit message.
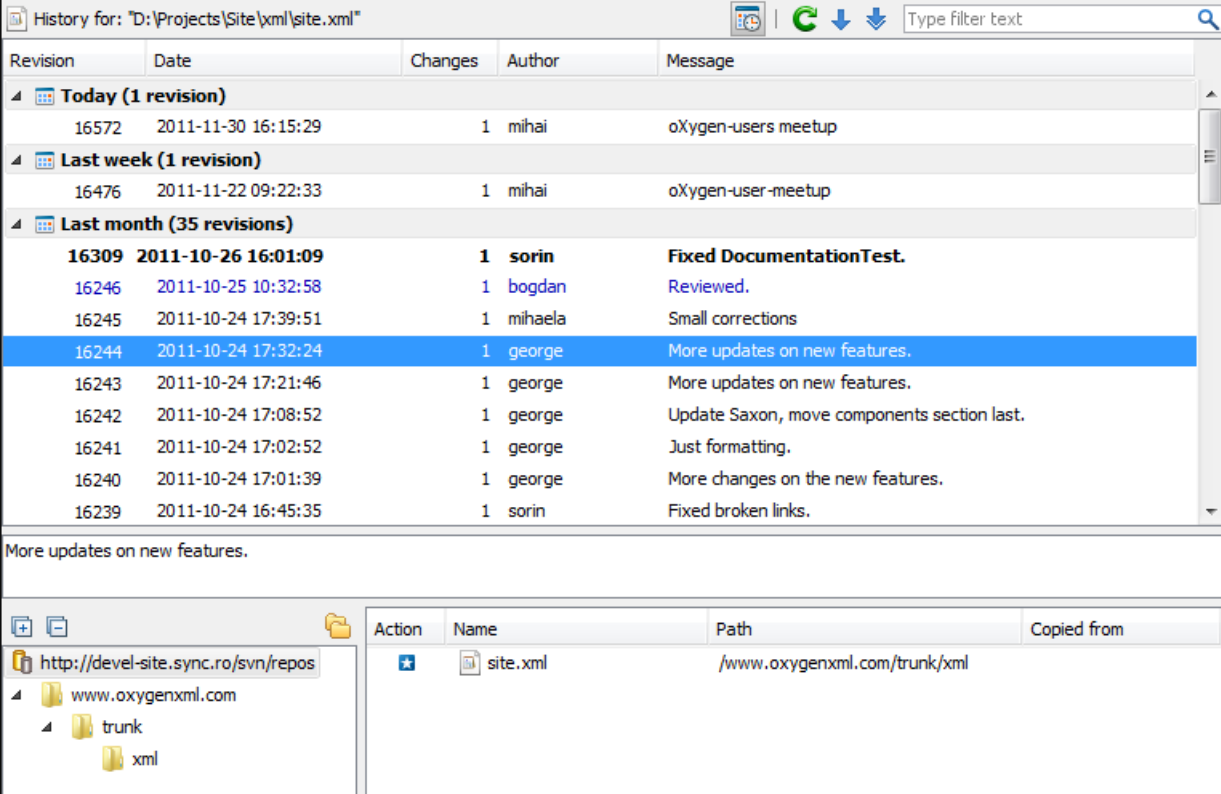
  Some revisions may be highlighted to emphasize:

  - the current revision of the resource for which the history is displayed - a bold font revision;
  - the last revision in which the content or properties of the resource were modified - blue font revision.

    📝 **Note:** Both font highlights may be applied for the same revision.

- The complete commit message for the selected revision;

- A tree structure showing the folders where the modified resources are located. You can compress this structure to a more compact form that focuses on the folders that contain the actual modifications;

- The list of resources modified in the selected revision. For each resource, the type of action done against it is marked with one of the following symbols:

  - ⊞ - A newly created resource;
  - ⊞ - A newly created resource, copied from another repository location;
  - ⊞ - The content/properties of the resource were *modified*;
  - ⓡ - Resource was *replaced* in the repository;
  - ⊟ - Resource was deleted from the repository.

Figure 243: History View

You can group revisions in predefined time frames (today, yesterday, this week, this month), by pressing the 🕐 **Group by date** button from the toolbar.

The History Filter Dialog

The **History view** does not always show all the changes ever made to a resource because there may be thousands of changes and retrieving the entire list can take a long time. Normally you are interested in the more recent ones. That is why you can specify the criteria for the revisions displayed in the **History view** by selecting one of several options presented in the **History** dialog which is displayed when you invoke the **Show History** action.
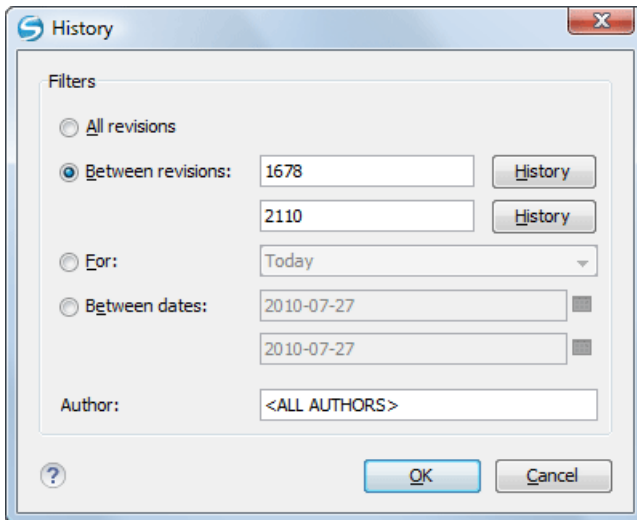
Figure 244: History Filters Dialog

Options for the set of revisions presented in the History view are:

- all revisions of the selected resource;

- only revisions between a start revision number and an end revision number;

- only revisions added in a period of time like today, last week, last month, etc.;

- only revisions between a start and an end date;

- only revisions committed by a specified SVN user.

The toolbar of the **History view** has two buttons for extending the set of revisions presented in the view: **Get next 50** and **Get all**.

The History Filter Field

When only the history entries which contain a specified substring need to be displayed in the **History view** the filter field displayed at the top of this view is the perfect fit. Just enter the search string in the field next to the label **Find**. Only the items with an author name, commit message, revision number or date which match the search string are kept in the **History view**. The filter action is executed and the content of the table is updated when the button 🔍 **Search** is pressed.

Features

Single selection actions:

Compare with working copy

Compares the selected revision with your working copy file. It is enabled only when you select a file.

Open

Opens the selected revision of the file into the Editor. This is enabled only for files.

Open with

Displays the **Open with...** dialog to specify the editor in which the selected file will be opened.

Get Contents

Replaces the current version from the working copy with the contents of the selected revision from the history of the file. The *BASE* version of the file is not changed in the working copy so that after this action the file will appear as modified in a synchronization operation, that is newer than the *BASE* version, even if the contents is from an older version from history.

Save as

Allows you to save the contents of a file as it was committed at a certain revision. This option is available only when you access the history of a file.

Copy to

Copies to the repository the item whose history is displayed, using the selected revision. This option is active only when presenting the history for a repository item (URL).

> **Note:** This action can be used to resurrect deleted items also.

Revert changes from this revision

Reverts changes which were made in the selected revision. The changes are reverted only in your working copy file, so it does not affect the repository file. It does not replace your working copy file with the entire file at the earlier revision, but only rolls-back earlier changes when other unrelated changes have been made since the date of the revision. This action is enabled when the resource history was launched for a local working copy resource.

Update to revision

Updates your working copy resource to the selected revision. This is useful if you want your working copy to reflect a time in the past. It is best to update a whole directory in your working copy, not just one file, otherwise your working copy is inconsistent and you are unable to commit your changes.

Check out

Checks out a new working copy of the directory for which the history is presented, from the selected revision.

Show Annotation Computes the latest revision number and author name that modified each line of the file up to the selected revision, that is no modification later than the selected revision is taken into account.

Change

Allows you to change commit data for a file:

- *Author* - Changes the name of the SVN user that committed the selected revision.
- *Message* - Changes the commit message of the selected revision.

When two resources are selected in the **History** view, the contextual menu contains the following actions:

Compare revisions

When the resource is a file, the action compares the two selected revisions using the **Compare** view. When the resource is a folder, the action displays the set of all resources from that folder that were changed between the two revision numbers.

Revert changes from these revisions

Similar to the `svn-merge` command, it merges two selected revisions into the working copy resource. This action is only enabled when the resource history was requested for a working copy item.

For more information about the **History view** and its features please read the sections *Request history for a resource* and *Using the resource history view*

Directory Change Set View

The result of comparing two reference revisions from the history of a folder resource is a set with all the resources changed between the two revision numbers. The changed resources can be contained in the folder or in a subfolder of that folder. These resources are presented in a tree format. For each changed resource all the revisions committed between the two reference revision numbers are presented.
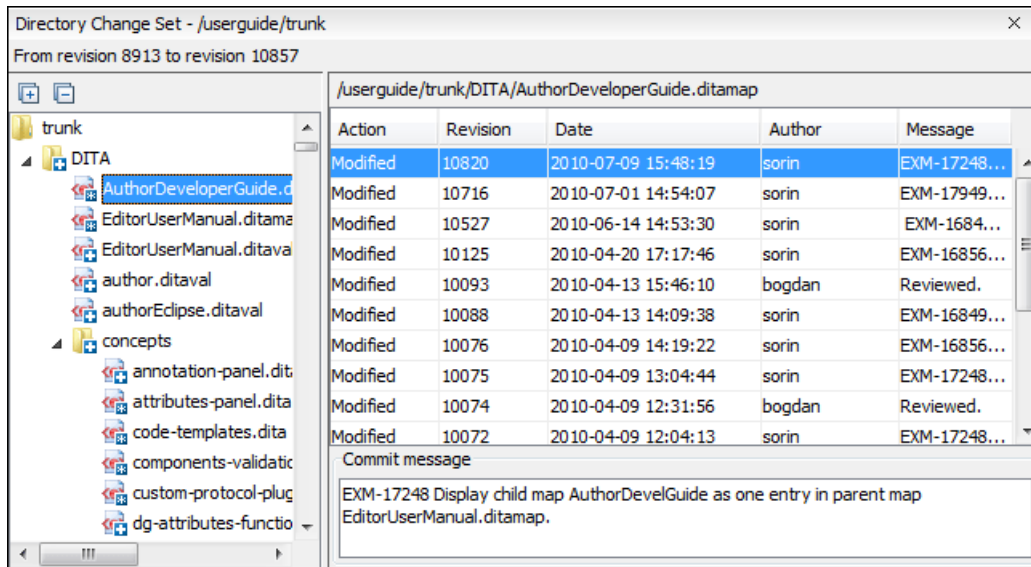
Figure 245: Directory Change Set View

The set of changed resources displayed in the tree is obtained by running the action **Compare revisions** available on the context menu of the **History** view when two revisions of a folder resource are selected in the **History** view.

The left side panel of the view contains the tree hierarchy with the names of all the changed resources between the two reference revision numbers. The right side panel presents the list with all the revisions of the resource selected in the left side tree. These revisions were committed between the two reference revision numbers. Selecting one revision in the list displays the commit message of that revision in the bottom area of the right side panel.

A double click on a file listed in the left side tree performs a diff operation between the two revisions of the file corresponding to the two reference revisions. A double click on one of the revisions displayed in the right side list of the view performs a diff operation between that revision and the previous one of the same file.

The context menu of the right side list contains the following actions:

Compare with previous version

Performs a diff operation between the selected revision in the list and the previous one.

Open

Opens the selected revision in the associated editor type.

Open with

Displays a dialog with the available editor types and allows the user to select the editor type for opening the selected revision.

Save as

Saves the selected file as it was in the selected revision.

Copy to

Copies to the repository the item whose history is displayed, using the selected revision.

> **Note:** This action can be used to resurrect deleted items also.

Check out

Checks out a new working copy of the selected directory, from the selected revision.

Show Annotation

Requests the annotations of the file and *displays them in the Annotations view*.

The Editor Panel of SVN Client

You can open a file for editing in an internal built-in editor. There are default associations between frequently used file types and the internal editors in *the File Types preferences panel*.

The internal editor can be accessed either from the *Working copy view* or from the *History view*. No actions that modify the content are allowed when the editor is opened with a revision from history.

Only one file at a time can be edited in an internal editor. If you try to open another file it will be opened in the same editor window. The editor provides syntax highlighting for known file types. This means that a different color will be used for each recognized token type found in the file. If the file's content type is unknown you will be prompted to choose the proper way the file should be opened.

After editing the content of the file in an internal editor you can save it to disk by using the **Save** action from the *File* menu or the **Ctrl+S (Command+S on OS X)** key shortcut. After saving your file you can see the file changed status in *the Working Copy view*.

If the internal editor associated with a file type is not the XML Editor, then the encoding set in *the preference Encoding for non XML files* is used for opening and saving a file of that type. This is necessary because in case of XML files the encoding is usually declared at the beginning of the XML file in a special declaration or it assumes the default value UTF-8 but in case of non XML files there is no standard mechanism for declaring the file's encoding.

Annotations View

Sometimes you need to know not only what was changed in a file, but also who made those changes. This view displays the author and the revision that changed every line in a file. Just click on a line in the editor panel where the file is opened to see the revision that edited that line last time highlighted in the **History view** and to see all the lines changed by that revision highlighted in the editor panel. Also the entries of the **Annotations view** corresponding to that revision are highlighted. So the **Annotations view**, the **History view** and the editor panel are synchronized. Clicking on a line in one of them highlights the corresponding lines in the other two.

Figure 246: The Annotations View

The annotations of a file are computed with the **Show Annotation** action available on the right click menu of *the History view* and *the Repository view*.

If the file has a very long history, the computation of the annotation data can take long. If you want only the annotations of a range of revisions you can specify the start revision and the end revision of the range in a dialog similar with *the History filter dialog* that will be displayed in *the History view*. The action is called **Show Annotation** and is available on the right click menu of *the Working Copy view*.

Compare View

In the Oxygen XML Author there are three types of files that can be checked for differences: text files, image files and binary files. For the text files and image files you can use the built-in **Compare view**.
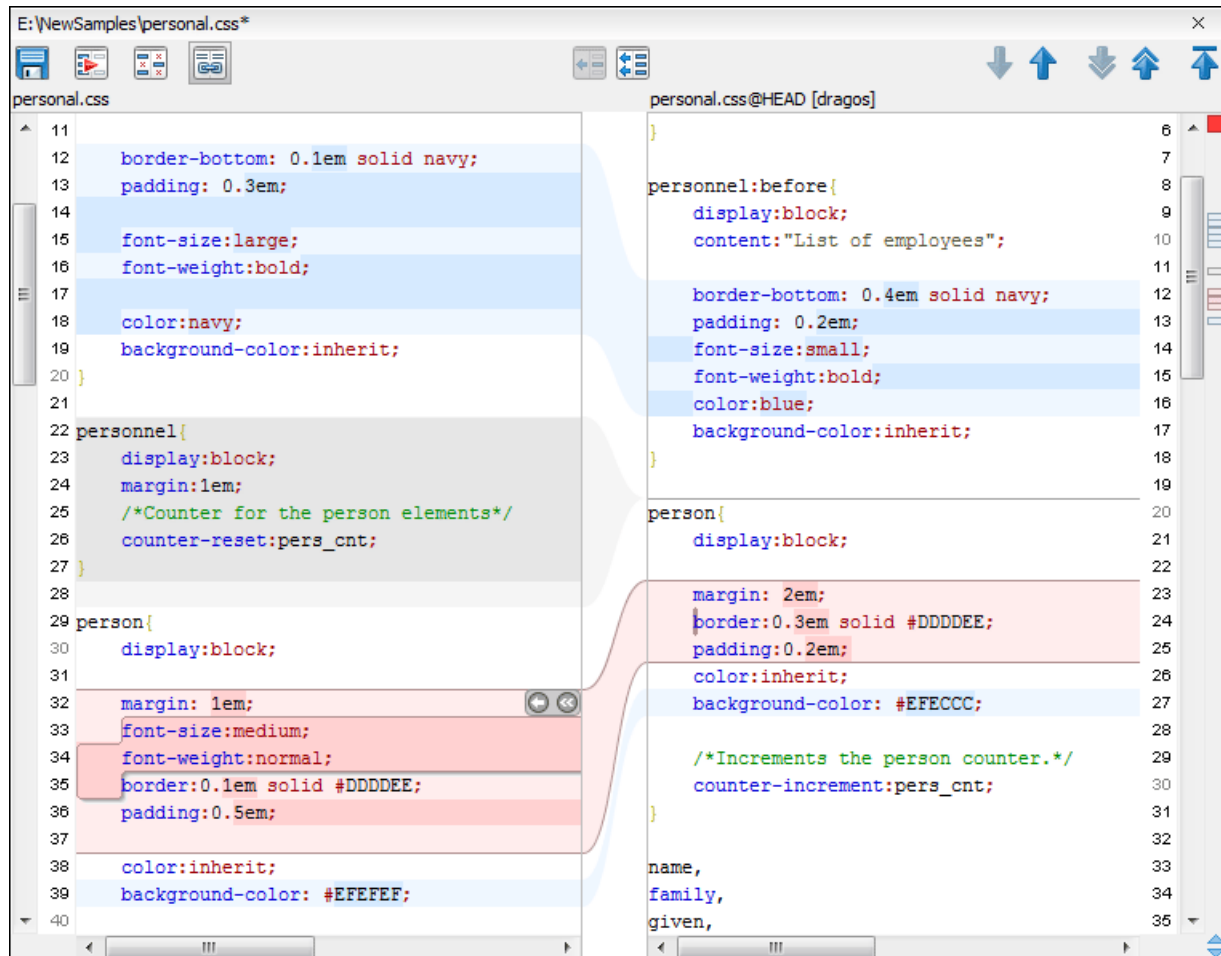

Figure 247: Compare View

At the top of each of the two editors, there are presented the name of the opened file, the corresponding SVN revision number (for remote resources) and the author who committed the associated revision.

When comparing text, the differences are computed using a *line differencing algorithm*. The view can be used to show the differences between two files in the following cases:

- after obtaining the outgoing status of a file with a **Refresh** operation, the view can be used to show the differences between your working file and the pristine copy. In this way you can find out what changes you will be committing;

- after obtaining the incoming and outgoing status of the file with the **Synchronize** operation, you can examine the exact differences between your local file and the *HEAD* revision file;

- you can use the **Compare view** from the **History view** to compare the local file and a selected revision or compare two revisions of the same file.

The Compare view contains two editors. Edits are allowed only in the left editor and only when it contains the working copy file. To learn more about how the view can be used in the day by day work see *View differences*.

Toolbar

The list of actions available in the toolbar consists of:

Save action

Saves the content of the left editor when it can be edited.

Perform Files Differencing

Performs a comparison between the source and target files.

Ignore Whitespaces

Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.

Synchronized scrolling

Synchronizes scrolling of the two open files, so that a selected difference can be seen on both sides of the application window. This action enables/disables the previous described behavior.

Next Block of Changes

Jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes in the document;

> **Note:** A change block groups one or more consecutive lines that contain at least one change.

Previous Block of Changes

Jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes in the document;

Next Change

Jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change.

Previous Change

Jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change.

First Change

Jumps to the first change from the current file.

Copy change from right to left

Copies the selected change from the right editor to the left editor.

Copy all non-conflicting changes from right to left

Copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.

These actions are available also from the *Compare* menu.

Image Preview

You can view your local files by using the built-in **Image preview** component. The view can be accessed from the *Working copy view* or from the *Repository view*. It can also be used from the *History view* to view a selected revision of a image file.

Only one image file can be opened at a time. If an image file is opened in the *Image preview* and you try to open another one it will be opened in the same window. Supported image types are *GIF*, *JPEG/JPG*, *PNG*, *BMP*. Once the image is

displayed in the **Image preview** panel using the actions from the contextual menu one can scale the image at its original size (**1:1** action) or scale it down to fit in the view's available area (**Scale to fit** action).

Compare Images View

The images are compared using the Compare images view. The images are presented in the left and right part of the view, scaled to fit the view's available area. You can use the contextual menu actions to scale the images at their original size or scale them down to fit the view's available area.

The supported image types are: *GIF*, *JPG / JPEG*, *PNG*, *BMP*.

Properties View

The properties view presents Apache Subversion™ properties for the currently selected resource from either the **Working Copy** view or the **Repositories** view.
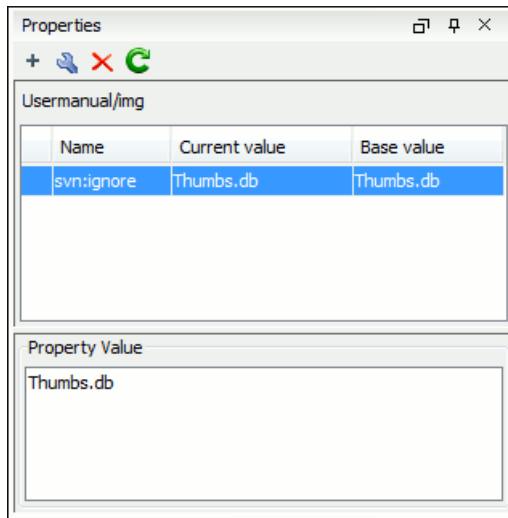
Figure 248: The Properties View

Above the table it is specified the currently active resource for which the properties are presented. Here you will also find a warning when an unversioned resource is selected.

The table in which the properties are presented has four columns:

- **State** - can be one of:
    - (empty) - normal unmodified property, same current and base values;
    - *(asterisk) - modified property, current and base values are different;
    - +(plus sign) - new property;
    - -(minus sign) - removed property.

- **Name** - the property name.
- **Current value** - the current value of the property.
- **Base value** - the base(original) value of the property.

The `svn:externals` Property

The `svn:externals` property can be set on a folder or a file. In the first case it stores *the URL of a folder from other repository*.

In the second case it stores the URL of a file from other repository. The external file will be added into the working copy as a versioned item. There are a few differences between directory and file externals:

- The path to the file external must be in a working copy that is already checked out. While directory externals can place the external directory at any depth and it will create any intermediate directories, file externals must be placed into a working copy that is already checked out.

- The external file URL must be in the same repository as the URL that the file external will be inserted into; inter-repository file externals are not supported.

- While commits do not descend into a directory external, a commit in a directory containing a file external will commit any modifications to the file external.

The differences between a normal versioned file and a file external:

- File externals cannot be moved or deleted; the `svn:externals` property must be modified instead; however, file externals can be copied.

A file external shows up as a X in the switched status column.

Toolbar / Contextual Menu

The properties view toolbar and contextual menu contain the following actions:

- ✛ **Add a new property** - This button invokes the *Add property* dialog in which you can specify the property name and value.

- ⚒ **Edit property** - This button invokes the *Edit property* dialog in which you can change the property value and also see its original(base) value.

- ✕ **Remove property** - This button will prompt a dialog to confirm the property deletion. You can also specify if you want to remove the property recursively.

- ⟳ **Refresh** - This action will refresh the properties for the current resource.

Console View

The **Console View** shows the traces of all the actions performed by the application. Part of the displayed messages mirror the communication between the application and the Apache Subversion™ server. The output is expressed as subcommands to the Subversion server and simulates the Subversion command-line notation. For a detailed description of the Subversion console output read the **SVN User Manual**.

The view has a simple layout, with most of its space occupied by a message area. On its right side, there is a toolbar holding the following buttons:

✷ **Clear**

Erases all the displayed messages.

🔒 **Lock scroll**

Disables the automatic scrolling when new messages are appended in the view.

The maximum number of lines displayed in the console (length of the buffer) can be modified in the *Preferences* page. By default this value is set to 100.

Dynamic Help View

Dynamic Help view is a help window that changes its content to display the help section referring to the currently selected view. As you change the focused view, you are able to read a short description of it and its functionality.

The Revision Graph of a SVN Resource

The history of a SVN resource can be watched on a graphical representation of all the revisions of that resource together with the tags in which the resource was included. The graphical representation is identical to a tree structure and very easy to follow.

The graphical representation of a resource history is invoked with the **Revision graph** action available on the right click menu of a SVN resource in *the Working Copy view* and *the Repository view*.
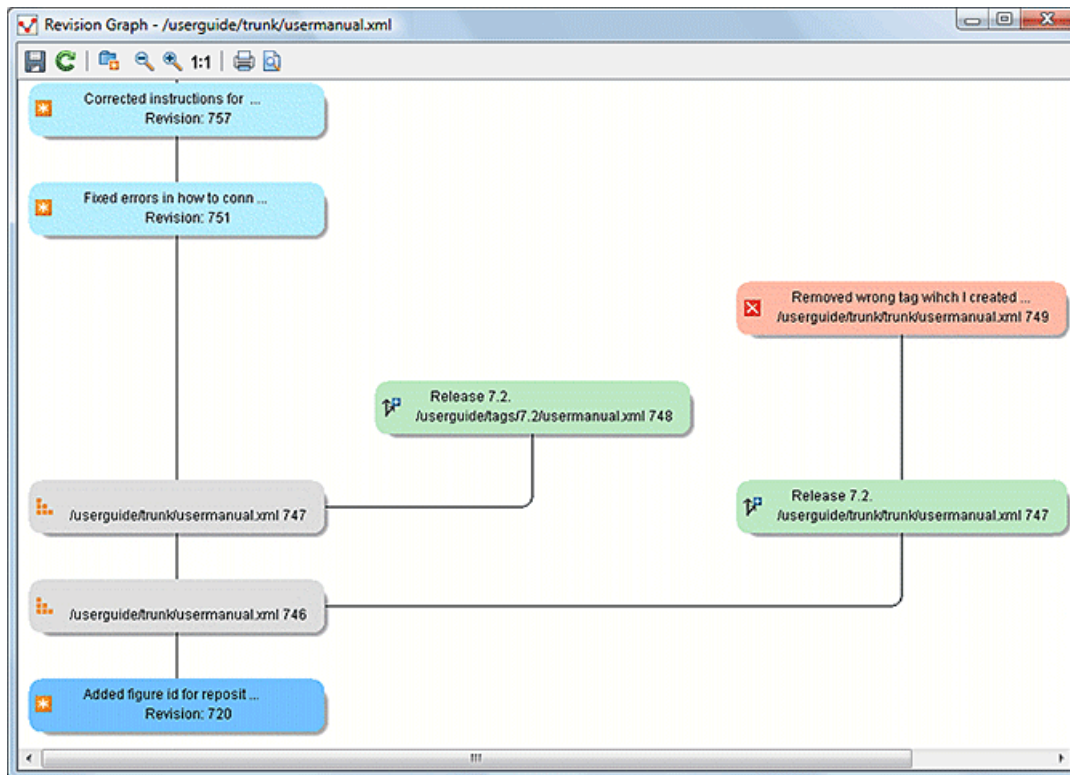

Figure 249: The Revision Graph of a File Resource

In every node of the revision graph an icon and the background color represent the type of operation that created the revision represented in that node. Also the commit message associated with that revision, the repository path and the revision number are contained in the node. The tooltip displayed when the mouse pointer hovers over a node specifies the URL of the resource, the SVN user who created the revision of that node, the revision number, the date of creation, the commit message, the modification type and *the affected paths*.

The types of nodes used in the graph are:

Added resource

The icon for a new resource added to the repository ( ) and green background;

Copied resource

The icon for a resource copied to other location, for example when a SVN tag is created ( ) and green background;

Modified resource

The icon for a modified resource ( ) and blue background;

Deleted resource

The icon for a resource deleted from the repository ( ) and red background;

Replaced resource

The icon for a resource removed and replaced with another one on the repository ( ) and orange background;

Indirect resource

The icon for a revision from where the resource was copied or an indirectly modified resource, that is a directory in which a resource was modified ( ) and grey background; the *Modification type* field of the tooltip specifies how that revision was obtained in the history of the resource.

A directory resource is represented with two types of graphs:

simplified graph

   Lists only the changes applied directly to the directory;

complete graph

   Lists also the indirect changes of the directory resource, that is the changes applied to the resources contained in
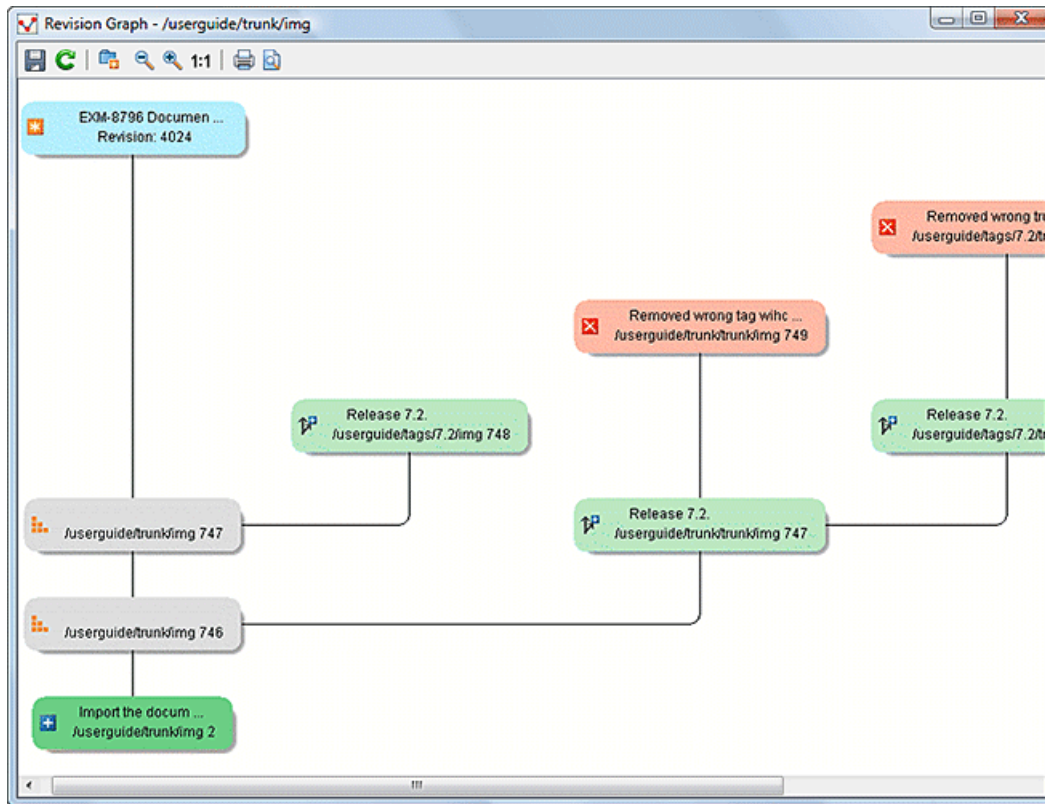   the directory.

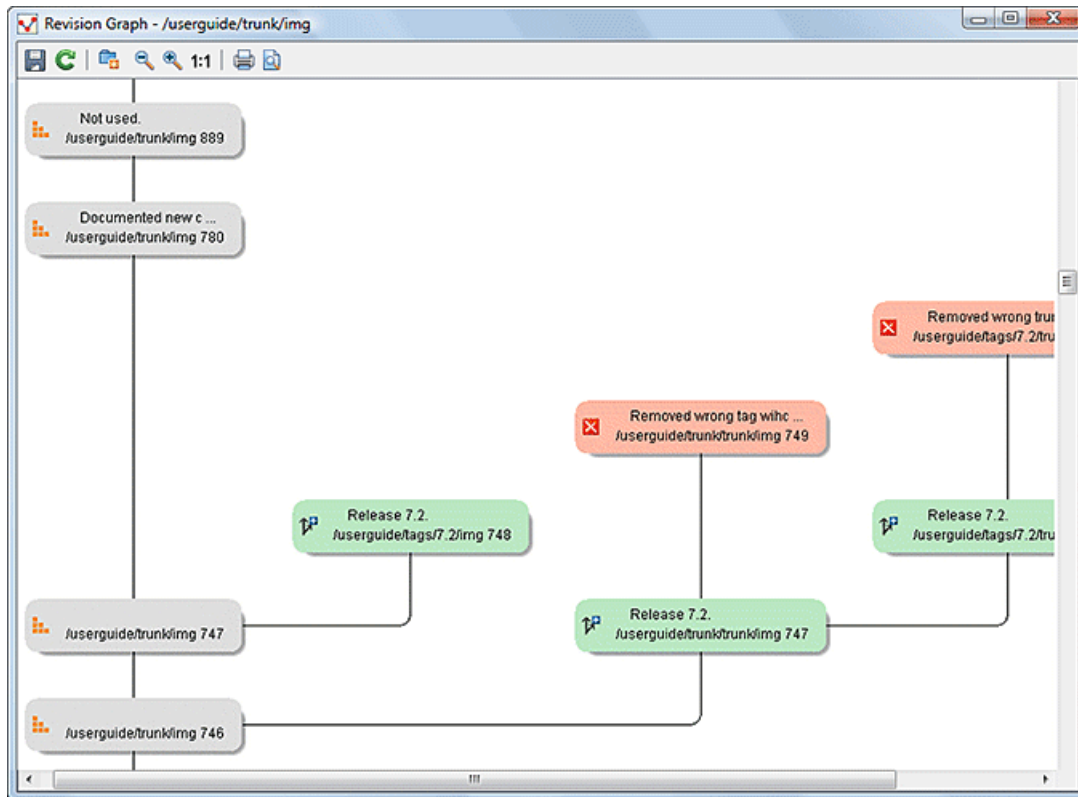**Figure 250: The Revision Graph of a Directory (Direct Changes)**

Figure 251: The Revision Graph of a Directory (Also Indirect Changes)

The **Revision graph** dialog toolbar contains the following actions:

Save as image

Saves the graphical representation as image. For a large revision graph you have to *set more memory in the startup script*. The default memory size is not enough when there are more than 100 revisions that are included in the graph.

Show/Hide indirect modifications

Switches between simplified and complete graph.

Zoom In

Zooms in the graph.

Zoom Out

Zooms out the graph. When the font reaches its minimum size, the graph nodes will display only the icons, leading to a very compact representation of the graph.

1:1 Reset scale

Resets the graphical scale to a default setting.

Print

Prints the graph.

Print preview

Offers a preview of the graph to allow you to check the information to be printed.

Right clicking any of the graph nodes display a contextual menu containing the following actions:

Open

Opens the selected revision in the editor panel. Available only for files.

Open with

Opens the selected revision in the editor panel. Available only for files.

Save as

Saves the file for which the revision graph was generated, based on the selected node revision.

Copy to

Copies to the repository the item whose revision graph is displayed, using the selected revision.

> **Note:** This action can be used to resurrect deleted items also.

Compare with HEAD

Compares the selected revision with the HEAD revision and displays the result in the diff panel. Available only for files.

🕐 **Show History**

Displays the history of the resource in *the History view*. Available for both files and directories.

📥 **Check out**

Checks out the selected revision of the directory. Available only for directories.

When two nodes are selected in the revision graph of a file the right click menu of this selection contains only the **Compare** for comparing the two revisions corresponding to the selected nodes. If the resource for which the revision graph was built is a folder then the right click menu displayed for a two nodes selection also contains the **Compare** action but it computes the differences between the two selected revisions as a set of directory changes. The result is displayed in the *Directory Change Set* view.

> ⚠ **Attention:**
>
> Generating the revision graph of a resource with many revisions may be a slow operation. You should enable caching for revision graph actions so that future actions on the same repository will not request the same data again from the SVN server which will finish the operation much faster.

Oxygen XML Author Preferences

The options used in the SVN client are saved and loaded independently from the Oxygen XML Author options. However if at the Oxygen XML Author's first startup it cannot be determined a set of SVN options to be loaded, some of the preferences are imported from the XML Editor options (e.g. License key and HTTP Proxy settings).

The preferences dialog can be accessed from the *Options* -> Preferences. The preferences panels are called *Global* , *SVN* , *Diff colors* and HTTP/Proxy Configuration.

There is a second set of preferences applied to the SVN client: the preferences set in the global SVN files called config and servers. These are the files holding parameters that act as defaults applied to all the SVN client tools that are used by the same user on his/hers login account on the computer. These files can be opened for editing with the two edit actions available in the SVN client tool on the **Global Runtime Configuration** submenu of the **Options** menu.

Technical Issues

This section contains special technical issues found during the use of Syncro SVN Client.

Authentication Certificates Not Saved

If Syncro SVN Client prompts you to enter the authentication certificate, although you already provided it in a previous session, then you should make sure that your local machine user account has the necessary rights to store certificate files in the *Subversion* configuration folder (write access to *Subversion* folder and all its subfolders). Usually, it is located in the following locations:

- Windows: `[HOME_DIR]\AppData\Roaming\Subversion`
- Mac OS X and Linux: `[HOME_DIR]/.subversion`

Updating Newly Added Resources

When you want to get from the repository a resource which is part of a newly created structure of folders, you need to also get its parent folders.
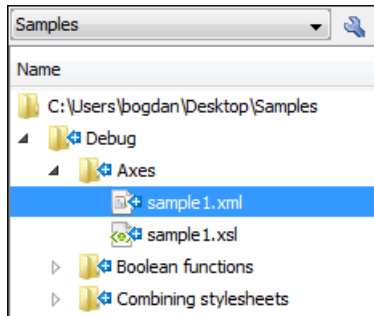

Figure 252: An incoming structure of folders from the repository

Syncro SVN Client allows you to choose how you want to deal with the entire structure from that moment onwards:

Update ancestor directories recursively

This option brings the entire newly added folders structure into your working copy. In this case, the update time depends on the total number of newly incoming resources, because of the full update operation (not updating only selected resource).

Update selected files only (leave ancestor directories empty)

This option brings a skeleton structure composed of the resource's parent folders only, and the selected resource at the end of the operation. All of the parent directories will have depth set to *empty* in your working copy, thus subsequent **Synchronize** operations will not report any remote modifications in those folders. If you need to update the folders to full-depth, you can use **Update to revision/depth** option from the working copy.

Cannot Access a Repository through HTTPS

If you have issues when trying to access a repository through HTTPS protocol, one of the possible causes can be the encryption protocol currently used by the application. This is happening when:

• you are running Oxygen XML Author with Java 1.6 or older;
• the repository is set to use only one of the SSLv3 or TLSv1 encryption protocols.

To solve this issue, set the *HTTPS encryption protocols* option to **SSLv3 only** or **TLSv1 only** (depending on the repository configuration).

Accessing Old Items from a Repository

Usually, you point to an item from a repository using an URL. However, sometimes this might not be enough, because the URL alone might point to a different item than the one you want and a *peg revision* is needed.

A Subversion repository tracks any change made to its items by using *revisions*, which contain information like the name of the author who made the changes, the date when they were made, and a number that uniquely identifies each of them. During time, an item from a specific location in a repository evolves as a result of committing different changes to it. When an item is deleted, its entire life cycle (all changes made to it, from the moment it was created) still remains recorded in the history of the repository. If a new item is created, with the same name and in the same location of the repository as a previously existing one, then both items are identified by the same URL even though they are located in different time frames. This is when a *peg revision* comes in handy. A *peg revision* is nothing more than a normal revision, but the difference between them is made by their usage. Many of the Subversion commands accept also a peg revision as a way to precisely identify an item in time, beside an *operative revision* (the revision we are interested in, regarding the used command).

Let's assume that:

- we created a new repository file `config`, identified by the URL
  `http://host.com/myRepository/dir/config`;
- the file has been created at revision `10`;
- during time, the file was modified by committing revisions `12`, `15`, `17`;

To access a specific version of the file identified by the
`http://host.com/myRepository/dir/config` URL, we need to use a corresponding
revision (the operative revision):

- if we use a revision number less than `10`, an error is triggered, because the file has not been created
  yet;
- if we use a revision number between `10` and `19`, we will obtain the specific version we are interested
  in;

  > **Note:** Although the file was modified in revisions `12`, `15`, `17`, it existed in
  > all revisions between `10` and `19`. Starting with a revision at which the file is
  > modified, it has the same content across all revisions generated in the repository
  > until another revision in which it is modified again.

At this point, we delete the file, creating revision `20`. Now, we cannot access any version of the file,
because it does not exist anymore in the latest repository revision. This is due to the fact that Subversion
automatically uses the `HEAD` revision as a peg revision (it assumes any item currently exists in the
repository if not instructed otherwise). However, using any of the revision numbers from the `10-19`
interval (when the file existed) as a peg revision (beside the operative revision), will help Subversion
to properly identify the time frame when the file existed, and access the file version corresponding to
the operative revision. If we use a revision number greater than `19`, this will also trigger an error.

Continuing our example, let's suppose that at revision `30` we create a directory, incidentally called
`config`, in the same repository location as our deleted file. This means that our new directory will
be identified by the same repository address: `http://host.com/myRepository/dir/config`.
If we use only this URL in any Subversion command, we will access the new directory. We will also
access the same directory if we use as peg revision any revision number equal with or greater than
`30`. But, if we are interested in accessing an old version of the previously existing file, then we must
use as a peg revision one of the revisions at which it existed (`10-19`), just like in the previous case.

Comparing and Merging Documents

In large teams composed either of developers or technical authors, the usage of a shared repository for the source or
document files is a must. Often many authors are editing the same file at the same time.

Finding what has been modified in your files and folders can be hard. If your data is changing, you can benefit from
accurate identification and processing of changes in your files and folders with Oxygen XML Author's features for
comparing files and directories. These are powerful and easy to use tools that will do the job fast and thoroughly. With
the new possibilities of differencing and merging, it is now easy to manage multiple changes.

Oxygen XML Author provides a simple means of performing file and folder comparisons. You can see the differences
in your files and folders and merge the changes.

There are two levels on which the comparison can be done, namely comparing directories or comparing individual files.
These two operations are available from the **Tools** menu.

> **Note:** Oxygen XML Author uses diff directories to compare archive files.

The comparison tool can also be started using command line arguments. In the installation folder there are two executable
shells (`diffFiles.bat` and `diffDirs.bat` on Windows, `diffFiles.sh` and `diffDirs.sh` on Unix/Linux,
`diffFilesMac.sh` and `diffDirsMac.sh` on OS X). You can pass one or two command-line arguments to each
of these shells. Both arguments can point to directories or archives (supported formats: *zip*, *docx*, and *xlsx*).

For example, to start the comparison between two Windows directories, use the following command:

```
diffDirs.bat "c:\Program Files" "c:\ant"
```

If there are spaces in the path names, surround the paths with quotes. If you pass only one argument, you are prompted to manually choose the second directory or archive. This is valid for the files diff utility as well.

Directories Comparison

The directories comparison result is presented as a tree of files and directories. The directories that contain different files are expanded automatically, so you can focus directly on the differences. You can merge the directories contents using the copy actions. A double click or an Enter key on a line with a pair of files starts *comparing the file content* of the two files from that line in the **Compare Files** window. Please note that the content is compared only in case of known file types, that is the files associated with the built-in editors and the file types associated with a built-in editor when the user was prompted to specify such an association (when opening for the first time a file of an unknown type).
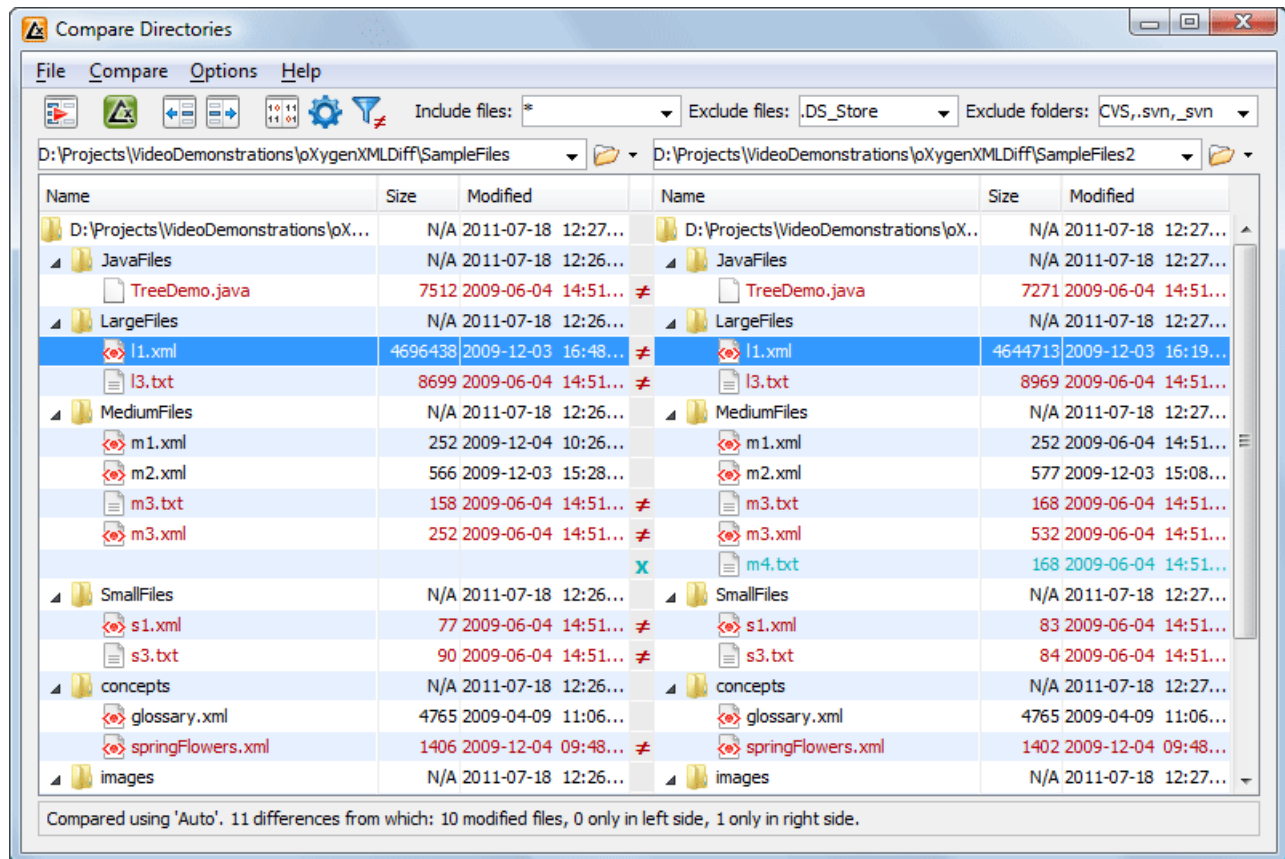

Figure 253: The Compare Directories Window

Directories Comparison User Interface

This section explains the user interface of the **Directories Comparison** window.

Compare Menu

This menu contains the following action:

- **Perform Directories Differencing** - Looks for differences between the two directories displayed in the left and right side of the application window.
- **Perform Files Differencing** - Compares the currently selected files.
- **Copy Change from Right to Left** - Copies the selected change from the right side to the left side (if there is no file/folder in the right side, the left file/folder is deleted).

- 🔜 **Copy Change from Left to Right** - Copies the selected change from the left side to the right side (if there is no file/folder in the left side, the right file/folder is deleted).

Compare Toolbar

The toolbar contains the following actions:

Figure 254: The Compare toolbar

Perform directories differencing

Looks for differences between the two directories displayed in the left and right side of the application window.

Perform files differencing

Compares the currently selected files.

Copy Change from Right to Left

Copies the selected change from the right side to the left side (if there is no file/folder in the right side, the left file/folder is deleted).

Copy Change from Left to Right

Copies the selected change from the left side to the right side (if there is no file/folder in the left side, the right file/folder is deleted).

Binary Compare

Performs a byte-level comparison on the selected files.

Diff Options

Opens the Directories Comparison preferences page.

Show Only Modifications

Displays a more uncluttered file structure by hiding all identical files.

Files and folders filters

Differences can be filtered using three filter boxes: **Include files, Exclude files, Exclude folders**. They come with predefined values and are editable to allow more custom values. All of them accept multiple comma separated values and the ***** and **?** wildcards. For example, to filter out all jpeg and gif image files, edit the **Exclude files** filter box to read ***.jpeg, *.png**. Each filter keeps a list with the latest 15 filters applied in the drop-down list of the filter box.

Directories Selector

To open the directories you want to compare, select a folder from each **Browse for local directory** button. The Diff Tool keeps track of the folders you are currently working with and those you opened in this window. You can see and select them from the two combo-boxes.

If you want to compare two archives' content you can select the archives from the **Browse for archive file** button.

> ℹ️ **Tip:** By default, the supported archives are not treated as directories and the comparison is not performed over the files inside them. To make Oxygen XML Author treat supported archives as directories, go to the *Diff preferences page* and enable the **Look in archives** option.

Comparison Result

The directory comparison result is presented using two tree-like structures, showing files and folders names, size and modification date.

Figure 255: Comparison result

A column holding graphic symbols separate the two tree-like structures. The graphic symbols can be:

- an "X" sign, when a file or a folder exists only in one of the compared directories;
- a "not-equal" sign, if a file exists in both directories, but the content is different. The same sign appears when a collapsed folder contains modified files.

The color used for painting the sign and the directory or file name can be customized in the *Directories Comparison / Appearance* preferences page. You can double-click the line marked with the "not-equal" sign to open a new **File Content Comparison** window, showing the differences between the two files.

Compare Images

When double-clicking a line containing two different images a compare images dialog is displayed. The dialog presents the images in the left and right part scaled to fit the view available area. You can use the contextual menu actions to scale the images at their original size or scale them down to fit in the view area.

The supported image types are: GIF, JPG / JPEG, PNG, BMP.

Files Comparison

To compare two files side by side, open the **Diff Files** dialog from **Tools** > **Compare Files**. Using the 🗁 ▾ **Browse for local file** button open a file in the left side of the dialog, and the file you want to compare it to in the right side. To highlight the differences between the two files, click the 🔳 **Perform File Differencing** button. The line numbers on each side help you to identify the locations of the differences quickly.

You can edit both the source and the target file. The differences are refreshed when you save the modified document.
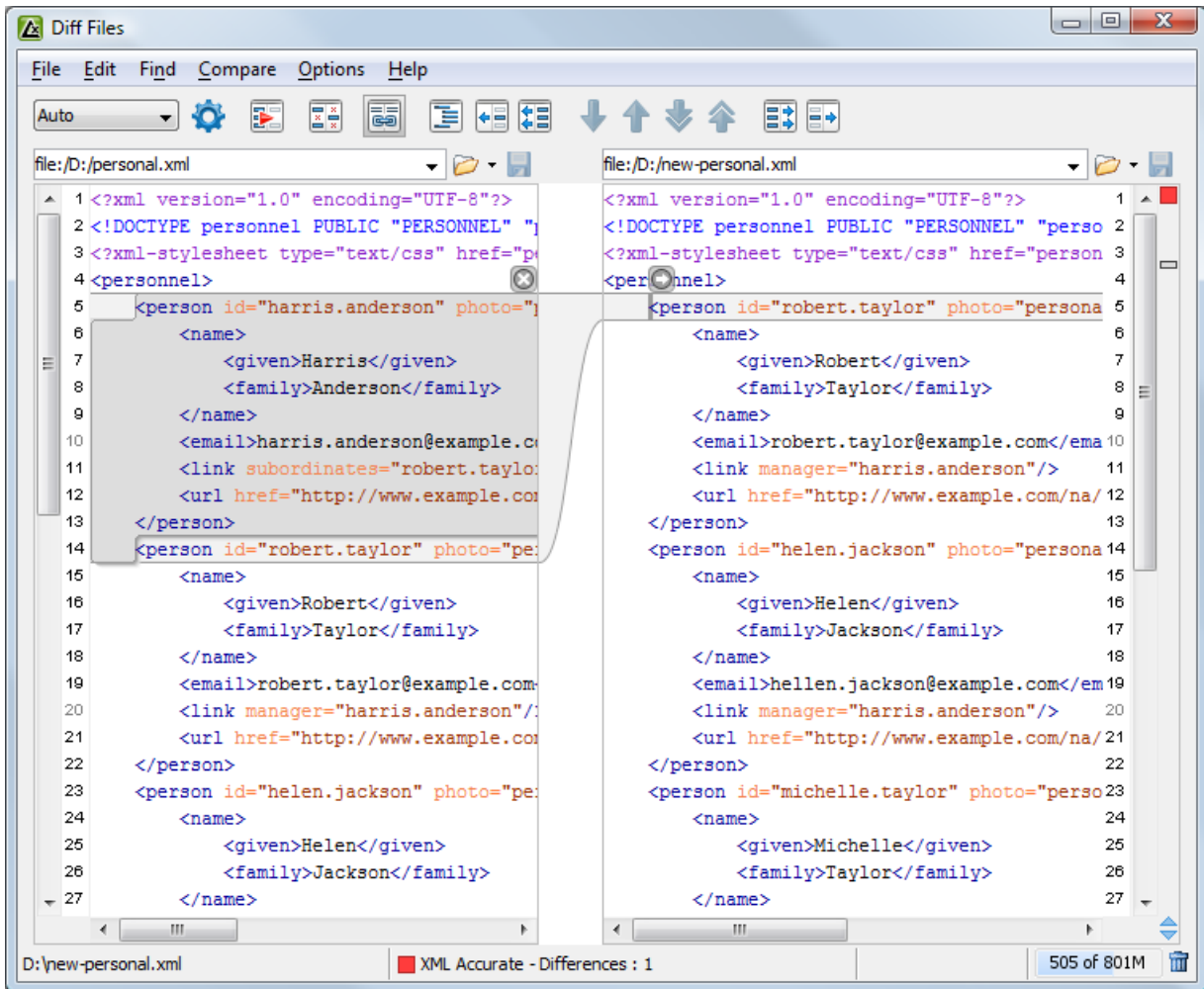
Figure 256: The Compare Files Window

Adjacent changes are grouped into blocks of changes. This layout allows an easier visual identification and focuses on a group of related changes.

A widget containing actions that can be used to copy or append changes from either of the two sides is displayed when you select a change:

⊗ **Append left change to right** and ⊗ **Append right change to left**

Copy the content of the selected change from one side and appends it after the content of the corresponding change from the other side; as a result, the side towards the arrows point to, will contain the changes from both sides.

⊙ **Copy change from left to right** and ⊙ **Copy change from right to left**

Replace the content of a change from one side with the content of the corresponding change from the other side.

Oxygen XML Author offers six different diff algorithms to choose from for file comparison:

* two XML diff algorithms: **XML Fast** and **XML Accurate**,
* a **Syntax Aware** algorithm that gives very good results on all file types known by Oxygen XML Author,
* three all-purpose algorithms: line based, word based and character based.

You can use any of these six algorithms to perform differences on request, but Oxygen XML Author also offers an automatic mode that selects the most appropriate one, based on the file content and size.

Main Menu

This section explains the menu actions of the **Files Comparison** window.

File Menu

The following actions are available:

Source

The file is displayed in the left side of the application window.

- **Source** > 📂 **Open** - Browses for a source file.
- **Source** > 📂 **Open URL** - Opens URL to be used as a source file. See *Open URL* for details.
- **Source** > 📂 **Open File from Archive** - Browses an archive content for a source file.
- **Source** > 💾 **Save** - Saves the changes made in the source file.
- **Source** > **Save As...** - Displays the **Save As** dialog that allows you to save the source file with a new name.

Target

The file is displayed in the right side of the application window.

- **Target** > 📂 **Open** - Browses for a target file.
- **Target** > 📂 **Open URL** - Opens URL to be used as a target file. See *Open URL* for details.
- **Target** > 📂 **Open File from Archive** - Browses an archive content for a target file.
- **Target** > 💾 **Save** - Saves the changes made in the target file.
- **Target** > **Save As...** - Displays the **Save As** dialog that allows you to save the target file with a new name.

Exit

Quits the application.

Edit Menu

The following actions are available:

✂ **Cut**

Cut selection to clipboard from the local file currently opened in the focused **Compare** editor.

📋 **Copy**

Copy selection to clipboard from the local file currently opened in the focused **Compare** editor.

📋 **Paste**

Paste selection from clipboard in the local file currently opened in the focused **Compare** editor.

↩ **Undo**

Undo edit changes in the local file currently opened in the focused **Compare** editor.

↪ **Redo**

Redo edit changes in the local file currently opened in the focused **Compare** editor.

Find Menu

The find actions are the following:

🔍 **Find/Replace**

Perform *find/replace* operations in the file currently opened in the focused **Editor**.

🔍 **Find Next**

Go to the next match using the same options of the last *find* operation. The action runs in the two editor panels.

🔍 **Find Previous**

Go to the previous match using the same options of the last *find* operation. The action runs in the two editor panels.

Compare Menu

The following actions are available in this menu:

Perform Files Differencing

Performs a comparison between the source and target files.

Next Block of Changes

Jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes in the document;

> **Note:** A change block groups one or more consecutive lines that contain at least one change.

Previous Block of Changes

Jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes in the document;

Next Change

Jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change.

Previous Change

Jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change.

Last Change

Jumps to the last change from the current file.

First Change

Jumps to the first change from the current file.

Copy All Changes from Left to Right

Copies all changes from source to target file.

Copy All Changes from Right to Left

Copies all changes from target to source file.

Copy Change from Left to Right

Copies the selected difference from source to target file.

Copy Change from Right to Left

Copies the selected difference from target to source file.

Show Word Level Details

Provides a word-level comparison of the selected change.

Show Character Level Details

Provides a character-level comparison of the selected change.

Options Menu

- **Preferences** - Opens the preferences pages.

- **Menu Shortcut Keys** - Opens the **Menu Shortcut Keys** option page. Here you can configure all keyboard shortcuts available for menu items.

- **Reset Global Options** - Resets options to their default values.

- **Import Global Options** - Allows you to import an options set you have previously exported.

- **Export Global Options** - Allows you to export the current options set to a file.

Help Menu

The **Help** menu contains the following actions:

Help

Opens the **Help** dialog.

Report problem

Opens a dialog that allows the user to write the description of a problem that was encountered while using the application. You are able to change the URL where the reported problem is sent using the *com.oxygenxml.report.problems.url* system property. The report is sent in XML format through the `report` parameter of the POST HTTP method.

Improvement Program Options

Allows you to activate or deactivate the application Product Improvement Program.

Support center

Opens the Oxygen XML Author Support Center web page in a browser.

Compare Toolbar

This toolbar contains the operations that can be performed on the source and target files.

Figure 257: The Compare Toolbar

The following actions are available:

Algorithm

This option box allows you to select one of the 6 available compare algorithms:

- *Characters* Algorithm computes the differences at character level.

- *Words* Algorithm computes the differences at word level.

- *Lines* algorithm computes the differences at line level, meaning that it compares two files looking for the first identical line of text. When it is found, it is considered a match. The content that precedes the match is considered to be a difference and marked accordingly. Then the algorithm continues to look for matching lines marking the content between them as differences.

- *Syntax Aware* for known file types, like the ones listed in the **New** dialog box, for example the XML file type (which includes, among others, XSLT files, XSL-FO files, XSD files, RNG files, NVDL files), the XQuery file type (`.xquery`, `.xq`, `.xqy`, `.xqm` extensions), the DTD file type (`.dtd`, `.ent`, `.mod` extensions), the TEXT file type (`.txt` extension), the PHP file type (`.php` extension), etc.

  This algorithm splits the files into sequences of *tokens* and computes the differences between them. A *token* can have a different meaning, depending on the type of the compared files. For example:

  - when comparing XML files, a token can be one of the following:

    - the name of an XML tag
    - the '<' character
    - the '/>' sequence of characters
    - the name of an attribute inside an XML tag
    - the '=' sign
    - the '"' character
    - an attribute value
    - the text string between the start tag and the end tag (that is a text node which is a child of the XML element corresponding to the XML tag that encloses the text string)

- when comparing plain text files (identified by the .txt extension), a token can be any continuous sequence of word characters (letters, digit, the '_' character) or any continuous sequence of whitespace characters including a newline character.

- *XML Fast* works on larger files but it is less precise than XML Accurate.

- *XML Accurate* works best on small XML files.

- *Auto* selects the most appropriate algorithm, based on the files content and size. By default, the **Auto** mode is selected.

Diff Options

Opens the *Files Comparison page*.

Perform Files Differencing

Performs a comparison between the source and target files.

Ignore Whitespaces

Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.

Synchronized scrolling

Synchronizes scrolling of the two open files, so that a selected difference can be seen on both sides of the application window. This action enables/disables the previous described behavior.

Format and Indent Both Files

Formats and indents both files before you compare them. Use this option when you compare files that contain long lines which makes it difficult to spot differences.

Copy Change from Right to Left

Copies the selected difference from target to source file.

Copy All Changes from Right to Left

Copies all changes from target to source file.

Next Block of Changes

Jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes in the document;

> **Note:** A change block groups one or more consecutive lines that contain at least one change.

Previous Block of Changes

Jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes in the document;

Next Change

Jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change.

Previous Change

Jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change.

Copy All Changes from Left to Right

Copies all changes from source to target file.

Copy Change from Left to Right

Copies the selected difference from source to target file.

First Change

Jumps to the first change from the current file.

Files Selector

To open the source and target files where you want to see the differences, use one of the **Browse for local**, **Browse for remote** or **Browse for archive** buttons. The Diff Tool keeps track of the files you are currently working with and those you opened in this window. You can see and select them from the two combo-boxes.

You can also save the changes in the source file or the target file by clicking the corresponding "Save" button.

File Contents Panel

The files are opened in two side-by-side editors. The text view is used, offering a better view of the differences.

The two editors are kept in sync, so if you scroll the text in one of them, the other one also scrolls to show the difference side-by-side. The differences are indicated using highlights connected through colored areas. To navigate between differences, do one of the following:

- press the **Go to modification** buttons;
- select a change by clicking it in the overview ruler located in the right-most part of the window. The overview ruler contains a success indicator in its upper part. It turns green in case there are no differences or red if some differences are found.
- click a colored area between the text editors.

You can edit either the source or the target file. The differences are refreshed when you save the modified document.

Both editors provide a contextual menu that contains *edit*, *merge* and *navigation* actions.

The **Find/Replace** dialog is displayed by pressing **Ctrl+F (Command+F on OS X)**. **Find/Replace** options are also available:

- **F3** used for performing another search using the last search configuration.
- **Shift+F3** used for performing another search in backward direction using the last search configuration.

If the compared blocks of text are too large and you want to see the differences at a finer level, you can use the comparison at **Word** or **Character** level.

Word Level Comparison

This option is only available if differences exist between the source and the target file. You can go to Word Level Comparison by clicking the **Show word level details** menu item from the **Compare** menu.
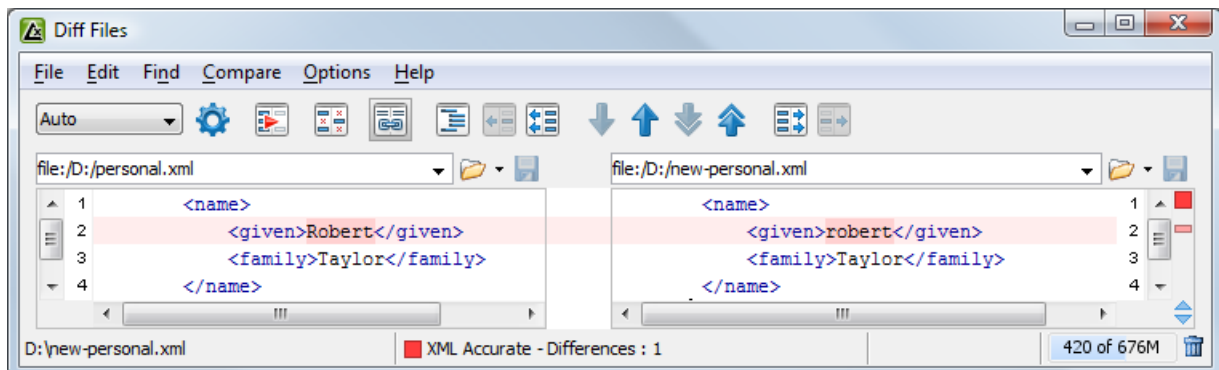

Figure 258: Word Level Comparison

Character Level Comparison

This option is only available if modifications exist between the source and the target file. You can go to Character Level Comparison by clicking the **Show Character Level details** menu item from the **Compare** menu.
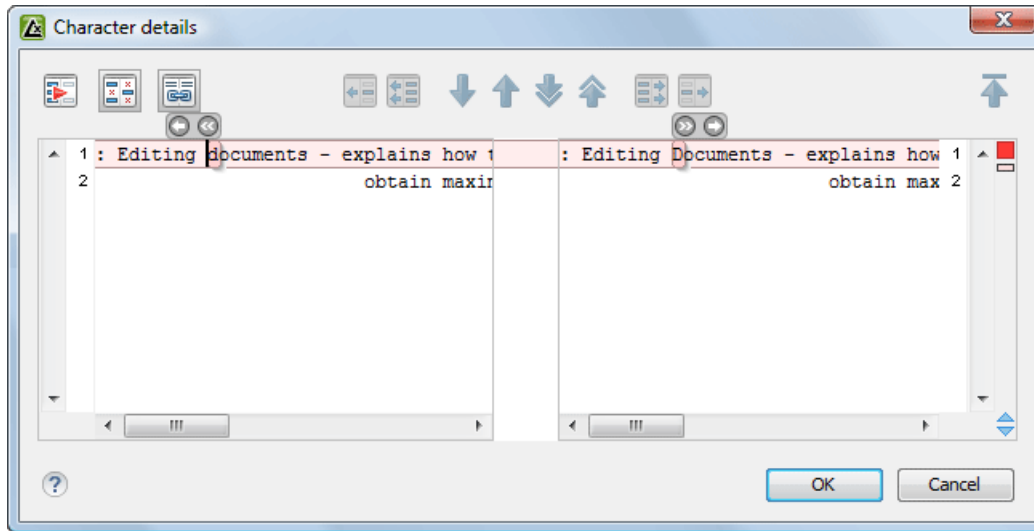

Figure 259: Character Level Comparison

XML Digital Signatures

This chapter explains how to apply and verify digital signatures on XML documents.

Overview

Digital signatures are widely used as security tokens, not just in XML. A digital signature provides a mechanism for assuring integrity of data, the authentication of its signer, and the non-repudiation of the entire signature to an external party:

- A digital signature must provide a way to verify that the data has not been modified or replaced to ensure integrity.
- The signature must provide a way to establish the identity of the data's signer for authentication.
- The signature must provide the ability for the data's integrity and authentication to be provable to a third party for non-repudiation.

A public key system is used to create the digital signature and it's also used for verification. The signature binds the signer to the document because digitally signing a document requires the originator to create a hash of the message and then encrypt that hash value with his own private key. Only the originator has that private key and he is the only one that can encrypt the hash so that it can be unencrypted using his public key. The recipient, upon receiving both the message and the encrypted hash value, can decrypt the hash value, knowing the originator's public key. The recipient must also try to generate the hash value of the message and compare the newly generated hash value with the unencrypted hash value received from the originator. If the hash values are identical, it proves that the originator created the message, because only the actual originator could encrypt the hash value correctly.

XML Signatures can be applied to any digital content (data object), including XML (see W3C Recommendation, *XML-Signature Syntax and Processing*). An XML Signature may be applied to the content of one or more resources:

- enveloped or enveloping signatures are applied over data within the same XML document as the signature
- detached signatures are applied over data external to the signature element; the signature is "detached" from the content it signs. This definition typically applies to separate data objects, but it also includes the instance where the signature and data object reside within the same XML document but are sibling elements.

The XML Signature is a method of associating a key with referenced data. It does not normatively specify how keys are associated with persons or institutions, nor the meaning of the data being referenced and signed.

The original data is not actually signed. Instead, the signature is applied to the output of a chain of canonicalization and transformation algorithms, which are applied to the data in a designated sequence. This system provides the flexibility to accommodate whatever "normalization" or desired preprocessing of the data that might be required or desired before subjecting it to being signed.

To canonicalize something means to put it in a standard format that everyone generally uses. Because the signature is dependent on the content it is signing, a signature produced from a not canonicalized document could possibly be different from one produced from a canonicalized document. The canonical form of an XML document is physical representation of the document produced by the method described in this specification. The term canonical XML refers to XML that is in canonical form. The XML canonicalization method is the algorithm defined by this specification that generates the canonical form of a given XML document or document subset. The term XML canonicalization refers to the process of applying the XML canonicalization method to an XML document or document subset. XML canonicalization is designed to be useful to applications that require the ability to test whether the information content of a document or document subset has been changed. This is done by comparing the canonical form of the original document before application processing with the canonical form of the document result of the application processing.

A digital signature over the canonical form of an XML document or document subset would allows the signature digest calculations to be oblivious to changes in the original document's physical representation. During signature generation, the digest is computed over the canonical form of the document. The document is then transferred to the relying party, which validates the signature by reading the document and computing a digest of the canonical form of the received document. The equivalence of the digests computed by the signing and relying parties (and hence the equivalence of the canonical forms over which they were computed) ensures that the information content of the document has not been altered since it was signed.

The following canonicalization algorithms are used in Oxygen XML Author: Canonical XML (or Inclusive XML Canonicalization)(*XMLC14N*) and Exclusive XML Canonicalization(*EXCC14N*). The first is used for XML where the context doesn't change while the second was designed for canonicalization where the context might change.

Inclusive Canonicalization copies all the declarations, even if they are defined outside of the scope of the signature. In this way all the declarations you might use will be unambiguously specified. A problem appears when the signed XML is moved into another XML document which has other declarations because the Inclusive Canonicalization will copy then and the signature will be invalid.

Exclusive Canonicalization finds out what namespaces you are actually using (the ones that are a part of the XML syntax) and just copies those. It does not look into attribute values or element content, so the namespace declarations required to process these are not copied.

This type of canonicalization is useful when you have a signed XML document that you wish to insert into other XML documents and it will insure the signature verifies correctly every time, so it is required when you need self-signed structures that support placement within different XML contexts.

Inclusive Canonicalization is useful when it is less likely that the signed data will be inserted in other XML document and it's the safer method from the security perspective because it requires no knowledge of the data that are to be secured in order to safely sign them.

The canonicalization method can specify whether or not comments should be included in the canonical form output by the XML canonicalization method. If a canonical form contains comments corresponding to the comment nodes in the input node-set, the result is called canonical XML with comments. In an uncommented canonical form comments are removed, including delimiter for comments outside document element.

These three operations: Digital Signing, Canonicalization and Verification of the signature are available from the **Tools** menu or from the Editor's **contextual menu** > **Source**.

Canonicalizing Files

The user can select the canonicalization algorithm to be used for his document from the following dialog displayed by the action **Canonicalize** available from the editor panel's **contextual menu** > **Source** and also from menu **Tools**.
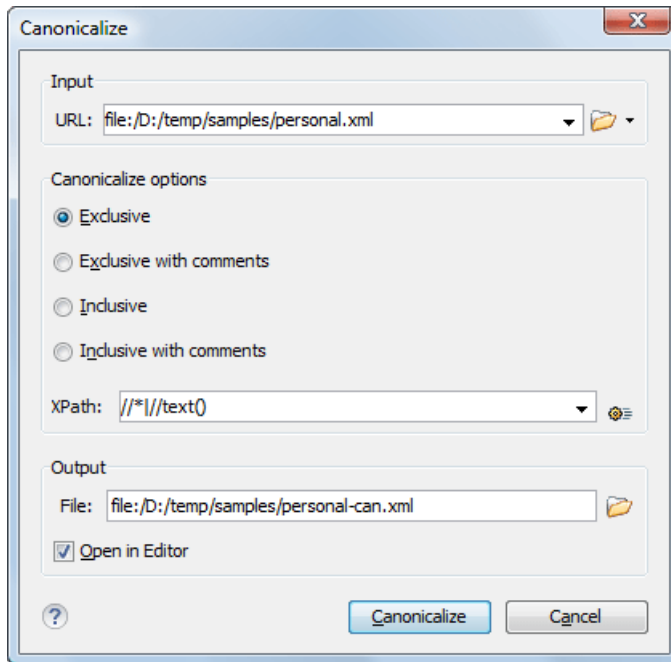
Figure 260: Canonicalization settings dialog

You can set the following:

- **URL** - Specifies the location of the input URL.
- **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.
- **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
- **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.
- **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.
- **XPath** - The XPath expression provides the fragments of the XML document to be signed.
- **Output** - Specifies the output file path where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in the editor.

Certificates

A certificate is a digitally signed statement from the issuer (an individual, an organization, a website or a firm), saying that the public key (and some other information) of some other entity has a particular value. When data is digitally signed, the signature can be verified to check the data integrity and authenticity. Integrity means that the data has not been modified. Authenticity means the data comes indeed from the entity that claims to have created and signed it. Certificates are kept in special repositories called keystores.

A *keystore* is an encrypted file that contains private keys and certificates. All keystore entries (key and trusted certificate entries) are accessed via unique aliases. An alias must be assigned for every new entry of either a key or certificate as a reference for that entity. No keystore can store an entity if 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 Author 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, *open the **Preferences** dialog* and go to **Certificates**.

Signing Files

The user can select the type of signature to be used for his document from the following dialog displayed by the action **Sign** available from the editor panel's **contextual menu** > **Source** and also from menu **Tools** .
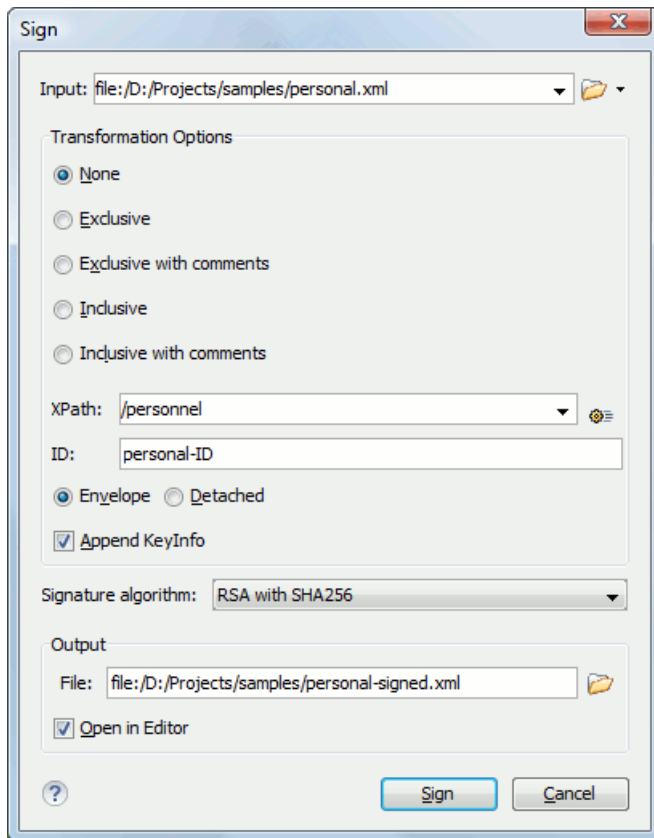
Figure 261: Signature settings dialog

The following options are available:

- **Input** - Specifies the location of the input URL.
- **None** - If selected, no canonicalization algorithm is used.
- **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.
- **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
- **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.
- **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.
- **XPath** - The XPath expression provides the fragments of the XML document to be signed.
- **ID** - Provides ID of the XML element to be signed.
- **Envelope** - If selected, the enveloping signature is used.
- **Detached** - If selected, the detached signature is used.
- **Append KeyInfo** - The element `ds:KeyInfo` will be added in the signed document only if this option is checked.
- **Signature algorithm** - Algorithm used for signing the document. The following options are available: **RSA with SHA1**, **RSA with SHA256**, **RSA with SHA384**, and **RSA with SHA512**.
- **Output** - Specifies the output file path where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in the editor.

Verifying the Signature

You can verify the signature of a file from **Tools** > **Verify Signature** or from the contextual menu of the file: **Source** > **Verify Signature**. The **URL** field in the **Verify Signature** dialog specifies the location of the file whose signature is verified.

In case the signature is valid, a dialog displaying the name of the signer is displayed. If not, an error shows details about the problem.

Large File Viewer

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

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

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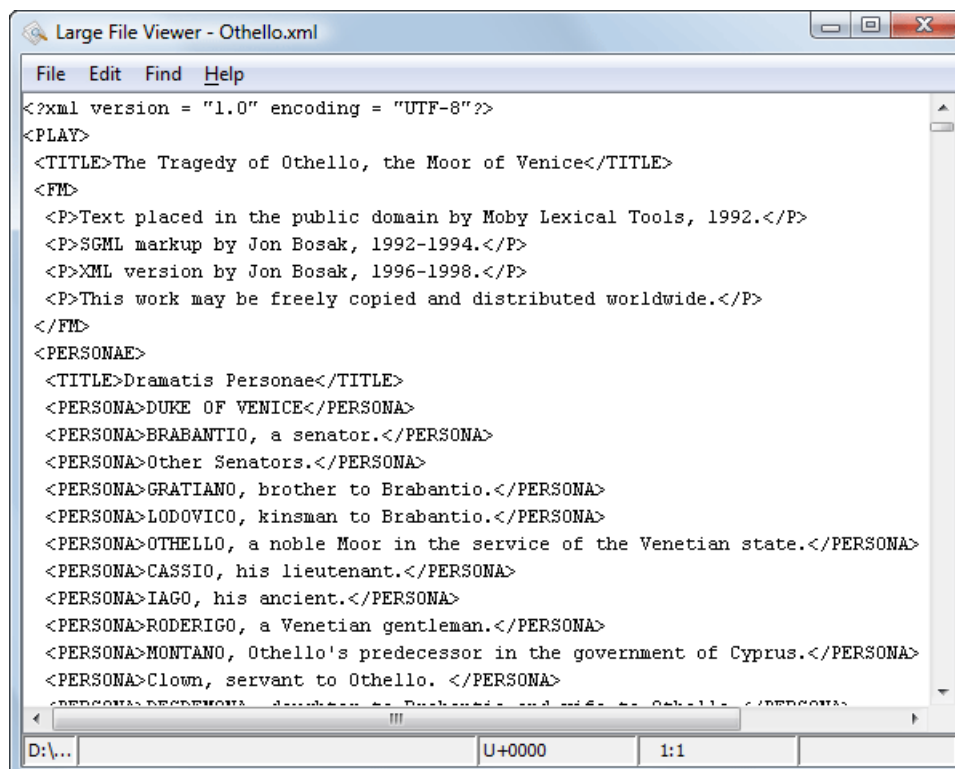

Figure 262: The Large File Viewer

Large File Viewer components:

• The menu bar provides menu driven access to all the features and functions available in **Large File Viewer**:

File > Open

Opens files in the viewer (also available in the contextual pop-up menu).

File > Exit

Closes the viewer.

Edit > Copy

Copies the selected text to clipboard (also available in the contextual pop-up menu).

Find > Find

opens a reduced **Find** dialog providing some basic search options like:

• **Case sensitive** - When checked, operations are case-sensitive.
• **Regular Expression** - When checked, allows using any regular expression in *Perl-like syntax*.
• **Wrap around** - Continues the find from the start (end) of the document after reaching the end (start) if the search is in forward (backward) direction.

Help > Help

Provides access to the User Manual.

• The status bar provides information about the current opened file path, the Unicode representation of the character at caret position and the line and column in the opened document where the caret is located.

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

ℹ **Tip:** The best performance of the viewer is accomplished for encodings that use a fixed number of bytes per character, like UTF-16 or ASCII. The performance for UTF-8 is very good for documents that use mostly characters of the European languages. For the same encoding the rendering performance is high for files consisting of short lines (up to a few thousand characters) and may degrade for long lines.

Hex Viewer

When the Unicode characters that are visible in a text viewer or editor are not enough and you need to see the byte values of each character of a document, you can start the hex viewer that is available on the **Tools** menu. It has two panels: the characters are rendered in the right panel and the bytes of each character are displayed in the left panel. There is a 1:1 correspondence between the characters and their byte representation: the byte representation of a character is displayed in the same matrix position of the left panel as the character in the matrix of the right panel.
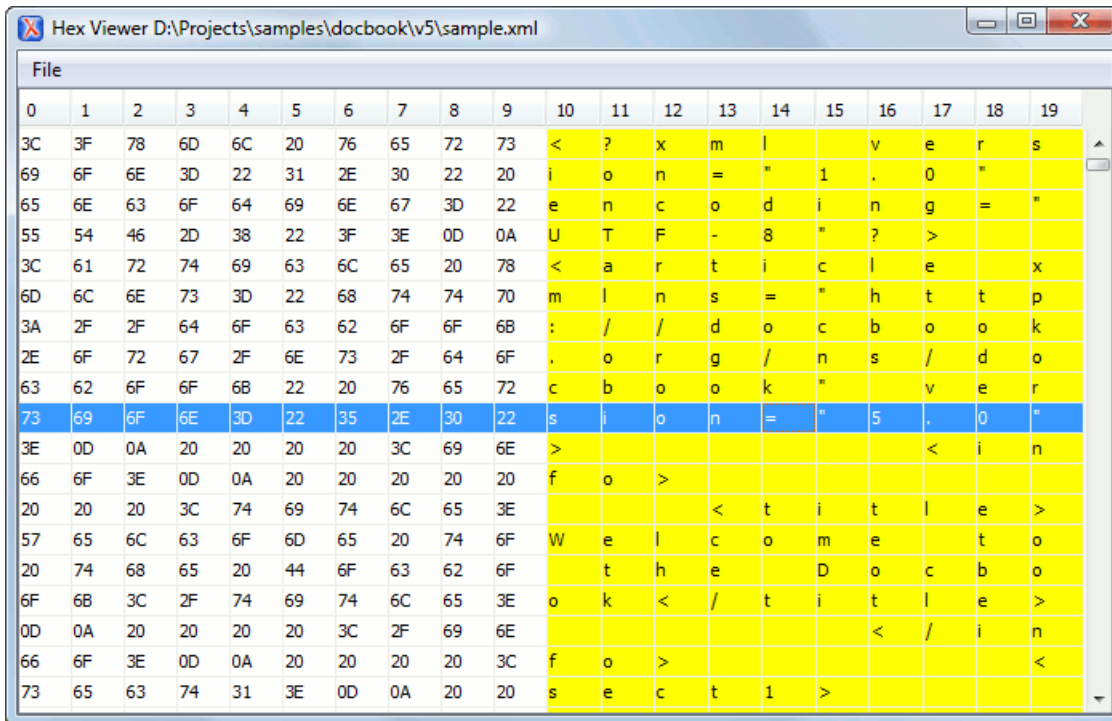
Figure 263: Hex Viewer

To open a file in **Hex Viewer** use the **File** > **Open** action. Alternatively, you can drag a file and drop it in the **Hex Viewer** panel.

Integrating External Tools

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

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

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

Integrating the Ant Tool

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

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

```
ant clean
```

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

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

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

BUILD SUCCESSFUL
Total time: 1 second
```

Chapter

18

Extending Oxygen XML Author with Plugins

A plugin is a software component which adds extra functionality to the standalone version of the application using a series of application-provided extension points.

This chapter explains how to write and install a plugin for the standalone version of Oxygen XML Author. The *Plugins Development Kit* contains sample plugins (source and compiled Java code) and the Javadoc API necessary for developing custom plugins.

If you want to customize the Oxygen XML Author Eclipse Plugin you can look at the *Eclipse IDE Integration Sample Project* to see how an Eclipse plugin can interact with the Oxygen XML Author APIs.

Introduction

A plugin can have multiple plugin extensions. The following plugin extensions are available (in the order of importance).

Plugins which are used in the entire workspace:

- *Workspace Access Plugin Extension* on page 709
- *Components Validation Plugin Extension* on page 711
- *Custom Protocol Plugin Extension* on page 712
- *Resource Locking Custom Protocol Plugin Extension* on page 712
- *Open Redirect Plugin Extension* on page 713

Plugins which work only in the Text editing mode:

- *General Plugin Extension* on page 715
- *Selection Plugin Extension* on page 715
- *Document Plugin Extension* on page 715

General configuration of an Oxygen XML Author plugin

The Oxygen XML Author functionality can be extended with plugins that implement a clearly specified API.

On the Oxygen XML Author website there is an *SDK* with sample plugins (source and compiled Java code) and the Javadoc API necessary for developing custom plugins.

The minimal implementation of a plugin must provide:

- a Java class that extends the `ro.sync.exml.plugin.Plugin` class
- a Java class that implements the `ro.sync.exml.plugin.PluginExtension` interface
- a plugin descriptor file called `plugin.xml`

A `ro.sync.exml.plugin.PluginDescriptor` object is passed to the constructor of the subclass of the `ro.sync.exml.plugin.Plugin` class. It contains the following data items about the plugin:

- basedir - *File* object - the base directory of the plugin.
- description - *String* object - the description of the plugin.
- name - *String* object - the name of the plugin.
- vendor - *String* object - the vendor name of the plugin.
- version - *String* object - the plugin version.
- id - *String* object - an unique identifier.
- classLoaderType - a choice between `preferOxygenResources` (default value) and `preferReferencedResources`. When choosing `preferOxygenResources` the libraries which are referenced in the Oxygen XML Author `lib` directory will have precedence over those referred in the `plugin.xml` configuration file, if they have the same package names. When choosing `preferReferencedResources` the libraries which are referenced in the `plugin.xml` configuration file will have precedence over those found in the Oxygen XML Author `lib` directory, if they have the same package names.

The *`ro.sync.exml.plugin.PluginDescriptor`* fields are filled with information from the `plugin.xml` configuration file.

The plugin descriptor is an XML file that defines how the plugin is integrated in Oxygen XML Author and what libraries are loaded. The structure of the plugin descriptor file is fully described in a DTD grammar located in `[OXYGEN_DIR]/plugins/plugin.dtd`. Here is a sample plugin descriptor used by the *Capitalize Lines* sample plugin:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin
    name="Capitalize Lines"
```

```
        description="Capitalize the first character on each line"
        version="1.0.0"
        vendor="SyncRO"
        class="ro.sync.sample.plugin.caplines.CapLinesPlugin">
        <runtime>
            <library name="lib/caplines.jar"/>
        </runtime>
        <extension type="selectionProcessor"
        class="ro.sync.sample.plugin.caplines.CapLinesPluginExtension" keyboardShortcut="ctrl shift EQUALS"/>
</plugin>
```

If your plugin is of type *Selection*, *Document* or *General*, and thus contributes an action either to the contextual menu or to the main menu of the Text editing mode, then you can assign a keyboard shortcut for it. You can use the keyboardShortcut attribute for each extension element to specify the desired shortcut.

> *i* **Tip:** To compose string representations of the desired shortcut keys you can go to the Oxygen XML Author **Menu Shortcut Keys** preferences page, press **Edit** on any action, press the desired key sequence and use the representation which appears in the edit dialog.

Referencing libraries

To refer libraries, use either of the following elements:

* `<library name="libraryName" scope="global"/>` to point to specific libraries
* `<librariesFolder name="libraryFolderPath" scope="global"/>` to point to multiple libraries located in the specified folder

Both elements support the scope attribute that defines the loading priority. It can have one of the following three values:

* *local* - the library is loaded in the plugin's own class loader. This is the default behaviour
* *global* - the library is loaded in the main application class loader as the last library in the list (as if it would be present in the application lib directory)
* *globalHighPriority* - the library is loaded in the main application class loader as the first library in the list (useful to patch certain resources located in other JARs of the application)

Installation

To install a plugin in Oxygen XML Author, follow these steps:

1. Go to the Oxygen XML Author installation directory and locate the plugins directory.

   The plugins directory contains all the plugins available to Oxygen XML Author.

2. In the plugins directory create a subfolder to store the plugin files.

3. In the new folder, place the plugin descriptor file (plugin.xml), the Java classes of the plugin and the other files that are referenced in the descriptor file.

4. Restart Oxygen XML Author.

   Alternatively, you can go to **Help** > **Manage Addons** and use the **Manage Add-ons** dialog box. This dialog allows you to install available plugins and manage the ones that are currently in use.

Types of plugin extensions

A plugin can have one or more defined plugin extensions which provide functionality to the application.

Workspace Access Plugin Extension

This plugin type allows you to contribute actions to the main menu and toolbars of Oxygen XML Author, to create custom views and to interact with the application workspace, make modifications to opened documents and add listeners for various events.

Many complex integrations, like integrations with Content Management Systems (CMS) usually requires access to some workspace resources like the toolbar, menus and to the opened XML editors. This type of plugin is also useful because it allows you to make modifications to an opened editor's XML content.

The plugin must implement the interface *ro.sync.exml.plugin.workspace.WorkspaceAccessPluginExtension*. The callback method `applicationStarted` of this interface allows access to a parameter of type *ro.sync.exml.workspace.api.standalone.StandalonePluginWorkspace* which in its turn allows for API access to the application workspace.

The interface `StandalonePluginWorkspace` has three methods which can be called in order to customize the toolbars, menus and views:

- `addToolbarComponentsCustomizer` - Contributes to or modifies existing toolbars. You can specify in the associated `plugin.xml` descriptor additional toolbar IDs using the following construct:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomWorkspaceAccess" ..............>
 <runtime>
  ........
 </runtime>

 <extension type="WorkspaceAccess" ............../>
 ..............
  <toolbar id="SampleWorkspaceAccessToolbarID" initialSide="NORTH" initialRow="1"/>
</plugin>
```

  The `toolbar` element adds a toolbar in the Oxygen XML Author interface and allows you to contribute your own plugin specific actions. The following attributes are available:

  - `id` - unique identifier of the plugin toolbar
  - `initialSide` - specifies the place where the toolbar is initially displayed. The allowed values are NORTH and SOUTH.
  - `initialRow` - specifies the initial row on the specified side where the toolbar is displayed. For example the main menu has an initial row of "0" and the "Edit" toolbar has an initial row of "1".

  The *ro.sync.exml.workspace.api.standalone.ToolbarInfo* toolbar component information with the specified id will be provided to you by the customizer interface. You will thus be able to provide Swing components which will appear on the toolbar when the application starts.

- `addViewComponentCustomizer` - Contributes to or modifies existing views or contributes to the reserved custom view. You can specify in the associated `plugin.xml` descriptor additional view IDs using the following construct:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomWorkspaceAccess" ..............>
 <runtime>
  ........
 </runtime>

 <extension type="WorkspaceAccess" ............../>
 ..............
 <view id="SampleWorkspaceAccessID" initialSide="WEST" initialRow="0"/>
</plugin>
```

  The `view` element adds a view in the Oxygen XML Author interface and allows you to contribute your own plugin specific UI components. The following attributes are available:

  - `id` - unique identifier of the view component
  - `initialSide` - specifies the place where the view is initially displayed. The allowed values are NORTH, SOUTH, EAST and WEST.
  - `initialRow` - specifies the initial row on the specified side where the view is displayed. For example the **Project** view has an initial row of 0 and the Outline view has an initial row of 1. Both views are in the WEST part of the workbench.

The `ro.sync.exml.workspace.api.standalone.ViewInfo` view component information with the specified id will be provided to you by the customizer interface. You will thus be able to provide Swing components which will appear on the view when the application starts.

• `addMenuBarCustomizer` - Contributes to or modifies existing menu components.

Access to the opened editors can be done first by getting access to all URLs opened in the workspace using the API method `StandalonePluginWorkspace.getAllEditorLocations(int editingArea)`. There are two available editing areas: the DITA Maps Manager editing area (available only in Oxygen XML Editor and OxygenXML Author) where only DITA Maps are edited and the main editing area. Using the URL of an opened resource you can gain access to it using the `StandalonePluginWorkspace.getEditorAccess(URL location, int editingArea)` API method. A *`ro.sync.exml.workspace.api.editor.WSEditor`* allows then access to the current editing page. Special editing API is supported only for the **Text** (*`ro.sync.exml.workspace.api.editor.page.text.WSTextEditorPage`*) page and the **Author** (*`ro.sync.exml.workspace.api.editor.page.author.WSAuthorEditorPage`*) page.

In order to be notified when editors are opened, selected and closed you can use the API method `StandalonePluginWorkspace.addEditorChangeListener` to add a listener.

Option Page Plugin Extension

This extension type allows developers to add custom preference pages to the application **Preferences** dialog.

The extension must implement the `ro.sync.exml.plugin.option.OptionPagePluginExtension` interface. The provided callbacks allow the developer to create the custom Swing component which will be added to the page and to react to various calls in order to persistently save the page's settings using the `OptionsStorage` API.

All preferences pages which are contributed by a plugin appear listed in the **Preferences** dialog in the **Plugins** category. The `plugin.xml` configuration file can specify one or more such extensions using constructs like:

```
<extension type="OptionPage" class="my.package.CustomOptionPagePluginExtension"/>
```

Components Validation Plugin Extension

This plugin type allows the developer to make customization of the editor menus, toolbars, and some other components by allowing or filtering them from the user interface.

This plugin provides the following API:

• The interface `ComponentsValidatorPluginExtension` - There is one method that must be implemented:

   • `getComponentsValidator()` - Returns a *`ro.sync.exml.ComponentsValidator`* implementation class used for validating the menus, toolbars, and their actions.

• The `ComponentsValidator` interface provides methods to filter various features from being added to the GUI of Oxygen XML Author:

   • `validateMenuOrTaggedAction(String[] menuOrActionPath)` - Checks if a menu or a tag action from a menu is allowed and returns a `boolean` value. A tag is used to uniquely identifying an action. The `String[]` argument is the tag of the menu / action and the tags of its parent menus if any.
   • `validateToolbarTaggedAction(String[] toolbarOrAction)` - Checks if an action from a toolbar is allowed and returns a *boolean* value. The `String[]` argument is the tag of the action from a toolbar and the tag of its parent toolbar if any.
   • `validateComponent(String key)` - Checks if the given component is allowed and returns a `boolean` value. The `String` argument is the tag identifying the component. You can remove toolbars entirely using this callback.
   • `validateAccelAction(String category, String tag)` - Checks if the given accelerator action is allowed to appear in the GUI and returns a `boolean` value. An accelerator action can be uniquely identified so it will be removed both from toolbars or menus. The first argument represents the action category, the second is the tag of the action.

- `validateContentType(String contentType)` - Checks if the given content type is allowed and returns a `boolean` value. The `String` argument represents the content type. You can instruct Oxygen XML Author to ignore content types like `text / xsl` or `text / xquery`.
- `validateOptionPane(String optionPaneKey)` - Checks if the given options page can be added in the preferences option tree and returns a `boolean` value. The `String` argument is the option pane key.
- `validateOption(String optionKey)` - Checks if the given option can be added in the option page and returns a `boolean` value. The `String` argument is the option key. This method is mostly used for internal use and it is not called for each option in a preferences page.
- `validateLibrary(String library)` - Checks if the given library is allowed to appear listed in the **About** dialog and returns a `boolean` value. The `String` argument is the library. This method is mostly for internal use.
- `validateNewEditorTemplate(EditorTemplate editorTemplate)` - Checks if the given template for a new editor is allowed and returns a `boolean` value. The `EditorTemplate` argument is the editor template. An `EditorTemplate` is used to create an editor for a given extension. You can thus filter what appears in the **New** dialog list.
- `isDebuggerperspectiveAllowed()` - Check if the debugger perspective is allowed and returns a `boolean` value.
- `validateSHMarker(String marker)` - Checks if the given marker is allowed and returns a `boolean` value. The `String` argument represents the syntax highlight marker to be checked. If you decide to filter certain content types, you can also filter the syntax highlight options so that the content type is no longer present in the Preferences options tree.

> **Tip:** The best way to decide what to filter is to observe the values that Oxygen XML Author passes when these callbacks are called. You have to create an implementation for this interface which lists in the console all values received by each function. Then you can decide on the values to filter and act accordingly.

Custom Protocol Plugin Extension

This type of plugins allows the developer to work with a custom designed protocol for retrieving and storing files.

It provides the following API:

- The interface `URLStreamHandlerPluginExtension` - There is one method that must be implemented:
  - `getURLStreamHandler(String protocol)` - It takes as an argument the name of the protocol and returns a `URLStreamHandler` object, or null if there is no URL handler for the specified protocol.
- With the help of the `URLChooserPluginExtension2` interface, it is possible to write your own dialog that works with the custom protocol. This interface provides two methods:
  - `chooseURLs(StandalonePluginWorkspace workspaceAccess)` - Returns a `URL[]` object that contains the URLs the user decided to open with the custom protocol. You can invoke your own URL chooser dialog here and then return the chosen URLs having your own custom protocol. You have access to the workspace of Oxygen XML Author.
  - `getMenuName()` - Returns a `String` object that is the name of the entry added in the **File** menu.
- With the help of the `URLChooserToolbarExtension` interface, it is possible to provide a toolbar entry which is used for launching the custom URLs chooser from the `URLChooserPluginExtension` implementation. This interface provides two methods:
  - `getToolbarIcon()` - Returns the `javax.swing.Icon` image used on the toolbar.
  - `getToolbarTooltip()` - Returns a `String` that is the tooltip used on the toolbar button.

Resource Locking Custom Protocol Plugin Extension

This plugin type allows the developer to work with a custom designed protocol for retrieving and storing files. It can lock a resource on opening it in Oxygen XML Author. This type of plugin extends the custom protocol plugin type with resource locking support.

Such a plugin provides the following API:

- The interface `URLStreamHandlerWithLockPluginExtension` - The plugin receives callbacks following the simple protocol for resource locking and unlocking imposed by Oxygen XML Author.

  There are two additional methods that must be implemented:

  - `getLockHandler()` - Returns a `LockHandler` implementation class with the implementation of the lock specific methods from the plugin.
  - `isSupported(String protocol)` - Returns a `boolean` that is `true` if the plugin accepts to manage locking for a certain URL protocol scheme like `ftp`, `http`, `https`, or `customName`.

Open Redirect Plugin Extension

This type of plugin is useful for opening more than one file with only one open action.

For example when a zip archive or an ODF file or an OOXML file is open in the **Archive Browser** view a plugin of this type can decide to open a file also from the archive in an XML editor panel. This file can be the `document.xml` main file from an OOXML file archive or a specific XML file from a zip archive.

The plugin must implement the interface `OpenRedirectExtension`. It has only one callback: `redirect(URL)` that receives the URL of the file opened by the Oxygen XML Author user. If the plugin decides to open also other files it must return an array of information objects (`OpenRedirectInformation[]`) that correspond to these files. Such an information object must contain the URL that is opened in a new editor panel and the content type, for example `text/xml`. The content type is used for determining the type of editor panel. A `null` content type allows auto-detection of the file type.

Targeted URL Stream Handler Plugin Extension

This type of plugin can be used when it is necessary to impose custom URL stream handlers for specific URLs.

This plugin extension can handle the following protocols: `http`, `https`, `ftp` or `sftp`, for which Oxygen XML Author usually provides specific fixed URL stream handlers. If it is set to handle connections for a specific protocol, this extension will be asked to provide the URL stream handler for each opened connection of an URL having that protocol.

To use this type of plugin, you have to implement the `ro.sync.exml.plugin.urlstreamhandler.TargetedURLStreamHandlerPluginExtension` interface, that provides the following methods:

- `boolean canHandleProtocol(String protocol)`

  This method checks if the plugin can handle a specific protocol. If this method returns `true` for a specific protocol, the `getURLStreamHandler(URL)` method will be called for each opened connection of an URL having this protocol.

- `URLStreamHandler getURLStreamHandler(URL url)`

  This method provides the URL handler for the specified URL and it is called for each opened connection of an URL with a protocol for which the `canHandleProtocol(String)` method returns `true`.

  If this method returns `null`, the default Oxygen XML Author `URLStreamHandler` is used.

To use this type of extension in your plugin, create an extension of `TargetedURLHandler` type in your `plugin.xml` and specify the class that implements `TargetedURLStreamHandlerPluginExtension`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomTargetedURLStreamHandlerPlugin" ..............>
 <runtime>
  ........
 </runtime>

 <extension type="TargetedURLHandler" class="CustomTargetedURLStreamHandlerPluginExtension"/>
 ..............

</plugin>
```

This extension can be useful in situations when connections opened from a specific host must be handled in a particular way. For example, the Oxygen XML Author HTTP URLStreamHandler may not be compatible for sending and receiving SOAP using the SUN Web Services implementation. In this case you can override the stream handler set by Oxygen XML Author for HTTP to use the default SUN URLStreamHandler which is more compatible with sending and receiving SOAP requests.

```
public class CustomTargetedURLStreamHandlerPluginExtension
   implements TargetedURLStreamHandlerPluginExtension {

  @Override
  public boolean canHandleProtocol(String protocol) {
    boolean handleProtocol = false;
    if ("http".equals(protocol) || "https".equals(protocol)) {
      // This extension handles both HTTP and HTTPS protocols
      handleProtocol = true;
    }
    return handleProtocol;
  }

  @Override
  public URLStreamHandler getURLStreamHandler(URL url) {
    // This method is called only for the URLs with a protocol
    // for which the canHandleProtocol(String) method returns true (HTTP and HTTPS)

    URLStreamHandler handler = null;

    String host = url.getHost();
    String protocol = url.getProtocol();
    if ("some_host".equals(host)) {
      // When there are connections opened from some_host, the SUN HTTP(S)
      // handlers are used
      if ("http".equals(protocol)) {
        handler = new sun.net.www.protocol.http.Handler();
      } else {
        handler = new sun.net.www.protocol.https.Handler();
      }
    }
    return handler;
  }
}
```

Lock Handler Factory Plugin Extension

This type of extension is used for locking resources from a specific protocol.

It provides the following API:

• The interface LockHandlerFactoryPluginExtension.

  You need to implement the following two methods:

  • LockHandler getLockHandler()

    Gets the lock handler for the current handled protocol. Might be null if not supported.

  • boolean isLockingSupported(String protocol)

    Checks if a lock handler can be provided for a specific protocol.

To use this type of extension in your plugin, create an extension of LockHandlerFactory type in your plugin.xml and specify the class implementing LockHandlerFactoryPluginExtension:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin name="CustomLockHandler" ..............>
 <runtime>
  ........
 </runtime>

 <extension type="LockHandlerFactory" class="LockHandlerFactoryPluginExtensionImpl"/>
 ...............

</plugin>
```

StylesFilter Plugin Extension

This plugin type allows the developer to dynamically modify the CSS styles used to render elements in the **Author** mode.

The plugin must extend the *ro.sync.exml.plugin.author.css.filter.GeneralStylesFilterExtension* class. This class has a callback on which you can alter the styles for an Author element.

This extension point is similar with the Styles Filter that you set at the *Document Type* level. The only difference is that the plugin filters styles from any opened XML document, regardless of the document type. The changes made by this plugin are prioritised over the changes made by the Document Type level filter.

Plugin Extensions designed to work only in the Text Editing Mode

These plugin extensions operate only when editing documents in the Text mode. They are mounted automatically by the application on the contextual menu in the **Plugins** submenu.

General Plugin Extension

This plugin type allows the developer to invoke custom code and to interact with the workspace of Oxygen XML Author.

This plugin is the most general plugin type. It provides a limited API:

- The interface `GeneralPluginExtension` - Intended for general-purpose plugins, kind of external tools but triggered from the **Plugins** menu. The implementing classes must provide the method `process(GeneralPluginContext)` which must provide the plugin processing. This method takes as a parameter a `GeneralPluginContext` object.
- The class `GeneralPluginContext` - Represents the context in which the general plugin extension does its processing. The `getPluginWorkspace()` method allows you access to the workspace of Oxygen XML Author.

Selection Plugin Extension

A selection plugin can be applied to both an XML document and a non-XML document. It works as follows: the user makes a selection in the editor, displays the contextual menu, and selects from the **Plugins** submenu the item corresponding to the plugin.

This plugin type provides the following API:

- The interface `SelectionPluginExtension` - The context containing the selected text is passed to the extension and the processed result is going to replace the initial selection. The `process(GeneralPluginContext)` method must return a `SelectionPluginResult` object which contains the result of the processing. The `String` value returned by the `SelectionPluginResult` object can include editor variables like *${caret}* and *${selection}*.
- The `SelectionPluginContext` object represents the context. It provides four methods:
  - `getSelection()` - Returns a `String` that is the current selection of text.
  - `getFrame()` - Returns a `Frame` that is the editing frame.
  - `getPluginWorkspace()` - Returns access to the workspace of Oxygen XML Author.
  - `getDocumentURL()` - Returns the URL of the current edited document.

Document Plugin Extension

This plugin type can be applied only to an XML document. It can modify the current document which is received as callback parameter.

The plugin is started by selecting the corresponding menu item from the contextual menu of the XML editor (Text mode), **Plugins** submenu. It provides the following API:

- The interface `DocumentPluginExtension` - Receives the context object containing the current document in order to be processed. The `process(GeneralPluginContext)` method can return a `DocumentPluginResult` object containing a new document.
- The `DocumentPluginContext` object represents the context. It provides three methods:
  - `getDocument()` - Returns a `javax.swing.text.Document` object that represents the current document.

- `getFrame()` - Returns a `java.awt.Frame` object that represents the editing frame.
- `getPluginWorkspace()` - Returns access to the workspace of Oxygen XML Author.

How to

Different tutorials about how to implement complex plugins.

How to Write a CMS Integration Plugin

In order to have a complete integration between Oxygen XML Author and any CMS you usually have to write a plugin which combines two available plugin extensions:

- *Workspace Access*
- *Custom protocol*

The usual set of requirements for an integration between Oxygen XML Author and the CMS are the following:

- Contribute to the Oxygen XML Author toolbars and main menu with your custom **Check Out** and **Check In** actions:

  - **Check Out** triggers your custom dialogs which allow you to browse the remote CMS and choose the resources you want to open;
  - **Check In** allows you to send back to the server the modified content.

    You can use the **Workspace Access** plugin extension (and provided sample Java code) for all these operations.

- When **Check Out** is called, use the Oxygen XML Author API to open your custom URLs (URLs created using your custom protocol). It is important to implement and use a **Custom Protocol** extension in order to be notified when the files are opened and saved and to be able to provide to Oxygen XML Author the content for the relative references the files may contain. Your custom `java.net.URLStreamHandler` implementation checks out the resource content from the server, stores it locally and provides its content. Sample **Check Out** implementation:

```
/**
 * Sample implementation for the "Check Out" method.
 *
 * @param pluginWorkspaceAccess The plugin workspace access (Workspace Access plugin).
 * @throws MalformedURLException
 */
private void checkOut(StandalonePluginWorkspace pluginWorkspaceAccess) throws MalformedURLException {
    //TODO Show the user a custom dialog for browsing the CMS
    //TODO after the user selected the resource create an URL with a custom protocol
    // which will uniquely map to the resource on the CMS using the URLHandler
    //something like:
    URL customURL = new URL("mycms://host/path/to/file.xml");
    //Ask Oxygen to open the URL
    pluginWorkspaceAccess.open(customURL);
    //Oxygen will then your custom protocol handler to provide the contents for the resource
"mycms://host/path/to/file.xml"
    //Your custom protocol handler will check out the file in a temporary directory for example and provide
the content from it.
    //Oxygen will also pass through your URLHandler if you have any relative references which need to be
opened/obtained.
  }
```

Here is a diagram of the **Check Out** process:
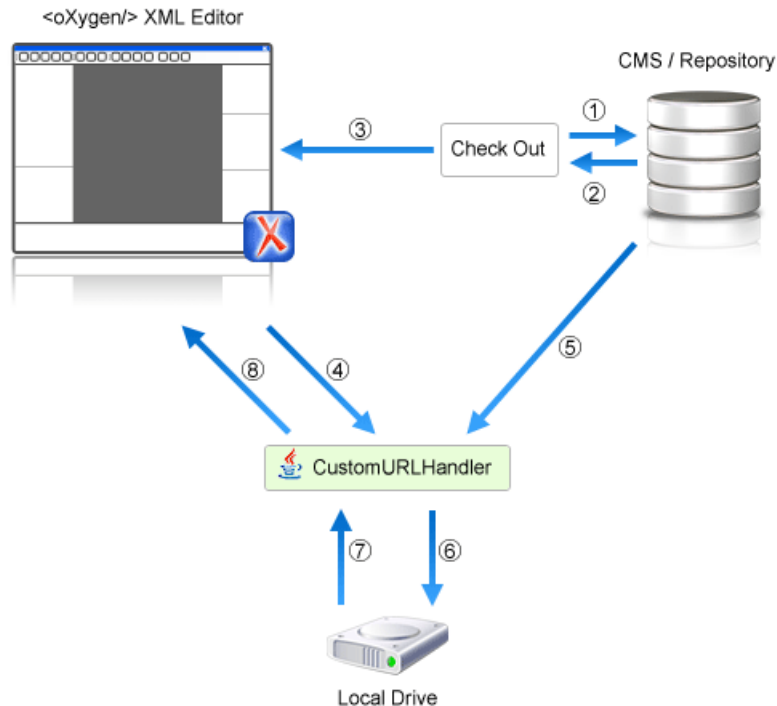
Each phase is described below:

1. Browse CMS repository
2. User chooses a resource
3. Use API to open custom URL: `mycms://path/to/file.xml`
4. Get content of URL: `mycms://path/to/file.xml`
5. Get content of resource
6. Store on disk for faster access
7. Retrieve content from disk if already checked out
8. Retrieved content

- Contribute a special **Browse CMS** action to every dialog in Oxygen XML Author where an URL can be chosen to perform a special action (like the **Insert a DITA Content Reference** action or the **Insert Image Reference** action). Sample code:

```
        //Add an additional browse action to all dialogs/places where Oxygen allows selecting an URL.
        pluginWorkspaceAccess.addInputURLChooserCustomizer(new InputURLChooserCustomizer() {
          public void customizeBrowseActions(List<Action> existingBrowseActions, final InputURLChooser chooser)
  {
            //IMPORTANT, you also need to set a custom icon on the action for situations when its text is not
used for display.
            Action browseCMS = new AbstractAction("CMS") {
              public void actionPerformed(ActionEvent e) {
                URL chosenResource = browseCMSAndChooseResource();
                if (chosenResource != null) {
                  try {
                    //Set the chosen resource in the dialog's combo box chooser.
                    chooser.urlChosen(chosenResource);
                  } catch (MalformedURLException e1) {
                    //
                  }
                }
              }
            };
            existingBrowseActions.add(browseCMS);
          }
        });
```

When inserting references to other resources using the actions already implemented in Oxygen XML Author, the reference to the resource is made by default relative to the absolute location of the edited XML file. You can gain control over the way in which the reference is made relative for a specific protocol like:

```
//Add a custom relative reference resolver for your custom protocol.
//Usually when inserting references from one URL to another Oxygen makes the inserted path relative.
//If your custom protocol needs special relativization techniques then it should set up a custom relative

//references resolver to be notified when resolving needs to be done.
pluginWorkspaceAccess.addRelativeReferencesResolver(
    //Your custom URL protocol for which you already have a custom URLStreamHandlerPluginExtension set
up.
    "mycms",
    //The relative references resolver
    new RelativeReferenceResolver() {
  public String makeRelative(URL baseURL, URL childURL) {
    //Return the referenced path as absolute for example.
    //return childURL.toString();
    //Or return null for the default behavior.
    return null;
  }
});
```

• Write the `plugin.xml` descriptor. Your plugin combines the two extensions using a single set of libraries. The descriptor would look like:

```
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin
 name="CustomCMSAccess"
 description="Test"
 version="1.0.0"
 vendor="ACME"
 class="custom.cms.CMSAccessPlugin">
 <runtime>
  <library name="lib/cmsaccess.jar"/>
 </runtime>
 <!--Access to add actions to the main menu and toolbars or to add custom views.-->
 <!--See the "ro.sync.sample.plugin.workspace.CustomWorkspaceAccessPluginExtension" Java sample for more
details-->
 <extension type="WorkspaceAccess"
  class="custom.cms.CustomWorkspaceAccessPluginExtension"/>
 <!--The custom URL handler which will communicate with the CMS implementation-->
 <!--See the "ro.sync.sample.plugin.workspace.customprotocol.CustomProtocolURLHandlerExtension" Java sample
for more details-->
 <extension type="URLHandler"
  class="custom.cms.CustomProtocolURLHandlerExtension"/>
</plugin>
```

• Create a `cmsaccess.jar` JAR archive containing your implementation classes.

• Copy your new plugin directory in the `plugins` subfolder of the Oxygen XML Author install folder and start Oxygen XML Author.

Class Loading Issues

It is possible that the Java libraries you have specified in the plugin libraries list conflict with the ones already loaded by Oxygen XML Author. In order to instruct the plugin to prefer its libraries over the ones used by Oxygen XML Author, you can add the following attribute on the `<plugin>` root element:`classLoaderType="preferReferencedResources"` from the `plugin.xml` descriptor.

A **Late Delegation Class Loader** (the main class loader in Oxygen XML Author) is a `java.net.URLClassLoader` extension which prefers to search classes in its own libraries list and only if a class is not found there to delegate to the parent class loader.

The main Oxygen XML Author Class Loader uses as libraries all jars specified in the `[OXYGEN_DIR]\lib` directory. Its parent class loader is the default JVM Class loader. For each instantiated plugin a separate class loader is created having as parent the Oxygen XML Author Class Loader.

The plugin class loader can be either a standard `java.net.URLClassLoader` or a `LateDelegationClassLoader` (depending on the attribute `classLoaderType` in the `plugin.xml`). Its parent class loader is always the Oxygen XML Author `LateDelegationClassLoader`.

If you experience additional problems like the following:

```
java.lang.LinkageError: ClassCastException: attempting to cast
jar:file:/C:/jdk1.6.0_06/jre/lib/rt.jar!/javax/xml/ws/spi/Provider.classtojar:file:/D:/Program
    Files/Oxygen XML Editor
    12/plugins/wspcaccess/../../xdocs/lib/jaxws/jaxws-api.jar!/javax/xml/ws/spi/Provider.class
    at javax.xml.ws.spi.Provider.provider(Provider.java:94) at
    javax.xml.ws.Service.<init>(Service.java:56)
...................................................................
```

The cause could be the fact that some classes are instantiated using the context class loader of the current thread. The most straightforward fix is to write your code in a *try/finally* statement:

```
ClassLoader oldClassLoader = Thread.currentThread().getContextClassLoader();
try {
  //This is the implementation of the WorkspaceAccessPluginExtension plugin interface.
  Thread.currentThread().setContextClassLoader(
      CustomWorkspaceAccessPluginExtension.this.getClass().getClassLoader());
  //WRITE YOUR CODE HERE
} finally {
  Thread.currentThread().setContextClassLoader(oldClassLoader);
}
```

How to Write A Custom Protocol Plugin

For creating a custom protocol plugin, apply the following steps:

1. Write the handler class for your protocol that implements the `java.net.URLStreamHandler` interface.

Be careful to provide ways to correct and uncorrect the URLs of your files.

2. Write the plugin class by extending *ro.sync.exml.plugin.Plugin*.

3. Write the plugin extension class that implements the *ro.sync.exml.plugin.urlstreamhandler.URLStreamHandlerPluginExtension* interface.

It is necessary that the plugin extension for the custom protocol implements the `URLStreamHandlerPluginExtension` interface. Without it, you cannot use your plugin, because Oxygen XML Author is not able to find the protocol handler.

You can choose also to implement the *URLChooserPluginExtension* interface. It allows you to write and display your own customized dialog for selecting resources that are loaded with the custom protocol.

An implementation of the extension *URLHandlerReadOnlyCheckerExtension* allows you to:

• mark a resource as read-only when it is opened
• switch between marking the resource as read-only and read-write while it is edited

It is useful when opening and editing CMS resources.

4. Write the `plugin.xml` descriptor.

Remember to set the name of the plugin class to the one from the second step and the plugin extension class name with the one you have chosen at step 3.

5. Create a .jar archive with all these files.

6. Install your new plugin in the `plugins` subfolder of the Oxygen XML Author install folder.

How to Pack and Deploy an Add-on

Packing a Plugin or Framework as an Add-on

This procedure is suitable for developers who want a better control over the add-on package or those who want to automate some of the steps:

1. Pack the plugin or framework as a `ZIP` file or a *JAR*. Please note that you should pack the entire root directory not just its contents.

2. Digitally sign the package. Please note that you can perform this step only if you have created a *JAR* at the previous step. You will need a certificate signed by a trusted authority. To sign the jar you can either use the `jarsigner`

command line tool inside Oracle's Java Development Kit. (`[JDK_DIR]/bin/jarsigner.exe`) or, if you are working with *Ant*, you can use the `signjar` task (which is just a front for the `jarsigner` command line tool).

> 📋 **Note:** The benefit of having a signed add-on is that the user can verify the integrity of the add-on issuer. If you don't have such a certificate you can generate one yourself using the keytool command line tool. Please note that this approach is mostly recommended for tests since anyone can create a self signed certificate.

3. Create a descriptor file. You can use a template that Oxygen XML Author provides. To use this template, go to **File** > **New** and select the **Oxygen add-ons update site** template. Once deployed, this descriptor file is referred as *update site*.

Alternatively, you can use the **Add-ons Packager** plugin by following this procedure:

1. Install the **Add-ons Packager** plugin from *http://www.oxygenxml.com/InstData/Addons/optional/updateSite.xml* as described in the *Installing Add-ons procedure*.
2. Restart Oxygen XML Author. If the add-on is correctly installed, the **Add-ons packager** toolbar action is available.
3. Invoke the **Add-ons packager** toolbar action and input the required information in the displayed dialog box.
4. Press **OK** to complete the packaging process.

Deploying an Add-on

To deploy an add-on, copy the ZIP/*JAR* file and the descriptor file to an HTTP server. The URL to this location serves as the *Update Site URL*.

How to Share the Classloader Between a Framework and a Plugin

In some cases you may need to extend the functionality of Oxygen XML Author both through a framework and through a plugin. Normally, a framework and a plugin both run in their own private classloader. If the framework and the plugin use the same JAVA extensions/classes, it is recommended that they share the same classloader. This way, the common classes are loaded by only one classloader and they will both use the same static objects and have the ability to cast objects between one another.

To do this, *open the **Preferences** dialog* and go to **Document Type Association**, select the document type, go to the **Classpath** tab and in the **Use parent classloader from plugin with ID** fields introduce the ID of the plugin. This ID is declared in the *configuration file of the plugin*.

> ⚠ **Important:** The shared classed must be specified only in the configuration files of the plugin, and not in the configuration file and the document type class path at the same time.

Example - A Selection Plugin

The following plugin is called `UppercasePlugin` and is an example of *selection plugin.*. It is used in Oxygen XML Author for capitalizing the characters in the current selection. This example consists of two Java classes and the plugin descriptor:

- `UppercasePlugin.java`:

```
package ro.sync.sample.plugin.uppercase;

import ro.sync.exml.plugin.Plugin;
import ro.sync.exml.plugin.PluginDescriptor;

public class UppercasePlugin extends Plugin {
    /**
     * Plugin instance.
     */
    private static UppercasePlugin instance = null;

    /**
     * UppercasePlugin constructor.
     *
     * @param descriptor Plugin descriptor object.
     */
```

```java
    public UppercasePlugin(PluginDescriptor descriptor) {
        super(descriptor);

        if (instance != null) {
            throw new IllegalStateException("Already instantiated !");
        }
        instance = this;
    }

    /**
     * Get the plugin instance.
     *
     * @return the shared plugin instance.
     */
    public static UppercasePlugin getInstance() {
        return instance;
    }
}
```

• `UppercasePluginExtension.java`:

```java
package ro.sync.sample.plugin.uppercase;

import ro.sync.exml.plugin.selection.SelectionPluginContext;
import ro.sync.exml.plugin.selection.SelectionPluginExtension;
import ro.sync.exml.plugin.selection.SelectionPluginResult;
import ro.sync.exml.plugin.selection.SelectionPluginResultImpl;

public class UppercasePluginExtension implements SelectionPluginExtension {
    /**
     * Convert the text to uppercase.
     *
     *@param  context  Selection context.
     *@return          Uppercase plugin result.
     */
    public SelectionPluginResult process(SelectionPluginContext context) {
        return new SelectionPluginResultImpl(
            context.getSelection().toUpperCase());
    }
}
```

• `plugin.xml`:

```xml
<!DOCTYPE plugin SYSTEM "../plugin.dtd">
<plugin
    name="UpperCase"
    description="Convert the selection to uppercase"
    version="1.0.0"
    vendor="SyncRO"
    class="ro.sync.sample.plugin.uppercase.UppercasePlugin">
    <runtime>
        <library name="lib/uppercase.jar"/>
    </runtime>
    <extension type="selectionProcessor"
     class="ro.sync.sample.plugin.uppercase.UppercasePluginExtension"/>
</plugin>
```

Creating and Running Automated Tests

If you have developed complex custom plugins and/or document types the best way to test your implementation and insure that further changes will not interfere with the current behavior is to make automated tests for your customization.

An Oxygen XML Author installation standalone (Author or Editor) comes with a main `oxygen.jar` library located in the `[OXYGEN_DIR]`. That JAR library contains a base class for testing developer customizations named `ro.sync.exml.workspace.api.PluginWorkspaceTCBase`.

Please see below some steps in order to develop `JUnit` tests for your customizations using the **Eclipse** workbench:

1. Create a new Eclipse Java project and copy to it the entire contents of the `[OXYGEN_DIR]`.
2. Add to the **Java Build Path**->**Libraries** tab all JAR libraries present in the `[OXYGEN_DIR]/lib` directory. Make sure that the main JAR library `oxygen.jar` or `oxygenAuthor.jar` is the first one in the Java classpath by moving it up in the **Order and Export** tab.
3. Click **Add Library** and add the `JUnit` libraries.

4. Create a new Java class which extends `ro.sync.exml.workspace.api.PluginWorkspaceTCBase`.

5. Pass on to the constructor of the super class the following parameters:

   - `File frameworksFolder` The file path to the frameworks directory. It can point to a custom frameworks directory where the custom framework resides.
   - `File pluginsFolder` The file path to the plugins directory. It can point to a custom plugins directory where the custom plugins resides.
   - `String licenseKey` The license key used to license the test class.

6. Create test methods which use the API in the base class to open XML files and perform different actions on them. Your test class could look something like:

```java
public class MyTestClass extends PluginWorkspaceTCBase {

 /**
  * Constructor.
  */
 public MyTestClass() throws Exception {
  super(new File("frameworks"), new File("plugins"),
        "------START-LICENSE-KEY------\n" +
    "\n" +
    "Registration_Name=Developer\n" +
    "\n" +
    "Company=\n" +
    "\n" +
    "Category=Enterprise\n" +
    "\n" +
    "Component=XML-Editor, XSLT-Debugger, Saxon-SA\n" +
    "\n" +
    "Version=14\n" +
    "\n" +
    "Number_of_Licenses=1\n" +
    "\n" +
    "Date=09-04-2012\n" +
    "\n" +
    "Trial=31\n" +
    "\n" +
    "SGN=MCwCFGNoEGJSeiC3XCYIyalvjzHhGhhqAhRNRDpEu8RIWb8icCJO7HqfVP4++A\\=\\\\=\n" +
    "\n" +
  "-------END-LICENSE-KEY-------");
 }

  /**
     * <p><b>Description:</b> TC for opening a file and using the bold operation</p>
     * <p><b>Bug ID:</b> EXM-20417</p>
     *
     * @author radu_coravu
     *
     * @throws Exception
     */
   public void testOpenFileAndBoldEXM_20417() throws Exception {
     WSEditor ed = open(new File("D:/projects/eXml/test/authorExtensions/dita/sampleSmall.xml").toURL());
     //Move caret
     moveCaretRelativeTo("Context", 1, false);

     //Insert <b>
     invokeAuthorExtensionActionForID("bold");
     assertEquals("<?xml version=\"1.0\" encoding=\"utf-8\"?>\n" +
       "<!DOCTYPE task PUBLIC \"-//OASIS//DTD DITA Task//EN\"
\"http://docs.oasis-open.org/dita/v1.1/OS/dtd/task.dtd\">\n" +
       "<task id=\"taskId\">\n" +
       "    <title>Task <b>title</b></title>\n" +
       "    <prolog/>\n" +
       "    <taskbody>\n" +
       "        <context>\n" +
       "            <p>Context for the current task</p>\n" +
       "        </context>\n" +
       "        <steps>\n" +
       "            <step>\n" +
       "                <cmd>Task step.</cmd>\n" +
       "            </step>\n" +
       "        </steps>\n" +
       "    </taskbody>\n" +
       "</task>\n" +
       "", getCurrentEditorXMLContent());
   }
}
```

Debugging a Plugin Using the Eclipse Workbench

To debug problems in the code of the plugin without having to re-bundle the Java classes of the plugin in a JAR library, follow these steps:

1. Download and unpack an *all platforms standalone version* of Oxygen XML Author/Editor to a folder on your hard drive.

> 📄 **Note:** Name the folder `[OXYGEN_DIR]`.

2. Set up the Oxygen SDK following *this set of instructions*.

3. Create an Eclipse Java Project (let's call it `MyPluginProject`) from one of the sample plugins (the Workspace Access plugin for example).

4. In the Project root folder, create a folder called `myPlugin` and add the `plugin.xml` from the sample plugin in there. Modify the added plugin.xml to add a library reference to the project's classes directory: `<library name="../classes"/>`.

5. Copy the `plugin.dtd` from the `[OXYGEN_DIR]/plugins` folder in the root Project folder.

6. In the Project's build path add external JAR references to all the JAR libraries in the `[OXYGEN_DIR]/lib` folder. Now your Project should compile successfully.

7. Create a new Java Application configuration for debugging. The Main Class should be `ro.sync.exml.Oxygen`. The given VM Arguments should be:

```
-Dcom.oxygenxml.app.descriptor=ro.sync.exml.EditorFrameDescriptor -Xmx1024m
-XX:MaxPermSize=384m -Dcom.oxygenxml.editor.plugins.dir=D:\projects\MyPluginProject
```

8. Add a break point in one of the source Java classes.

9. Debug the created configuration. When the code reaches your breakpoint, the debug perspective should take over.

Disabling a Plugin

To disable a plugin, use one of the following two methods:

- *Open the Preferences dialog* , go to **Plugins**, and deselect the plugin that you want to disable;
- Create an empty file called `plugin.disable` next to the plugin configuration file (`plugin.xml`). The plugin will be disabled and will no longer be loaded by the application on startup.

> 📄 **Note:** This is useful if you want to temporarily stop work on a plugin and use the application without it.

Chapter

19

Configuring Oxygen XML Author

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.

Preferences

You can configure Oxygen XML Author options using the **Preferences** dialog.

To open the preferences dialog, go to **Options** > **Preferences**.

You can select the preference page you are interested in from the tree on the left of the **Preferences** dialog. You can filter the tree by typing in the filter box above the tree.
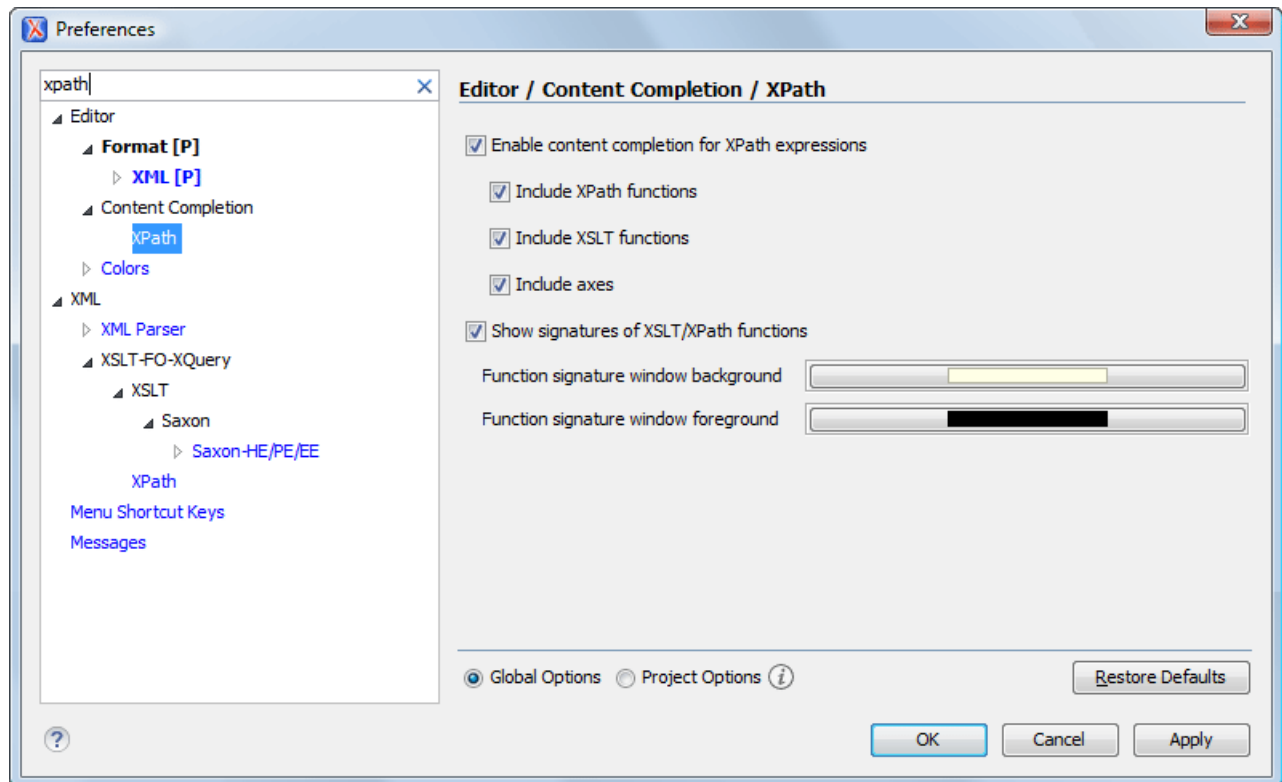
Figure 264: The Search field from the Preferences dialog

You can restore options to their default values by pressing the **Restore Defaults** button, available in each preferences page.

Press ⓘ of **F1** for help on any preferences page.

Global options and license information are stored in the following locations:

• [user-home-folder]\Application Data\com.oxygenxml.author for Windows XP

• [user-home-folder]\AppData\Roaming\com.oxygenxml.author for Windows Vista/7

• [user-home-folder]/Library/Preferences/com.oxygenxml.author for Mac OS X

• [user-home-folder]/.com.oxygenxml.author for Linux

Global Preferences

The global options cover a number of aspects of the overall operation of .To configure the **Global** options *open the Preferences dialog* and go to **Global**.

The following options are available in the **Global** preferences page:

• **Automatic Version Checking** - This option sets whether Oxygen XML Author will check for a new version on startup.

- **Language** - This option sets the language used in the user interface to English, French, German, Dutch, or Japanese. You must restart Oxygen XML Author for the change to take effect.
- **Other language** - This option sets the language used in the user interface using an interface localization file (either *properties* or XML). For details about creating this file, see *Localizing the User Interface* on page 808. You can use this option to set the language of the user interface to a language that is not shipped with Oxygen XML Author.

  > **Note:** If some interface labels are not rendered correctly after restarting the application, (for example Chinese or Korean characters are not displayed correctly), make sure that your operating system has the appropriate language pack installed (for example the East-Asian language pack).

- **Look and Feel** - This option lets you change the graphic style (look and feel) of the user interface.
- **Styles** - This option sets the style of the interface. On Windows the following styles are available:

  - Office 2003
  - Vsnet
  - Eclipse
  - Xerto
  - Default

  On Linux the following styles are available:

  - Eclipse
  - Default

  On OS X this option is not available.

  > **Note:** After changing the style, you must restart the application for the modification to take effect.

- **Themes** - This option sets the theme of the interface. This option is available only on Windows, and only if you have selected the**Office 2003** or **Default** styles. The following themes are available:

  - Normal Color
  - Home Stead
  - Metallic
  - Default
  - Gray

  > **Note:** This option is not available on Linux and OS X.

  > **Note:** On Windows, Oxygen XML Author inherits the high contrast themes colors set in the operating system if the **Look and Feel**, **Styles** and **Themes** options are set to their default values.

- **Line separator** - This option sets the line separator used when saving files. Use **System Default** to select the normal line separator for your OS. If you want the existing file separator of a file to be maintained, regardless of your current OS, check **Detect the line separator on file open**.
- **Detect the line separator on file open** - When this option is selected, the editor detects the line separator when a file is loaded and it uses it when the file is saved. New files are saved using the line separator defined by the**Line separator** option.
- **Default Internet browser** - This option sets the Web browser that Oxygen XML Author will use to:

  - open (X)HTML or PDF transformation results
  - open a web page (for example, pointing to specific paragraphs in the W3C recommendation of XML Schema in case of XML Schema validation errors)

  If you leave this setting blank, the system default browser will be used.

- **Open last edited files from project** - When this option is enabled,Oxygen XML Author opens the files you had open the last time you used a project whenever you open the application or switch to that project.

- **Beep on operation finished** - When this option is selected, Oxygen XML Author beeps when a validation or transform action ends. Different tones are used for success and failure. The tones used may depend on your operating system's sound settings.
- **Show memory status** - When this option is selected, the memoryOxygen XML Author uses is displayed in the status bar. To free memory, click the 🗑 **Free unused memory** button located at the right side of the status bar. The memory status bar turns yellow or red when Oxygen XML Author uses too much memory. You can change the amount of memory available to Oxygen by *changing the parameters of the application launcher*.
- **Check opened files for file system changes** - When this option is selected, Oxygen XML Author checks the content of the all opened editors to see if they have been updated by another application. If the file has changed, Oxygen XML Author will ask you if you want to reload the file.

  - **Auto update unmodified editors on file system changes** - If this option is selected, Oxygen XML Author automatically updates unmodified editors if the edited file changes externally.

- **Show Java vendor warning at startup** - If this option is selected, Oxygen XML Author displays a warning on startup if a non-recommended version of the Java virtual machine is being used.
- **File Chooser Dialog** - This options sets the directory that will be shown when the *Open file dialog* is displayed.

  - **Last visited directory** - The last visited folder will be displayed.
  - **Directory of the edited file** - The folder where the currently edited file is stored will be displayed.

- **Show hidden files and directories** - If this option is selected, Oxygen XML Author shows system hidden files and folders in the file browser dialog and the folder browser dialog. This setting is not available on OS X.

Add-ons Preferences

You can use add-ons to enhance the functionality of . To configure the **Add-ons** options, *open the **Preferences** dialog* and go to **Add-ons**. The following options are available in the Add-ons preferences page:

- **Enable automatic updates checking** - When this option is selected, Oxygen XML Author will search for available updates automatically.
- **Update Sites URLs** - This is a list of the URLs of add-on sites. You can add, edit, and delete sites in this list.

Fonts Preferences

Oxygen XML Author lets you choose the fonts used in the **Text** and **Grid** editor modes. The fonts for the Author mode editor are set in the associated CSS stylesheet.To configure the **Fonts** options, *open the **Preferences** dialog* and go to **Fonts**.

The following options are available:

- **Editor** - Sets the fonts used in the editor.

- 📄   **Note:** On Mac OS X, the default font, Monaco, cannot be rendered in bold.

- **Author default font** - This option sets the default font used **Author** mode. The default font will be overridden by the fonts specified in any CSS file associated with the opened document.
- **Text antialiasing** - This option sets the text anti-aliasing behavior:

  - **Default** - allows the application to use the setting of the operating system, if available.
  - **On** - sets the text anti-aliasing to pixel level.
  - **Off** - disables text anti-aliasing.
  - sub-pixel anti-aliasing modes, like GASP, LCD_HRGB, LCD_HBGR, LCD_VRGB, and LCD_VBGR.

- **Text components** - This option sets the font used in text boxes in the interface. After changing the font, restart the application for the change to take full effect.
- **GUI** - This option sets the font used for user interface labels. After changing the font, restart the application for the change to take full effect.

Document Type Association Preferences

Oxygen XML Author uses document type associations to associate a *document type* with a set of functionality provided by a framework. To configure the **Document Type Association** options, *open the **Preferences** dialog* and go to **Document Type Association**.

The following actions are available in the preferences panel:

- **Change framework directory location** - specifies a custom frameworks folder from where Oxygen XML Author loads the document types;
- **Document types table** - presents the currently defined document type associations, ordered by priority and alphabetically. Each edited document type has a set of association rules (rules used by the application to detect the proper document type association to use for an opened XML document). A rule is described by:

  - **Namespace** - specifies the namespace of the root element from the association rules set (** (any)* by default). If you want to apply the rule only when the root element 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 extend the document type.

- **Duplicate** - opens a new dialog allowing you to duplicate the configuration of an existing document type association;
- **Extend** - extend an existing document type, allowing you to add or remove functionality, starting from a base document type. All these changes will be saved as a patch. When the base document type is modified and evolves (from one application version to another, for example) the extension will evolve along with the base document type, allowing it to use the new features/actions added in the base document type.
- **Delete** - deletes the selected associations;
- **Enable DTD/XML Schema processing in document type detection** - when this option is enabled, the matching process also examines the DTD/XML Schema associated with the document. For example, the fixed attributes declared in the DTD for the root element are also analyzed, if this is specified in the association rules.

  This is especially useful if you are writing DITA customizations. DITA topics and maps are also matched by looking for the `DITAArchVersion` attribute of the root element. This attribute is specified as `default` in the DTD and it is detected in the root element, helping Oxygen XML Author to correctly match the DITA customization.

  This option is enabled by default;

- **Only for local DTD's / XML Schemas** - when the previous feature is enabled, you can choose with this option to process only the local DTD's / XML Schemas.

  This option is enabled by default.

- **Enable DTD/XML Schema caching** - when this option is enabled the associated DTDs or XML Schema are cached when parsed for the first time, improving performance when opening new documents with similar schema associations.

  This option is enabled by default.

Locations Preferences

Oxygen XML Author allows you to change the location where *frameworks* are stored, and to specify additional framework directories. The **Locations** preferences page allows you to specify the main `frameworks` folder location. You can choose between the **Default** directory (`[OXYGEN_DIR]/frameworks`) or a **Custom** specified directory. You can

also change the current `frameworks` folder location value using the `com.oxygenxml.editor.frameworks.url` system property set either in the application's *`.vmoptions` configuration files or in the startup scripts*.

A list of additional `frameworks` directories can also be specified. The application will look in each of those folders for additional document type configurations to load. Use the **Add**, **Edit** and **Delete** buttons to manage the list of folders.

A document type (configuration) can be loaded from the following locations:

- internal preferences - The document type configuration is stored in the application's *Internal preferences*
- additional frameworks directories - The document type configuration is loaded from one of the specified **Additional frameworks directories** list
- add-ons - An add-on can contribute a framework. You can manage the add-ons locations in the *Add-ons preferences page*
- the `frameworks` folder - The main folder containing framework configurations

All loaded document type configurations are first sorted by priority, then by document type name and then by load location (in the exact order specified above). When an XML document is opened, the application chooses the first document type configuration from the sorted list which matches the specific document.

All loaded document type configurations are first sorted by priority, then by document type

The Document Type Dialog

This dialog allows you to create or edit a *Document Type Association*. The following fields are available in this dialog:

- **Name** - the name of the *Document Type Association*;
- **Storage** - displays the type of location where the framework configuration file is stored. Can be one of **External** (framework configuration is saved in a file) or **Internal** (framework configuration is stored in the application's internal options);

  > **Note:** 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.

- **Description** - a detailed description of the framework;
- **Priority** - depending on the priority level, Oxygen XML Author establishes the order in which the existing document type associations are evaluated to determine the type of a document you are opening. It can be one of the following: Lowest, Low, Normal, High, Highest. You can set a higher priority to Document Type Associations you want to be evaluated first;
- **Initial edit mode** - sets the default edit mode when you open a document for the first time;

You are able to configure the options of each *framework* in the following tabs:

- *Association rules*;
- *Schema*;
- *Classpath*;
- *Author*;
- *Templates*;
- *Catalogs*;
- *Transformation*;
- *Validation*;
- *Extensions*.

The Association Rules Tab

By combining multiple association rules you can instruct Oxygen XML Author to identify the type of a document. An Oxygen XML Author *association rule* holds information about *Namespace*, *Root local name*, *File name*, *Public ID*, *Attribute*, and *Java class*. Oxygen XML Author identifies the type of a document when the document matches at least one of the *association rules*. Using the **Document type rule** dialog, you can create *association rules* that activate on any document matching all the criteria in the dialog.

In the **Association rules** tab you can perform the following actions:

✚ **New**

Opens the **document type rule** dialog allowing you to create *association rules*.

🔧 **Edit**

Opens the **document type rule** dialog allowing you to edit the properties of the currently selected *association rule*.

✖ **Delete**

Deletes the currently selected *association rules*.

⬆ **Move Up**

Moves the selection to the previous *association rule*.

⬇ **Move Down**

Moves the selection to the following *association rule*.

The Schema Tab

In the **Schema** tab you can specify a schema that Oxygen XML Author uses in case an XML document does not contain a schema declaration and no default validation scenario is associated with it.

To set the **Schema URL**, use *editor variables* to specify the path to the Schema file.

> 📝 **Note:** It is a good practice to store all resources in the framework directory and use the `${framework}` editor variable to refer them. This is a recommended approach to designing a self-contained document type that can be easily maintained and shared between different users.

The Classpath Tab

The **Classpath** tab displays a list of folders and JAR libraries that hold implementations for API extensions, implementations for custom Author operations, different resources (such as stylesheets), and framework translation files. Oxygen XML Author loads the resources looking in the folders in the order they appear in the list.

In the **Classpath** tab you can perform the following actions:

✚ **New**

Opens a dialog that allows you to add a resource in the **Classpath** tab.

🔧 **Edit**

Opens a dialog that allows you to edit a resource in the **Classpath** tab.

✖ **Delete**

Deletes the currently selected resource.

⬆ **Move Up**

Moves the selection to the previous resource.

⬇ **Move Down**

Moves the selection to the following resource.

The *Use parent classloader from plugin with ID* specifies the ID of a plugin. The current framework has access to the classes loaded for the plugin.

The Author Tab

The **Author** tab is a container that holds information regarding the CSS file used to render a document in the **Author** mode, and regarding framework-specific actions, menus, contextual menu, toolbar and content completion list of proposals.

The options that you configure in the **Author** tab are grouped in the following sub-tabs:

• *CSS*;
• *Actions*;
• *Menu*;

- *Contextual menu*;
- *Toolbar*;
- *Content Completion*.

CSS

The **CSS** sub-tab contains the CSS files that Oxygen XML Author uses to render a document in the **Author** mode. In this sub-tab, you set alternate CSS files. When you are editing a document in the **Author** mode, you are able to switch between these CSS files from the **Author CSS Alternatives** toolbar.

The following actions are available in the **CSS** sub-tab:

New

Opens a dialog that allows you to add a CSS file.

Edit

Opens a dialog that allows you to edit a CSS file.

Delete

Deletes the currently selected CSS file.

Move Up

Moves the selection to the previous CSS file.

Move Down

Moves the selection to the following CSS file.

ignore CSSs from the associated document type

The CSS files set in the CSS tab are overwritten by the CSS files specified in the document itself.

merge them with CSSs from the associated document type

The CSS files set in the CSS tab are merged with the CSS files specified in the document itself.

Actions

The **Actions** sub-tab holds the framework specific actions. Each action has an unique ID, a name, a description, and a shortcut key.

The following actions are available in this sub-tab:

New

Opens *the Action dialog* that allows you to add an action.

Duplicate

Duplicates the currently selected action.

Edit

Opens a dialog that allows you to edit an existing action.

Delete

Deletes the currently selected action.

The Action Dialog

To edit an existing document type action or create a new one, *open the Preferences dialog* and go to **Document Type Association**, select a document type, and click **Edit** or **New**. The **Document type** dialog is presented. In this dialog go to the **Author** tab, click **Actions**, select an action, and click **Edit** or **New**.
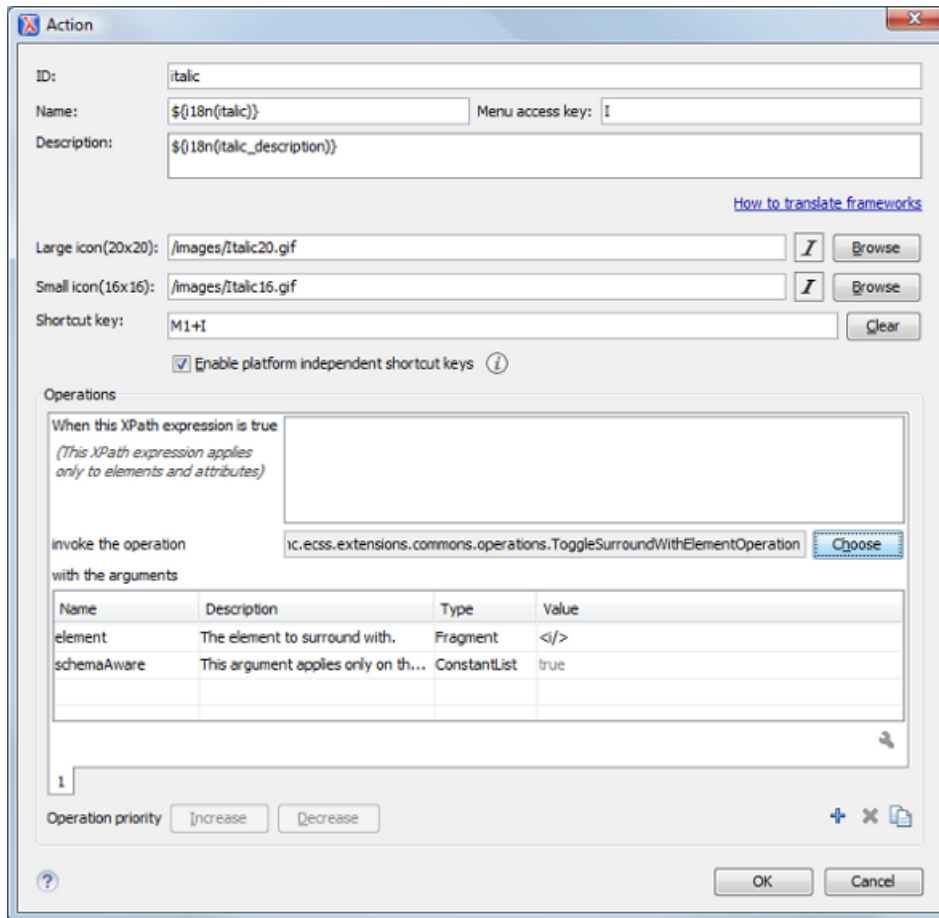
Figure 265: The Action Dialog

The following options are available:

- **ID** - specifies a unique action identifier;
- **Name** - specifies the name of the action. This name is displayed as a tooltip or as a menu item;
- **Menu access key** - on Windows, the menu items are accessed using the **Alt+"Letter"** shortcut, when the menu is visible. The letter is visually represented by underlining the first letter from the menu item name having the same value;
- **Description** - a description of the action;
- **Large icon** - select an image that Oxygen XML Author uses for the toolbar action.

  > **Note:** A good practice is to store the image files inside the framework directory and use the `${frameworks}` editor variable to make the image relative to the framework location. In case the images are bundled in a *jar* archive, together with some Java operations implementation for instance, it is convenient to refer the images by their relative path location in the *class-path*.

- **Small icon** - select an image that Oxygen XML Author uses for the contextual menu icon;
- **Shortcut key** - this field allows you to configure a shortcut key for the action you are editing. The + character separates the keys. The shortcut you are specifying in this field is platform-independent. The following modifiers are used:

  - M1 represents the **Command** key on MacOS X, and the **Ctrl** key on other platforms;
  - M2 represents the **Shift** key;
  - M3 represents the **Option** key on MacOS X, and the **Alt** key on other platforms;
  - M4 represents the **Ctrl** key on MacOS X, and is undefined on other platforms.

- **Operations**

  In this section of the **Action** dialog you are configuring the functionality of the action you are editing. An action has one or mode operation modes. The evaluation of an XPath expression activates an operation mode. The first enabled operation mode is activated when you trigger the action. The scope of the XPath expression must consist only of element nodes and attribute nodes of the edited document. Otherwise, the XPath expression does not return a match and does not fire the action. More details here:*Action Mode Activation using XPath Expressions* on page 734.

  The following options are available in this section:

  - **When this XPath expression is true** - an XPath 2.0 expression that applies to elements and attributes. More details here:*Action Mode Activation using XPath Expressions* on page 734.
  - **invoke the operation** - specifies the invoked operation;
  - **with the arguments** - specifies the arguments of the invoked operation;
  - **Edit** - allows you to edit the arguments of the operation.

- **Operation priority** - increases or decreases the priority of an operation. The operations are invoked in the order of their priority. In case more than one XPath expression is true, the operation with the highest priority is invoked;
- **Add** - adds an operation;
- **Remove** - removes an operation;
- **Duplicate** - duplicates an operation.

Action Mode Activation using XPath Expressions

An Author extension action can have multiple modes, each mode invoking an Author Operation with certain configured parameters. Each action mode has an XPath 2.0 expression for activating it.

For each action mode the application will check if the XPath expression is fulfilled (when it returns a not empty nodes set or a *true* result). If it is fulfilled, the operation defined in the action mode will be executed.

Two special XPath extension functions are provided: the `oxy:allows-child-element()` function that you can use to check whether an element is valid in the current context, considering the associated schema and the `oxy:current-selected-element()` function that you can use to get the currently selected element.

The `oxy:allows-child-element()` Function

This extension function allows author actions to be available in a context only if the associated schema permits it.

The `oxy:allows-child-element()` is evaluated at the caret position and has the following signature: `oxy:allows-child-element($childName, ($attributeName, $defaultAttributeValue, $contains?)?)`.

The following parameters are supported:

childName

the name of the element that you want to check whether it is valid in the current context. Its value is a string that supports the following forms:

- the child element with the specified local name that belongs to the default namespace.

  ```
  oxy:allows-child-element("para")
  ```

  The above example verifies if the para element (of the default namespace) is allowed in the current context.
- the child element with the local name specified by any namespace.

  ```
  oxy:allows-child-element("*:para")
  ```

  The above example verifies if the para element (of any namespace) is allowed in the current context.
- a qualified name of an element.

  ```
  oxy:allows-child-element("prefix:para")
  ```

The prefix is resolved in the context of the element where the caret is located. The function matches on the element with the para local name from the previous resolved namespace. In case the prefix is not resolved to a namespace, the function returns `false`.

- any element.

```
oxy:allows-child-element("*")
```

The above function verifies if any element is allowed in the current context.

> 📄 **Note:** A common use case of `oxy:allows-child-element("*")` is in combination with the `attributeName` parameter.

attributeName

the attribute of an element that you want to check whether it is valid in the current context. Its value is a string that supports the following forms:

- the attribute with the specified name from no namespace.

```
oxy:allows-child-element("*", "class", " topic/topic ")
```

The above example verifies if an element with the `class` attribute and the default value of this attribute (that contains the `topic/topic` string) is allowed in the current context.

- the attribute with the local name specified by any namespace.

```
oxy:allows-child-element("*", "*:localname", " topic/topic ")
```

- a qualified name of an attribute.

```
oxy:allows-child-element("*", "prefix:localname", " topic/topic ")
```

The prefix is resolved in the context of the element where the caret is located. In case the prefix is not resolved to a namespace, the function returns `false`.

defaultAttributeValue

a string that represents the default value of the attribute. Depending on the value of the next parameter the default value of the attribute must either contain this value or be equal with it.

contains

an optional boolean. The default value is `true`. For the `true` value, the default value of the attribute must contain the `defaultAttributeValue` parameter. In case the value is `false`, the two values must be the same.

The `oxy:current-selected-element()` Function

This function returns the fully selected element. In case no element is selected, the function returns an empty sequence.

```
oxy:current-selected-element()[self::p]/b
```

This example returns the `b` elements that are children of the currently selected `p` element.

Menu

In the **Menu** sub-tab you configure what *framework* specific actions appear in the Oxygen XML Author menu. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the Oxygen XML Author menu. To add an action in this section as a sibling of the currently selected action, use the 🔲 **Add as sibling** button. To add an image in this section as a child of the currently selected action use the 🔲 **Add as child** button.

The following actions are available in the **Current actions** section:

Edit

Edits an item.

Remove

Removes an item.

Move Up

Moves an item up.

Move Down

Moves an item down.

Contextual menu

In the **Contextual menu** sub-tab you configure what framework-specific action the **Content Completion Assistant** proposes. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the contextual menu of a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the **Add as sibling** button. To add an action in this section as a child of the currently selected action use the **Add as child** button.

The following actions are available in the **Current actions** section:

Edit

Edits an item.

Remove

Removes an item.

Move Up

Moves an item up.

Move Down

Moves an item down.

Toolbar

In the **Toolbar** sub-tab you configure what framework-specific action the Oxygen XML Author toolbar holds. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the Oxygen XML Author toolbar when you work with a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the **Add as sibling** button. To add an action in this section as a child of the currently selected action use the **Add as child** button.

The following actions are available in the **Current actions** section:

Edit

Edits an item.

Remove

Removes an item.

Move Up

Moves an item up.

Move Down

Moves an item down.

Content Completion

In the **Content Completion** sub-tab you configure what framework-specific the **Content Completion Assistant** proposes. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that the **Content Completion Assistant** proposes when you work with a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the ⯗ **Add as sibling** button. To add an action in this section as a child of the currently selected action use the ⌐ **Add as child** button.

The following actions are available in the **Current actions** section:

Edit

Edits an item.

Remove

Removes an item.

Move Up

Moves an item up.

Move Down

Moves an item down.

The Templates Tab

The **Templates** tab specifies a list of directories in which new file templates are located. These file templates are gathered from all the document types and presented in the **New** dialog wizard.

The Catalogs Tab

The **Catalogs** tab specifies a list of *XML catalogs* which are added to all the catalogs that Oxygen XML Author uses to resolve resources.

The Transformation Tab

In the **Transformation** tab you configure the transformation scenarios associated with the framework you are editing. These are the transformation scenarios that are presented in the **Configure Transformation Scenarios** dialog as associated with the type of the edited document.

You can set one or more of the scenarios from the **Transformation** tab as default. The scenarios set here as default are rendered bold in the **Configure Transformation Scenarios** dialog and are also displayed on the tooltip of the **Apply transformation Scenario(s)**.

The **Transformation** tab offers the following options:

New

Opens the **document type rule** dialog allowing you to create *association rules*.

Edit

Opens the **document type rule** dialog allowing you to edit the properties of the currently selected *association rule*.

Delete

Deletes the currently selected *association rules*.

Import scenarios

Imports transformation scenarios.

Export selected scenarios

Export transformation scenarios.

Move Up

Moves the selection to the previous *association rule*.

⬇ **Move Down**

Moves the selection to the following *association rule*.

The Validation Tab

In the **Validation** tab you configure the validation scenarios associated with the framework you are editing. These are the validation scenarios that are presented in the **Configure Validation Scenarios** dialog as associated with the type of the edited document.

You can set one or more of the scenarios from the **Validation** tab as default. The scenarios set here as default are rendered bold in the **Configure Transformation Scenarios** dialog and are also displayed on the tooltip of the **Apply transformation Scenario(s)** button.

The **Validation** tab offers the following options:

✚ **New**

Opens the **document type rule** dialog allowing you to create *association rules*.

🔧 **Edit**

Opens the **document type rule** dialog allowing you to edit the properties of the currently selected *association rule*.

✖ **Delete**

Deletes the currently selected *association rules*.

⬆ **Import scenarios**

Imports transformation scenarios.

⬆ **Export selected scenarios**

Export transformation scenarios.

⬆ **Move Up**

Moves the selection to the previous *association rule*.

⬇ **Move Down**

Moves the selection to the following *association rule*.

The Extensions Tab

The **Extension** tab specifies implementations of Java interfaces used to provide advanced functionality to the document type.

Libraries containing the implementations must be present in the *classpath* of your document type. The Javadoc available at *http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/* contains details about how each API implementation functions.

Document Type Sharing

Oxygen XML Author allows you to share the customizations for a specific XML type by creating your own *Document Type* in the **Document Type Association** preferences page.

A document type can be shared between authors as follows:

• Save it externally in a separate framework folder in the `[OXYGEN_DIR]/frameworks` directory.

> ⚠ **Important:** For this approach to work, have the application installed to a folder with full write access.

Please follow these steps:

1. Go to `[OXYGEN_DIR]/frameworks` and create a directory for your new framework (name it for example `custom_framework`). This directory will contain resources for your framework (CSS files, new file templates, schemas used for validation, catalogs). See the **Docbook** framework structure from the `[OXYGEN_DIR]/frameworks/docbook` as an example.
2. Create your custom document type and save it externally, in the `custom_framework` directory.

3. Configure the custom document type according to your needs, take special care to make all file references relative to the [OXYGEN_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_DIR]/frameworks/custom_framework/custom.framework after the Preferences are saved.

5. Then, to share the new framework directory with other users, have them copy it to their [OXYGEN_DIR]/frameworks directory. The new document type will be available in the list of Document Types when Oxygen XML Author starts.

> 📝 **Note:** In case you have a frameworks directory stored on your local drive, you can also go to the **Document Type Association** > **Locations** preferences page and add your frameworks directory in the **Additional frameworks directories** list.

- Save the document type at project level in the **Document Type Association** preferences page.

  Please see the following steps:

  1. On your local drive, create a directory with full write access, containing the Oxygen XML Author project file and associated document type resources (CSS files, new file templates, schemas used for validation, catalogs).
  2. Start Oxygen XML Author, go to the ***Project view*** and create a project. Save it in the newly created directory.
  3. In the **Document Type Association** preferences page, select **Project Options** at the bottom of the page.
  4. Create your custom document type using the default **internal** storage for it. It will actually be saved in the previously chosen Oxygen XML Author 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. When they open the customized project file in the ***Project view***, the new document type becomes available in the list of Document Types.

- Deploy your document type configuration *as an add-on*.

How to Pack and Deploy an Add-on

Packing a Plugin or Framework as an Add-on

This procedure is suitable for developers who want a better control over the add-on package or those who want to automate some of the steps:

1. Pack the plugin or framework as a ZIP file or a *JAR*. Please note that you should pack the entire root directory not just its contents.
2. Digitally sign the package. Please note that you can perform this step only if you have created a *JAR* at the previous step. You will need a certificate signed by a trusted authority. To sign the jar you can either use the jarsigner command line tool inside Oracle's Java Development Kit. ([JDK_DIR]/bin/jarsigner.exe) or, if you are working with *Ant*, you can use the signjar task (which is just a front for the jarsigner command line tool).

> 📝 **Note:** The benefit of having a signed add-on is that the user can verify the integrity of the add-on issuer. If you don't have such a certificate you can generate one yourself using the keytool command line tool. Please note that this approach is mostly recommended for tests since anyone can create a self signed certificate.

3. Create a descriptor file. You can use a template that Oxygen XML Author provides. To use this template, go to **File** > **New** and select the **Oxygen add-ons update site** template. Once deployed, this descriptor file is referred as *update site*.

Alternatively, you can use the **Add-ons Packager** plugin by following this procedure:

1. Install the **Add-ons Packager** plugin from *http://www.oxygenxml.com/InstData/Addons/optional/updateSite.xml* as described in the *Installing Add-ons procedure*.

2. Restart Oxygen XML Author. If the add-on is correctly installed, the **Add-ons packager** toolbar action is available.
3. Invoke the **Add-ons packager** toolbar action and input the required information in the displayed dialog box.
4. Press **OK** to complete the packaging process.

Deploying an Add-on

To deploy an add-on, copy the ZIP/*JAR* file and the descriptor file to an HTTP server. The URL to this location serves as the *Update Site URL.*

Localizing Frameworks

Oxygen XML Author supports framework localization (translating framework actions, buttons, and menu entries to different languages). This lets you develop and distribute a framework to users that speak different languages without changing the distributed framework. Changing the language used in Oxygen XML Author in the Global preferences page is enough to set the right language for each framework.

To localize the content of a framework, create a translation.xml file which contains all the translation (key, value) mappings. The translation.xml has the following format:

```
<translation>
    <languageList>
        <language description="English" lang="en_US"/>
        <language description="German" lang="de_DE"/>
        <language description="French" lang="fr_FR"/>
    </languageList>
    <key value="list">
        <comment>List menu item name.</comment>
        <val lang="en_US">List</val>
        <val lang="de_DE">Liste</val>
        <val lang="fr_FR">Liste</val>
    </key>
....................
</translation>
```

Oxygen XML Author matches the GUI language with the language set in the translation.xml file. In case this language is not found, the first available language declared in the languagelist tag for the corresponding framework is used.

Add the directory where this file is located to the **Classpath** list corresponding to the edited document type.

After you create this file, you are able to use the keys defined in it to customize the name and description of:

- framework actions
- menu entries
- contextual menus
- toolbar
- static CSS content

For example, if you want to localize the bold action *open the **Preferences** dialog* and go to **Document Type Association**. Open the **Document type** dialog, go to **Author** > **Actions**, and rename the bold action to ${i18n(translation_key)}. Actions with a name format different than ${i18n(translation_key)} are not localized. Translation_key corresponds to the key from the translation.xml file.

Now open the translation.xml file and edit the translation entry if it exists or create one if it does not exist. This example presents an entry in the translation.xml file:

```
<key value="translation_key">
        <comment>Bold action name.</comment>
        <val lang="en_US">Bold</val>
        <val lang="de_DE">Bold</val>
        <val lang="fr_FR">Bold</val>
    </key>
```

To use a description from the translation.xml file in the Java code used by your custom framework, use the new *ro.sync.ecss.extensions.api.AuthorAccess.getAuthorResourceBundle()* API method to request for a certain key the associated value. In this way all the dialogs that you present from your custom operations can have labels translated in different languages.

You can also refer a key directly in the CSS content:

```
title:before{
    content:"${i18n(title.key)} : ";
}
```

📄 **Note:** You can enter any language you want in the `languagelist` tag and any number of keys.

The `translation.xml` file for the DocBook framework is located
here:`[OXYGEN_DIR]/frameworks/docbook/i18n/translation.xml`. In the **Classpath**
list corresponding to the DocBook document type the following entry was added:
`${framework}/i18n/`.

To see how the DocBook actions are defined to use these keys for their name and description, *open the Preferences dialog* and go to **Document Type Association** > **Author** > **Actions**, you can If you
look in the Java class
`ro.sync.ecss.extensions.docbook.table.SADocbookTableCustomizerDialog`
available in the oxygen-sample-framework module of the *Oxygen SDK* Maven archetype, you can see
how the new `ro.sync.ecss.extensions.api.AuthorResourceBundle` API is used to
retrieve localized descriptions for different keys.

Perspectives Layout Preferences

Oxygen XML Author offers many views that you can arrange in different layouts to suit your needs. Use the **Perspectives Layout** preferences page to configure your preferred layout.

To configure the perspectives layout options, *open the Preferences dialog* and go to **Perspectives Layout**. The following options are available:

- **Select application layout**

    - **Default** - Use the default layout for all perspectives. Any modification of this layout (for instance closing / displaying views or a new view arrangement) is saved on exit and reloaded at start-up.
    - **Predefined** - Use one of the predefined layouts:

        - **Advanced** - all views are shown.
        - **Author** - an authoring-oriented layout, displaying the **Project**, **Archive Browser**, **DITA Maps Manager**, **Outline**, **Attributes**, **Model**, and **Elements** views.
        - **Basic** - only the **Project** view and the **Outline** view are visible. Recommended when you edit XML content and you need maximum screen space.
        - **Intermediate** - the **Project**, **Outline**, **Attributes**, **Model**, **Elements**, **Entities**, and **Transformation Scenarios** views are visible.
        - **Schema development** - the **Project**, **Component Dependencies**, **Resource Hierarchy/Dependencies**, **Outline**, **Palette**, and **Attributes** views are shown.
        - **XQuery development** - the **Project**, **Outline**, **Transformation Scenarios**, **XSLT/XQuery input** views are shown.
        - **XSLT development** - the **Project**, **Component Dependencies**, **Resource Hierarchy/Dependencies**, **Outline**, **Attributes**, **Model**, **XSLT/XQuery input**, **XPath Builder**, and **Transformation Scenarios** views are shown.
    - **Custom** - A custom layout is used. You can save your preferred layout using **Window** > **Save layout ...**, then enter the location of the saved layout file in this setting.

- **Reset layout at startup** - When this option is selected, Oxygen XML Author forgets any changes made to the layout during a session and reloads the default layout the next time it is started. This is useful when you want to keep a fixed layout from one section to another.
- **Remember layout changes for each project** - When this options is selected, Oxygen XML Author saves layouts individually for each project. When you switch projects, the layout you last used for that project is loaded automatically.

- **Allow detaching editors from main window** - When this options is selected, you can drag and drop an editor window outside of the main screen. This is useful especially when you are using two monitors and you want to view files side by side.

  > **Note:** If the main screen is maximized, you cannot drag and drop an editor outside of it.

- **View tab placement** - This option specifies whether the View tabs are located at the top or bottom of the window.
- **Editor tab placement** - This option specifies whether the Editor tabs are located at the top or bottom of the window.

The changes you make to any layout are preserved between working sessions. Also, changing to a different layout and returning to the previous one does not alter the changes you made to the first layout. The predefined *layout* files are saved in the preferences directory of Oxygen XML Author.

To watch our video demonstration about configuring the user interface of Oxygen XML Author, go to *http://oxygenxml.com/demo/Dockable_Views.html*.

Encoding Preferences

Oxygen XML Author lets you configure how character encodings are recognized when opening files and which encodings are used when saving files. To configure encoding options, *open the **Preferences** dialog* and go to **Encoding**. The following encoding options are available:

- **Fallback character encoding** - Default character encoding of non-XML type documents if their character encoding cannot be otherwise determined from other sources (like, for example, specified in the document itself, or determined by the file type).

  > **Note:** For certain document types, the following encoding detection rules are used:
  >
  > - For XML, DTD and CSS documents, Oxygen XML Author tries to collect the character encoding from the document. If no such encoding is found, then *UTF-8* is used.
  > - For JavaScript, JSON, SQL, XQuery, and RNC, the *UTF-8* encoding is used.

- **UTF-8 BOM handling** - This setting specifies how to handle the *Byte Order Mark* (BOM) when Oxygen XML Author saves a UTF-8 XML document:

  - **Don't Write** - do not save the BOM bytes. Loaded BOM bytes are ignored.
  - **Write** - save the BOM bytes.
  - **Keep** - do not alter the BOM declaration of the currently open file. This is the default option.

  > **Note:** The UTF-16 BOM is always preserved. UTF-32 documents have a *big-endian* byte order.

- **Encoding errors handling** - This setting specifies how to handle characters that cannot be represented in the character encoding that is used when the document is opened. The available options are:

  - **REPORT** - displays an error identifying the character that cannot be represented in the specified encoding. Unrecognized characters are rendered as an empty box. This is the default option.
  - **IGNORE** - the error is ignored and the character is not included in the document displayed in the editor.

    > **Attention:** If you edit and save the document, the characters that cannot be represented in the specified encoding are dropped.

  - **REPLACE** - the character is replaced with a standard replacement character. For example, if the encoding is UTF-8, the replacement character has the Unicode code FFFD, and if the encoding is ASCII, the replacement character code is 63.

Editor Preferences

Oxygen XML Author lets you configure how the editor appears. To configure the appearance of the text editor, *open the **Preferences** dialog* and go to **Editor**.

The following options are available:

- **Selection background color** - Sets the background color of selected text.
- **Selection foreground color** - Sets the text color of selected text.
- **Completion proposal background** - Sets the background color of the content completion window.
- **Completion proposal foreground** - Sets the foreground color of the content completion window.
- **Documentation window background** - Sets the background color of the documentation of elements suggested by the content completion assistant.
- **Documentation window foreground** - Sets the foreground color for the documentation of elements suggested by the content completion assistant.
- **Find highlight color** - Sets the color of the highlights generated by the **Find** and **Find all** actions.
- **XPath highlight color** - Sets the color of the highlights generated when you run an XPath expression.
- **Maximum number of highlights** - Sets the maximum number of highlights that Oxygen XML Author displays.
- **Show TAB/NBSP/EOL/EOF marks** - Makes the *TAB*/*NBSP*/*EOL*/*EOF* characters visible in the editor. You can use the color picker to choose the color of the marks.

  - **Show SPACE marks** - Makes the space character visible in the editor.

- **Can edit read only files** - If this option is selected, Oxygen XML Author will let you edit, but not save, a read only file. If the option is not selected, you cannot edit or save a read only file.
- **Display quick-assist and quick-fix side hints** - Displays the Quick Assist and Quick Fix icon in the editor's left side line number stripe. Works both in the **Text** and **Author** edit modes.
- **Undo history size** - Sets the maximum amount of undo operations you can perform in either of the editor modes (Text, Author, Design, Grid).

Print Preferences

Oxygen XML Author lets you configure how files are printed out of the editor. Note that these setting cover how files are printed directly from Oxygen XML Author itself, not how they are printed after the XML source has been transformed by a publishing stylesheet. To configure the **Print** options, *open the **Preferences** dialog* and go to **Editor** > **Print** .

This page allows you to customize the headers and footers added to a printed page when you print from the *Text* mode or *Author* mode editors. These settings do not apply to the *Grid* and schema **Design** modes.

You can specify what is printed on the **Left**, **Middle** and **Right** of the header and footer using plain text of any if the following variables:

- *${currentFileURL}* - Current file as URL, that is the absolute file path of the current edited document represented as URL.
- *${cfne}* - Current file name with extension. The current file is the one currently opened and selected.
- *${cp}* - Current page number. Used to display the current page number on each printed page in the **Editor / Print** Preferences page.
- *${tp}* - Total number of pages in the document. Used to display the total number of pages on each printed page in the **Editor / Print** Preferences page.
- *${env(VAR_NAME)}* - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the *${system(var.name)}* editor variable.
- *${system(var.name)}* - Value of the var.name Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as -Dvar.name=var.value. If you are looking for operating system environment variables, use the *${env(VAR_NAME)}* editor variable instead.
- *${date(pattern)}* - Current date. The allowed patterns are equivalent to the ones in the *Java SimpleDateFormat class*. Example: yyyy-MM-dd;

  📄 **Note:** This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to *http://www.w3.org/TR/xmlschema-2/#date*. For details about xs:datetime, go to *http://www.w3.org/TR/xmlschema-2/#dateTime*.

For example, to show the current page number and 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} of ${tp}.

You can also set the **Color** and **Font** used in the header and footer. Default font is `SansSerif`.

You can place a line below the header or above the footer by selecting **Underline/Overline**.

Edit modes Preferences

Oxygen XML Author lets you configure which *edit mode* a file is opened in the first time it is opened. This setting only affects the first time a file is opened. The current editing mode of each file is saved when the file is closed and restored the next time it is opened. To configure the **Edit modes** options, *open the Preferences dialog* and go to **Editor** > **Edit modes** .

If **Allow Document Type specific edit mode setting to override the general mode setting** is selected, the initial edit mode setting set in *the Document Type dialog* overrides the general edit mode setting from the table below.

The initial edit mode can be one of the following:

- *Text*
- *Author*
- *Grid*
- Design (available only for the W3C XML Schema editor).

The Oxygen XML Author "Edit modes" Preferences Page

Text Preferences

Oxygen XML Author lets you configure how *text mode* editor appears. To configure the Text mode editor options, *open the Preferences dialog* and go to **Editor** > **Edit modes** > **Text** .

The following preferences are available:

- **Editor background color** - Sets the background color of the **Text** editing mode, **Diff Files** editors and **Outline** view.
- **Editor caret color** - Sets the color of the caret.
- **Highlight current line** - Sets the foreground color of the line numbers displayed in the editor panels.
- **Show line numbers** - Shows line numbers in the editor panels and in the **Results** view of the Debugger perspective.
- **Show print margin** - In conjunction with the **Print margin column** option, allows you to set a safe print limit in the form of a vertical line displayed in the right side of the editor pane. You can also customize the print margin line color.

    - **Print margin column** - Safe print limit width measured in characters.

- **Line wrap** - Enables *soft wrap* of long lines, that is automatically wrap lines in edited documents. The document content is unaltered as the application does not use newline characters to break long lines.

    Note: When you enable the **Line wrap** option, Oxygen XML Author disables the **Highlight current line** option.

- **Cut / Copy whole line when nothing is selected** - Enables the *Cut* and *Copy* actions when nothing is selected in the editor. In this case the *Cut* and *Copy* actions operate on the entire current line.
- **Enable folding** - Displays the vertical stripe that holds the *folding markers*.
- **Highlight matching tag** - If you place the cursor on a start or end tag, Oxygen XML Author highlights the corresponding member of the pair. You can also customize the highlight color.
- **Lock the XML tags** - Default tag locking state of the opened editors. For more information, see the *Locking and Unlocking XML markup* section.

Grid Preferences

Oxygen XML Author provides a *Grid view* of an XML document. To configure the **Grid** options, *open the **Preferences dialog*** and go to **Editor** > **Edit modes** > **Grid**.

The following options are available:

- **Compact representation** - If selected, the *compact representation* of the grid is used: a child element is displayed beside the parent element. In the *non-compact representation*, a child element is nested below the parent.
- **Format and indent when passing from grid to text or on save** - If selected, the content of the document is *formatted and indented* each time you switch from the **Grid** view to the **Text** view.
- **Default column width (characters)** - Sets the default width in characters of a table column of the grid. A column can hold:
  - element names
  - element text content
  - attribute names
  - attribute values

  If the total width of the grid structure is too large you can resize any column by dragging the column margins with the mouse pointer, but the change is not persistent. To make it persistent, set the new column width with this option.
- **Active cell color** - Sets the background color for the active cell of the grid. There is only one active cell at a time. The keyboard input always goes to the active cell and the selection always contains it.
- **Selection color** - Background color for the selected cells of the grid except the active cell.
- **Border color** - The color used for the lines that separate the grid cells.
- **Background color** - The background color of grid cells that are not selected.
- **Foreground color** - The text color of the information displayed in the grid cells.
- **Row header colors - Background color** - The background color of row headers that are not selected.
- **Row header colors - Active cell color** - The background color of the row header cell that is currently active.
- **Row header colors - Selection color** - The background color of the header cells corresponding to the currently selected rows.
- **Column header colors - Background color** - The background color of column headers that are not selected.
- **Column header colors - Active cell color** - The background color of the column header cell that is currently active.
- **Column header colors - Selection color** - The background color of the header cells corresponding to the currently selected columns.

The column headers are painted with two color gradients, one for the upper 1/3 part of the header and the other for the lower 2/3 part. The start and end colors of the first gradient are set with the first two color buttons. The start and end colors of the second gradient are set with the last two color buttons.

Author Preferences

Oxygen XML Author provides and *author mode* editor which provides a configurable graphical editing interface to XML documents. To configure the **Author** mode preferences, *open the **Preferences dialog*** and go to **Editor** > **Edit modes** > **Author**.

The following options are available:

- **Author default background color** -Sets the default background color of the Author editing mode. The `background-color` property set in the CSS file associated with the current edited document overwrites this option.

- **Author default foreground color** - Sets the default foreground color of the Author editing mode. The `color` property set in the CSS file associated with the current edited document overwrites this option.
- **Show placeholders for empty elements** - When this option is selected, placeholders are displayed for elements with no content to make them clearly visible. The placeholder is rendered as a light grey box displaying the element name.
- **Show Author layout messages** - When this option is selected, all errors reported during layout creation are presented in the **Errors** view.
- **Show block range** - When this option is selected, a *block range indicator* is shown in a stripe located in the left side of the editor. The block range indicator is displayed as a heavy line that spans from the first line to the last line of the block.
- **Show XML comments** - When this option is selected, XML comments are displayed in **Author** mode, otherwise they are hidden.
- **Show processing instructions** - When this option is selected, XML processing instructions are displayed in **Author** mode, otherwise they are hidden.
- **Show doctype** - When this option is selected, the *doctype* declaration is displayed in **Author** mode;, otherwise it is hidden.
- **Display referred content (e.g.: entities, XInclude, DITA conref, etc.)** - when enabled, the references (like entities, XInclude, DITA conref) also display the content of the resources they refer. If you toggle this option while editing, reload the file for the modification to take effect.
- **Auto-scale images wider than (pixels)** - Sets the maximum width at which an image will be displayed. Wider images will be scaled to fit.

  - **Show very large images** - When this option is selected, images larger than 6 megapixels are displayed in **Author** mode, otherwise they are not displayed.

    ⚠ **Important:** If you enable this option and your document contains many such images, Oxygen XML Author may consume all available memory, throwing an *OutOfMemory error*. To resole this, increase the *available memory limit.* and restart the application.

- **Format and indent** - here you can set the format and indent method that is applied when a document is saved in **Author** mode, or when switching the editing mode (from Text to Author or vice versa):

  - **Only the modified content** - the **Save** operation only formats the nodes that were modified in the **Author** mode. The rest of the document preserves its original formatting.

    📝 **Note:** This option applies also to the DITA Maps open in the **DITA Maps Manager**.

  - **The entire document** - the **Save** operation applies formatting to the entire document regardless of the nodes that were modified in **Author** mode.

    If the **Apply also 'Format and Indent' action as in 'Text' edit mode** option is enabled, the content of the document is formatted by applying the **Format and Indent** rules from the *Editor/Format/XML* option page. In this case, the result of the **Format and indent** operation will be the same as when it is applied in **Text** editing mode.

- **Tags display mode** - Sets the default display mode for element tags presented in **Author** mode. You can choose between:

  - **Full Tags with Attributes** - All XML tags are shown, with attribute names and values, easing the transition from a Text based editing to an **Author** mode editing.
  - **Full Tags** - All XML tags are shown, but without attributes.
  - **Block Tags** - The XML tags that enclose block elements are shown in full. Compact tags (no element names) are shown for inline elements.
  - **Inline Tags** - The XML tags that enclose inline elements are shown in full. Block tabs are not shown.
  - **Partial Tags** - Partial tags (no names) are shown for all elements.
  - **No Tags** - No tags are displayed. This representation is as close as possible to a word-processor view.

- **Tags background color** - Sets the Author tags background color.
- **Tags foreground color** - Sets the Author tags foreground color.

Caret Navigation Preferences

Oxygen XML Author allows you to configure the appearance of the *caret* (text cursor) in the *author mode* editor. To set caret navigation preferences, *open the **Preferences** dialog* and go to **Author** > **Caret Navigation**. The following options are available:

- **Highlight elements near caret** - When this option is selected, the element containing the caret is highlighted. You can use the color picker to choose the color of the highlight.
- **Show caret position tooltip** - Oxygen XML Author uses *tool tips in Author mode to indicate the position of the caret in the element structure* of the underlying document. Depending on context, the tool tips may show the current element name or the names of the elements before and after the current caret position.
- **Show location tooltip on mouse move** - When this option is selected, Oxygen XML Author displays *Location Tooltips* when you are editing the document in certain tags display modes (Inline Tags, Partial Tags, No Tags) and the mouse pointer is moved between block elements.
- **Quick up/down navigation** - By default, when you navigate using the up and down arrow keys in Author mode, the caret is placed within each of the underlying XML elements between two blocks of text. (The caret turns horizontal when it is between blocks of text.) For instance, between a list item in one section and the title in a following sections, the caret might stop several times in the underlying structure: the list item, the list, the paragraph, the section, and the root element between sections, the new section, and finally in the title. Any one of these location is a place you might want to insert new content. When this option is selected, however, the caret does not stop at these positions, but jumps from one text line to another, similar to how the caret behaves in a word processor.
- **Arrow keys move the caret in the writing direction** - This setting determines how the left and right arrow keys behave in Author mode for bidirectional (BIDI) text. When this option is selected, the right arrow key advances the caret in the reading direction. When this option is not selected, pressing the right arrow will simply move the caret to the right, regardless of the text direction.

Schema Aware Preferences

Oxygen XML Author can use the schema of your XML language to improve the way the *Author mode* editor handles your content. To configure the **Schema Aware** options, *open the **Preferences** dialog* and go to **Editor** > **Edit modes** > **Author** > **Schema aware**.

- **Schema aware normalization, format, and indent**

  When you open or save a document in Author mode, white space is normalized using the `display` property of the current CSS stylesheet, and the values of the **Preserve space elements**, **Default space elements**, and **Mixed content elements** *settings*. When this option is selected, the schema will also be used to normalize white space, based on the content model (*element-only*, *simple-content*, or *mixed*)

- **Indent blocks-only content**

  Oxygen XML Author does not normally apply indenting to the source of a element with mixed content, to avoid accidentally introducing inappropriate white space around inline elements. However, if such an element does not contain any text content, and if all the elements it contains are block elements (as specified by the CSS property `display` on those elements), there is no danger of introducing inappropriate white space. If this option is selected, Oxygen XML Author will apply indenting to the source of these elements.

- **Schema Aware Editing**

  This setting determines how Oxygen XML Author will use the schema of a document to control the behavior of the author-mode editor.

  - **On** - Enables all schema- aware editing options.
  - **Off** - Disables all schema-aware editing options.
  - **Custom** - Allows you to select the schema-aware editing options from the following:

    - **Delete element tags with backspace and delete**

      Controls what happens when you attempt to delete an element tag. The options are:

- **Smart delete**

  If deleting the tag would make the document invalid, Oxygen XML Author will attempt to make the document valid by unwrapping the current element or by appending it to an adjacent element where the result would be valid. For instance, if you delete a bold tag, the content can be unwrapped and become part of the surrounding paragraph, but if you delete an list item tag, the list item content cannot become part of the list container. However, the content could be appended to a preceding list items.

- **Reject action when its result is invalid**

  Deletion that would leave the document in an invalid state is rejected.

- **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, Oxygen XML Author tries to find an appropriate insert position. Possibilities include:

    - Creating a sibling element that can accept the content. (For example, if you tried to paste a paragraph into an existing paragraph.)
    - Inserting the content into a parent or child element. (For example, if you tried to paste a list item into an existing list item, or into the space above or below and existing list.)
    - Inserting the content into an ancestor element where it would be valid.

  - **Reject action when its result is invalid**

    When selected, Oxygen XML Author will not let you paste content into a position where it would be invalid.

- **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 are:

  - **Allow only insertion of valid elements and attributes**

    If selected, the content completion list shows only the elements that can be inserted at the current position and will not allow you to enter any other element.

  - **Show all possible elements in the content completion list**

    If selected, the content completion list will show all the elements in the schema, even those that cannot be entered validly at the current position. If you select an element that is not valid at the current position, Oxygen XML Author will attempt to find a valid location to insert it and may present you with several options.

- **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 selected, a warning message will be displayed if the *Delete Element Tags* action will result in an invalid document. You will be asked to confirm the deletion.

- **Join Elements**

    If selected, a warning message will be displayed if the *Join Elements* action will result in an invalid document. You will be asked to confirm the join.

- **Convert external content on paste**

    If selected, turns on *smart paste*.

If the Schema Aware Editing is **On** or **Custom** all actions that can generate invalid content will be forwarded first toward *AuthorSchemaAwareEditingHandler*.

Review Preferences

Oxygen XML Author lets you *enter review comments and track changes* in your documents. The Review preferences control how the Oxygen XML Author review features work. To configure the **Review** options, *open the Preferences dialog* and go to **Editor** > **Edit modes** > **Author** > **Review**.

The available options are:

- **Author** - Specifies the name to be attached to all comments and to changes made while **Track Changes** is active. By default, Oxygen XML Author uses the system user name.
- **Initial State** - Track Changes is turned on and off separately for each document you open. You may have some opened documents for which track changes is turned on and others for which it is turned off. This setting specifies whether **Track Changes** is turned on when you open a document. The options are:

    - **Stored in document** - The current state of track changed is stored in the document itself, meaning that track changes on or off depending on the state the last time the document was saved. This is the recommended setting when multiple authors work on the same set of documents as it will make it obvious to other authors that changes have been made in the document.
    - **Always On** - The **Track Changes** feature is always on when you open a document. You can turn it off for an open document, but it will be turned on for the next document you open.
    - **Always Off** - The **Track Changes** feature is always off when you open a document. You can turn it on for an open document, but it will be turned off for the next document you open.

- **Display changed lines marker** - A changed line maker is a vertical line on the left side of the editor window indicating where changes have been made in the document. To hide the changed lines marker, deselect this option.
- **Inserted content color** - When **Track Changes** option is on, the newly inserted content is highlighted with an *insertion marker*, that uses a color to adjust the following display properties of the inserted content: foreground, background, and underline. This section allows you to customize the marker's color:

    - **Automatic** - Oxygen XML Author assigns a color to each user who inserted content in the current document. The colors are picked from the **Colors for automatic assignment** list, the priority being established by the change (deletion, insertion, or comment) timestamp.
    - **Fixed** - Uses the specified color for all insertion markers, regardless of who the author is.
    - **Use same color for text foreground** - Use the color defined above (**Automatic** or **Fixed**) to render the foreground of the inserted content.
    - **Use same color for background** - Use the color defined above (**Automatic** or **Fixed**) to render the background of the inserted content. A slider control allows you to set the transparency level of the marker's background.

- **Deleted content color** - When **Track Changes** option is on, the deleted content is highlighted with a *deletion marker*, that uses a color to adjust the following display properties of the deleted content: foreground, background, and strikethrough. This section allows you to customize the marker's color:

  - **Automatic** - Oxygen XML Author assigns a color to each user who deleted content in the current document. The colors are picked from the **Colors for automatic assignment** list, the priority being established by the change (deletion, insertion, or comment) timestamp.
  - **Fixed** - Uses the specified color for all deletion markers, regardless of who the author is.
  - **Use same color for text foreground** - Use the color defined above (**Automatic** or **Fixed**) to render the foreground of the deleted content.
  - **Use same color for background** - Use the color defined above (**Automatic** or **Fixed**) to render the background of the deleted content. A slider control allows you to set the transparency level of the marker's background.

- **Comments color (applies for all authors)** - Sets the background color of the text that is commented on. The options are:

  - **Automatic** - Oxygen XML Author assigns a color to each user who added a comment in the current document. The colors are picked from the **Colors for automatic assignment** list, the priority being established by the change (deletion, insertion, or comment) timestamp.
  - **Fixed** - Uses the specified color for all changes, regardless of the author's name. Use the slider to control the transparency level.

Callouts Preferences

Oxygen XML Author can display callouts for *review items such as comment, insertion, and deletions*. To set review callouts preferences, *open the **Preferences** dialog* and go to **Editor** > **Edit modes** > **Author** > **Review** > **Callouts**.

The options are:

- **Comments** - If selected, callouts are shown for comments. This option is enabled by default.
- **Track Changes deletion** - If selected, callouts are shown for deletions.

  - **Show deleted content in callout** - If selected, the deleted content is shown in the callout.

- **Track Changes insertion** - If selected, callouts are shown for insertions.

  - **Show inserted content in callout** - If selected, the inserted content is shown in the callout.

- **Show review time** - When selected, the date and time of a change are shown in the callout.
- **Show all connecting lines** - When selected, lines connect the callout to the location of the change.
- **Callouts pane width (px)** - Sets the width of the callout field. The default is 200 pixels.
- **Callouts text limit (characters)** - Sets the number of characters shown in the callout. The default is 160. Note that this does not limit the number of characters in a comment. I only limits the number of characters shown in the callout. To see the full comment, see the *review view*.

Profiling / Conditional Text Preferences

Oxygen XML Author lets you configure how *profiling and conditional text* is shown in Author view. It has built in support for the standard conditional text features of DITA and DocBook, which you can customize for your own projects. You can also add conditional support for other XML vocabularies, including your custom vocabularies.

To configure **Profiling/Conditional Text** options, *open the **Preferences** dialog* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text**.

- **Note:** Please note the following when configuring this setting:

  - This setting is used to define how profiled elements are treated in Author view. It does not create profiling or conditional text attributes or values in the underlying XML vocabulary. It just changes how the editor displays them.
  - This setting should be used for profiling / conditional text elements only. To change how other types of attributes are displayed in the text, use a CSS style sheet.

- If you are using the DITA XML vocabulary and a DITA Subject Scheme Map is defined in the root map of your document, it will be used in place of anything defined using this dialog.

This preferences page contains two items:

- **profiling attributes**, which allow to specify a set of allowable value for each profiling or conditional attribute.
- **profiling condition sets** which allow you to specify a specific set of profiling attributes to be used to specify a particular build configuration for your content.

If you have two or more identically named entries that match the same document type, Oxygen XML Author uses the one that is positioned highest in the table. Use the **Up / Down** buttons to change the priority of the entries.

The **Import from DITAVAL** button allows you to import profiling attributes from .ditaval files. You can merge these new profiling attributes with the existing ones, or replace them completely. If the imported attributes conflict with the existing ones, Oxygen XML Author displays a dialog box containing two tables: the first one previews the imported attributes and the second one previews the already defined attributes. You can choose either to keep the existing attributes or replace them with the imported ones.

> **Note:** When importing profiling attributes from DITAVAL files, Oxygen XML Author automatically creates condition sets based on these files.

Colors and Styles Preferences

Oxygen XML Author lets you set the colors and styles used to display *profiling / conditional text* in the *Author mode editor*. To set Colors and Styles preferences, *open the **Preferences** dialog* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text** > **Colors and Styles**.

The central area of the page contains a table that lists two categories of profiling styles:

- **Defined attributes values** - contains the styles for profiling attribute values defined in the *Profiling / Conditional Text* preferences page. Each profiling attribute value has an associated style. To ease the process of customizing styles, the **Defined attributes values** category contains by default the list of empty styles. All you have to do is to adjust the colors and decorations, thus skipping the process of manually defining the association rules (document type, attribute name and value). This is the reason why a style from this category can only be *reset*, not deleted.
- **Other** - this category contains styles for attribute values that are not marked as profiling values, in the *Profiling / Conditional Text* preferences page. In this category are listed:
  - all the styles that were defined in other projects (with different profiling attribute value sets)
  - all the styles set for the profiling attributes defined in a *subject scheme map*

Adding a profiling style

To add profiling styles use one of the following actions:

Import from DITAVAL...

Allows you to import profiling styles from .ditaval files. You can merge these new profiling styles with the existing ones, or replace them completely. If the imported styles conflict with the existing ones, Oxygen XML Author displays a dialog box containing two tables: the first one previews the imported styles and the second one previews the already defined styles. You can choose either to keep the existing styles or replace them with the imported ones.

Automatic styling

For every profiling attribute value that has no style defined, applies one of the following color or style: background, foreground, text decoration (underline, overline, double-underline) or text style (bold, italic).

New

Opens the **Add Profiling Style** dialog that allows you to associate a set of coloring and styling properties to a profiling value.

> **Note:** You can define a default style for a specific attribute by setting the **Attribute value** field to <ANY>. This style is applied for attribute values that do not have a specific style associated with it.

Modify a profiling style

To modify an already defined style use one of the following actions:

- double click the style in the styles table to open the **Edit Profiling Style** dialog
- select the style in the styles table and press **Edit** to open the **Edit Profiling Style** dialog

Resetting a profiling style from Defined attributes values category

To reset a style from the **Defined attributes values** category to its default (no color or decoration), select it and click the **Clear style** button.

Deleting a profiling style from Other category

To delete a style from the **Other** category, select it and click the **Delete** button.

Attributes Rendering Preferences

Oxygen XML Author lets you *display the profiling attributes applied to your content* in the Author mode editor. To configure how the profiling attributes appear, *open the **Preferences** dialog* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text** > **Attributes Rendering**. When the ***Show Profiling Attributes*** option is turned on, the Author page displays conditional text markers at the end of conditional text blocks. Use the options in this page to customize the rendering of these text markers.

You can set the following options:

- **Show profiling attribute name** - If checked, the names of the profiling attributes are shown with their values. If unchecked, only the values are shown.
- **Background color** - Sets the background color used to display the profiling attributes.
- **Attribute name foreground color** - Sets the foreground color used to display the names of the profiling attributes.
- **Attribute values foreground color** - Sets the foreground color used to display values of the profiling attributes.
- **Border color** - Sets the color of the border of the block that displays the profiling attributes.

Messages Preferences

Oxygen XML Author can open a DITA map in the regular editor view or in the *DITA Maps Manager*. You can select whether you want to always open DITA maps in the editor or in the DITA Maps Manger, or ask you each time. To set your preference, *open the **Preferences** dialog* and go to **Editor** > **Edit Modes** > **Author** > **Messages**. The following options are available:

- **Always open in the DITA Maps Manager** - A DITA Map file is always opened in the **DITA Maps Manager** view.
- **Always open as XML** - A DITA Map file is always opened in the XML editor.
- **Always ask** - When you open a DITA Map, you are prompted to choose between opening it in the XML editor panel or in the **DITA Maps Manager** view.

MathML Preferences

Oxygen XML Author allows you to *edit MathML* equations and see the results in a preview window. For a more specialized MathML editor, you can *install MathFlow*.

To configure the *MathML* editor or to enter your MathFlow license information, *open the **Preferences** dialog* and go to **Editor** > **Edit modes** > **Author** > **MathML**. You can configure the following parameters:

- **Equation minimum font size** - The minimum size of the font used for rendering mathematical symbols.
- **MathFlow installation directory** - The installation folder whereOxygen XML Author looks for *MathFlow* Components (the *MathFlow* SDK).
- **MathFlow license file** - The license for*MathFlow* Components (the *MathFlow* SDK).
- **MathFlow preferred editor** - A *MathML* formula can be edited in one of three editors of the *MathFlow* Components (*MathFlow* SDK): structure editor, style editor, and simple editor. More documentation is available on the *MathFlow SDK* website.

AutoCorrect Preferences

Oxygen XML Author can convert double and single quotes to the quotation characters of your choice as you type. To turn on smart quotes and select the quotation marks to use, *open the **Preferences** dialog* and go to **Editor** > **Edit Modes** > **Author** > **AutoCorrect**.

The following options are available:

- **Replace "double quotes" with "smart quotes"** - The selected open and close quote characters as inserted when you type double quotes.
- **Replace "single quotes" with "smart quotes"** - The selected open and close quote characters as inserted when you type single quotes.

To change the quotation marks that are inserted, click the **Start quote** and **End quote** buttons.

Format Preferences

Oxygen XML Author lets you control how the text of an XML document is formatted when you use the Format and Indent option in the *Text mode* editor. To configure the **Format** options, *open the **Preferences** dialog* and go to **Editor** > **Format**.

> **Note:** These settings apply to the Text mode editor. To control how the *Author mode* editor formats the underlying XML document, see *Author Whitespace Handling* on page 110.

> **Note:** These settings apply to the formatting of XML source documents, not the formatting of output documents. The formatting of output documents is determined by the *transformations that create them*.

The following options are available:

- **Detect indent on open** - The editor detects how a document is indented when it is opened. Choose this setting if you edit XML documents that are indented in different ways and want the formatting of each to be preserved as closely as possible. Additionally, 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 your files are stored in a version control system.

  > **Note:** If the document contains different-size indents, the application computes a weighted average value to used for new indents.

  - **Use zero-indent, if detected** - the formatting operation takes into account an indent level of zero in case this is detected.
- **Indent with tabs** - If selected, indents are created using tab characters. If unchecked, indents are formed with spaces. The number of space used is set by the **Indent size** option.
- **Indent size** - The number of spaces used to create an indent.
- **Hard line wrap** - When selected, Oxygen XML Author breaks the edited line automatically when its length exceeds the limit specified by the **Line width - Format and Indent**, or the length that was detected on open (if **Detect line width on open** is selected).
- **Indent on Enter** - When selected, Oxygen XML Author indents the new line when you press the Enter key.
- **Enable Smart Enter** - When selected, if you press the Enter key between a start and an end tag, Oxygen XML Author places the cursor in an indented position on the empty line formed between the start and end tag.
- **Detect line width on open** - When selected, detects the line width automatically when the document is opened.
- **Format and indent the document on open** - When selected, an XML document is formatted and indented before opening it in the editor panel. This option applies only to documents associated with the XML editor. This option does not apply to *read-only* documents when the *Can edit read only files* option is disabled.
- **Line width - Format and Indent** - Defines the point at which the **Format and Indent** (pretty-print) function 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 selected, Oxygen XML Author empties the undo buffer before doing a **Format and Indent** operation. This means you will not be able to undo any changes you made before the

format and indent operation. Select this option if you encounter out of memory problems (**OutOfMemoryError**) when performing the **Format and Indent** operation.

To watch our video demonstration about the formatting options offered by Oxygen XML Author, go to *http://oxygenxml.com/demo/Autodetect_Formating.html*.

XML Formatting Preferences

To configure the **XML** Format options, *open the **Preferences** dialog* and go to **Editor** > **Format** > **XML**.

The following options are available:

- **Preserve empty lines** - The**Format and Indent** operation preserves all empty lines found in the document;
- **Preserve text as it is** - The**Format and Indent** operation preserves text content as it is, without removing or adding any white space.
- **Preserve line breaks in attributes** - Line breaks found in attribute values are preserved;

> 📋 **Note:** When this option is enabled, **Break long lines** option is automatically disabled.

- **Break long attributes** - The**Format and Indent** operation breaks long attribute values;
- **Indent inline elements** - The *inline elements* are indented on separate lines if they are preceded by white spaces and they follow another element start or end tag. Example:

  Original XML:

  ```
  <root>
    text <parent> <child></child> </parent>
  </root>
  ```

  Indent inline elements enabled:

  ```
  <root> text <parent>
      <child/>
    </parent>
  </root>
  ```

  Indent inline elements disabled:

  ```
  <root> text <parent> <child/> </parent> </root>
  ```

- **Expand empty elements** - The**Format and Indent** operation outputs empty elements with a separate closing tag, ex. `<a atr1="v1"></a>`. When not enabled, the same operation represents an empty element in a more compact form: `<a atr1="v1"/>`;
- **Sort attributes** - The**Format and Indent** operation sorts the attributes of an element alphabetically;
- **Add space before slash in empty elements** - Inserts a space character before the trailing / and > of empty elements;
- **Break line before attribute's name** - The**Format and Indent** operation breaks the line before the attribute name;
- **Element spacing** - Controls how the application handles whites paces found in XML content;

  - **Preserve space** - List of elements for which the **Format and Indent** operation preserves the whitespaces (like blanks, tabs, and newlines). The elements can be specified by name or by XPath expressions:

    - `elementName`
    - `//elementName`
    - `/elementName1/elementName2/elementName3`
    - `//xs:localName`

    The namespace prefixes like `xs` are treated as part of the element name without taking into account its binding to a namespace.
  - **Default space** - This list contains the names of the elements for which contiguous whitespaces are merged by the **Format and Indent** operation into one blank character.
  - **Mixed content** - The elements from this list are treated as mixed when applying the **Format and Indent** operation. The lines are split only when whitespaces are encountered.

- **Schema aware format and indent** - The **Format and Indent** operation takes into account the schema information regarding the *space preserve*, *mixed*, or *element only* properties of an element;
- **Indent (when typing) in preserve space elements** - The *Preserve space* elements (identified by the `xml:space` attribute set to `preserve` or by their presence in the **Preserve space** elements list) are normally ignored by the **Format and Indent** operation. When this option is enabled and you are editing one of these elements, its content is formatted.
- **Indent on paste - sections with number of lines less than 300** - When you paste a chunk of text that has less than 300 lines, the inserted content is indented. To keep the indent style of the document you are copying content from, disable this option.

Whitespaces Preferences

Oxygen XML Author lets you configure which Unicode space characters are treated as space characters when normalizing whitespace in XML documents. To configure the **Whitespace** preferences, *open the **Preferences** dialog* and go to **Editor** > **Format** > **XML** > **Whitespaces**.

This table lists the Unicode whitespace characters. Check any that you want to have treated as whitespace when formatting and indenting an XML document.

The whitespaces are normalized when

- the **Format and Indent** action is applied on an XML document;
- you switch from **Text** mode to **Author** mode;
- you switch from **Author** mode to **Text** mode;

The characters with the codes 9 (TAB), 10 (LF), 13 (CR) and 32 (SPACE) are always considered to be whitespace characters and cannot be deselected.

CSS Properties Formatting Preferences

Oxygen XML Author can format and indent your CSS files. To configure the **CSS** Format options, go to **Editor** > **Format** > **CSS**.

The following options control how your CSS files are formatted and indented:

- **Indent class content** - The *class* content is indented. Enabled by default.
- **Class body on new line** - The *class* body (including the curly brackets) is placed on a new line.
- **Add new line between classes** - An empty line is added between two classes.
- **Preserve empty lines** - The empty lines from the CSS content are preserved.
- **Allow formatting embedded CSS** - The CSS content embedded in XML is formatted when the XML content is formatted.

JavaScript Properties Formatting Preferences

To configure the **JavaScript** format options, *open the **Preferences** dialog* and go to **Editor** > **Format** > **JavaScript**.

The following options control the behavior of the **Format and Indent** action:

- **Start curly brace on new line** - opening curly braces start on a new line;
- **Preserve empty lines** - empty lines in the JavaScript code are preserved. This option is enabled by default;
- **Allow formatting embedded JavaScript** - applied only to XHTML documents, this option allows Oxygen XML Author to format embedded JavaScript code, taking precedence over the *Schema aware format and indent* option. This option is enabled by default.

Content Completion Preferences

Oxygen XML Author provides a *Content Completion Assistant* that list available options at any point in a document and can auto-complete structures, elements, and attributes. These options control how the **Content Completion Assistant** works.

To configure the **Content Completion** preferences, *open the **Preferences** dialog* and go to **Editor** > **Content Completion**.

The following options are available:

- **Auto close the last opened tag** - Oxygen XML Author closes the last open tag when you type </.
- **Automatically rename/delete/comment matching tag** - If you rename, delete, of comment out a start tag,Oxygen XML Author automatically renames, deletes, or comments out the matching end tag.

> **Note:** If you select **Toggle comment** for multiple starting tags (or you delete them) and the matching end tags area interferes with start tags, the end tags are not commented (or deleted).

- **Use content completion** - Turns content completion on or off.
- **Close the inserted element** - When you choose an entry from the **Content Completion Assistant** list of proposals, Oxygen XML Author 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 Author inserts the required elements specified in the DTD, XML Schema, or RELAX NG schema that is *associated with the edited XML document*.

    - **Add optional content** - Oxygen XML Author inserts the optional elements specified in the DTD, XML Schema, or RELAX NG schema.
    - **Add first Choice particle** - Oxygen XML Author inserts the first **choice** particle specified in the DTD, XML Schema, or RELAX NG schema.

- **Case sensitive search** - The search in the content completion assistant window when you type a character is case-sensitive ('a' and 'A' are different characters).
- **Cursor position between tags** - When selected, Oxygen XML Author moves the cursor automatically between start and end tag after inserting the element. This applies to:

  - Elements with only optional attributes or no attributes at all.
  - Elements with required attributes, but only when the **Insert the required attributes** option is disabled.

- **Show all entities** - Oxygen XML Author 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 Author 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 Author 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 Author 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 and are separated from the rest of the suggestions by a thin grey line . This option is applied also in the **Author** mode of the XML editor;
- **Maximum number of recent items shown** - limits the number of recently used items presented at the top of the **Content Completion Assistant** window. This option is applied also in the **Author** mode of the XML editor;
- **Learn attributes values** - Oxygen XML Author 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 Author 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 (Command+Space on OS X)**) - When selected, Oxygen XML Author learns the typed words and makes them available in a content completion fashion by pressing **Ctrl+Space (Command+Space on OS X)** om your keyboard;

> **Note:** In order to be learned, the words need to be separated by space characters.

- **Activation delay of the proposals window (ms)** - Delay in milliseconds from last key press until the content completion assistant window is displayed.

Annotations Preferences

Different types of schemas (XML Schema, DTDs, Relax NG) can include annotations that document the various elements and attributes that they define. Oxygen XML Author can display these annotations when offering content completion suggestions. To configure the **Annotations** preferences *open the **Preferences** dialog* and go to **Editor** > **Content Completion** > **Annotations**.

The following options are available:

- **Show annotations in Content Completion Assistant** - Oxygen XML Author displays the schema annotations of an element, attribute, or attribute value currently selected in the **Content Completion Assistant** proposals list.
- **Show annotations in tooltip** - Oxygen XML Author displays the annotation of elements and attributes as a tooltip when you hover over them with the cursor in the editing area or in the **Elements** view.
- **Show annotation in HTML format, if possible** - This option allows you to view the annotations associated with an element or attribute in HTML format. It is available when editing XML documents that have associated an XML Schema or Relax NG schema. When this option is disabled the annotations are converted and displayed as plain text.
- **Prefer DTD comments that start with "doc:" as annotations** - To address the lack of dedicated annotation support in DTD documents, Oxygen XML Author recommends prefixing with the `doc:` particle all comments intended to be shown to the developer who writes an XML validated against a DTD schema.

  When this option is enabled, Oxygen XML Author uses the following mechanism to collect annotations:

  - if at least one `doc:` comment is found in the entire DTD, only comments of this type are displayed as annotations
  - if no `doc:` comment is found in the entire DTD, all comments are considered annotations and displayed as such

  When the option is disabled, all comments, regardless of their type, are considered annotations and displayed as such.

- **Use all Relax NG annotations as documentation** - When this option is selected, any element outside the Relax NG namespace, that is `http://relaxng.org/ns/structure/1.0`, is considered annotation and is displayed in the annotation window next to the **Content Completion Assistant** window and in the **Model** view. When this option is not selected, only elements from the Relax NG annotations namespace, that is `http://relaxng.org/ns/compatibility/annotations/1.0` are considered annotations.

XPath Preferences

Oxygen XML Author provides content-completion support for XPath expressions. To configure the options for the content completion in XPath expressions, *open the **Preferences** dialog* and go to **Editor** > **Content Completion** > **XPath**.

The following options are available:

- **Enable content completion for XPath expressions** - Enables the **Content Completion Assistant** in XPath expressions that you enter in the `match`, `select`, and `test` XSL attributes and also in the XPath toolbar.
  - **Include XPath functions** - When this option is selected, XPath functions are included in the content completion suggestions.
  - **Include XSLT functions** - When this option is selected, XSLT functions are included in the content completion suggestions.
  - **Include axes** - When this option is selected, XSLT axes are included in the content completion suggestions.
- **Show signatures of XSLT / XPath functions** - makes the editor indicate the signature of the XPath function located at the caret position in a tooltip.
- **Function signature window background** - specifies the background color of the tooltip window.
- **Function signature window foreground** - specifies the foreground color of the tooltip window.

JavaScript Preferences

Oxygen XML Author can provide content completion suggestions when you are writing JavaScript files. To configure content completion support for JavaScript, *open the **Preferences** dialog* and go to **Editor** > **Content Completion** > **JavaScript**. You can configure the following options:

- **Enable content completion** - Enables the content completion support for JavaScript files.

- **Use built-in libraries** - Allows Oxygen XML Author to include components (object names, properties, functions, and variables) collected from the built-in JavaScript library files when making suggestions.
- **Use defined libraries** - Oxygen XML Author can also use JavaScript libraries to when making suggestions. List the paths (URIs) of any JavaScript files you want Oxygen XML Author to use when making suggestions.

> **Note:** The paths can contain editor variables such as `${pdu}`, or `${oxygenHome}`. You can make these paths relative to the project directory or installation directory.

Colors Preferences

Oxygen XML Author supports syntax highlighting of XML in the *Text mode* editor, JavaScript / JSON, PHP, CSS documents.

To configure syntax highlighting, *open the **Preferences** dialog* and go to **Editor** > **Colors**.

To set syntax colors for a language, expand the listing for that language in the top panel to show the list of syntax items for that language. Use the color and style selectors to change how each syntax item is displayed. The results of your changes are shown in the preview panel. If you do not know the name of the syntax token that you want to configure, click on that token in the **Preview** area to select it.

> **Note:** All default color sets come with a high-contrast variant, which is automatically used when you switch to a black-background high-contrast theme in your Windows operating system settings. The high-contrast theme will not overwrite any default color you set in **Colors** preferences page.

The settings for XML documents are used also in XSD, XSL, RNG documents. The **Preview** are has separate tabs for XML, XSD, XSL, RNG.

The **Enable nested syntax highlight** option controls if different content types mixed in the same file (like PHP, JS and CSS scripts inside an HTML file) are highlighted according with the color schemes defined for each content type.

Elements / Attributes by Prefix Preferences

Oxygen XML Author lets you specify different colors for XML elements and attributes with specific namespace prefixes. To configure the **Elements / Attributes by Prefix** preferences, *open the **Preferences** dialog* and go to **Editor** > **Colors** > **Elements / Attributes by Prefix**.

To change the syntax coloring for a specific namespace prefix, choose the prefix from the list, or add a new one using the **New** button, and use the color and style selectors to set the syntax highlighting style for that namespace prefix.

> **Note:** Syntax highlighting is based on the literal namespace prefix, not the namespace that the prefix is bound to in the document.

If you want only the prefix, and not the whole element or attribute name, to be styled differently, select **Draw only the prefix with a separate color**.

Open / Save Preferences

Oxygen XML Author lets you control a number of things about how files are opened and saved. To configure the **Open / Save** options, *open the **Preferences** dialog* and go to **Editor** > **Open / Save**.

Open

The following options apply to opening bidirectional (BIDI) text in the *text mode* editor.

- **When bidirectional (BIDI) text is detected** - Allows you to choose what happens 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**.
- **Disable bidirectional text support for documents larger than (Characters)** - Enabling bidirectional text editing support can affect performance on large files. When this option is selected, bidirectional editing is turned off for files exceeding the specified size, even if the **When bidirectional (BIDI) text is detected** option is set to **Enable bidirectional editing mode**.

Save

The following options apply to saving files in all edit modes:

- **Safe save (only for local files)** - This option provides an increased degree of protection in the unlikely event of a failure of the **Save** action. It 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 directory, **OxygenXMLTemp**.
- **Make backup copy on save (only for local files)** - If selected, a backup copy is made when saving the edited document. This option is available only for local files (files that are stored on the local file system). The default backup file extension is `.bak`, but you can change it as you prefer.
- **Enable automatic save** - When selected, your document are saved automatically after a preset time interval.
- **Automatic save interval (minutes)** - Selects the interval in minutes between two automatic save actions.
- **Save all files before transformation or validation** - Saves all open files before validating or transforming an XML document. This ensures that any dependencies are resolved, for example when modifying both the XML document and its XML Schema.
- **Check errors on save** - If enabled, Oxygen XML Author checks your document for errors before saving it.
- **Save all files before calling external tools** - If selected, all files are saved before executing an *external tool*.

Performance

The following options cover performance issues when dealing with long files or files with long lines:

- **Optimize loading in the Text edit mode for files over (MB)** - File loading is optimized for reduced memory usage for any file with a size larger than this value. This is useful for editing very large files, but it comes with *several restrictions* on memory-intensive operations.
- **Show warning when loading large documents** - Oxygen XML Author will warn you if you open a file larger than specified.
- **Optimize loading for documents with lines longer than (Characters)** - Line wrap is turned on for a document containing lines that exceed the length specified with this option. For a list of the restrictions applied to a document with long lines, see *the **Editing Documents with Long Lines** section.*
- **Show warning when loading documents with long lines** - When selected, Oxygen XML Authorwill warn you when you open a file with any lines longer than specified. To reduce the length of lines in a file, *format and indent the document* after it is opened in the editor panel. For a list of the restrictions applied to a document with long lines, see *Editing Documents with Long Lines* on page 220.
- **Clear undo buffer on save** - If selected, Oxygen XML Author clears its undo buffer when you save a document. Only modifications made after you have saved the document can be undone. Select this option if you encounter frequent *out of memory* problems (**OutOfMemoryError**) when editing very large documents.
- **Consider application bundles to be directories when browsing** - This option is available only on the Mac OS X platform. When selected, the file browser dialog allows browsing inside an application bundle as in a regular folder. Otherwise, the file browser dialog does not allow browsing inside an application bundle, as the Finder application does on Mac OS X. The same effect can be obtained by setting the property *apple.awt.use-file-dialog-packages* to *true* or *false* in the `Info.plist` descriptor file of the Oxygen XML Author application:

```
<key>apple.awt.use-file-dialog-packages</key>
<string>false</string>
```

Templates Preferences

This page groups the preferences for code templates and document templates:

- *Code Templates*
- *Document Templates*

Code Templates Preferences

Code templates are code fragments that can be inserted at the editing position. Oxygen XML Author comes with a set of ready-to use templates for XSL, XQuery, and XML Schema. You can define your own code templates and share them with your colleagues using the template export and import functions.

To configure the **Code Templates** options, *open the **Preferences** dialog* and go to **Editor** > **Templates** > **Code Templates**.

This preferences page contains a list with all available code templates (both built-in and custom created ones) and a code preview area. You can disable any code template by deselecting it.

The following actions are available:

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.

You can use the following *editor variables* when you define a code template :

- *${caret}* - The position where the caret is inserted. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- *${selection}* - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- *${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}* - To prompt for values at runtime, use the *ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value'')* editor variable. You can set the following parameters:

  - `'message'` - The displayed message. Note the quotes that enclose the message.
  - `type` - Optional parameter. Can have one of the following values:

    - `url` - Input is considered an URL. Oxygen XML Author checks that the URL is valid before passing it to the transformation.
    - `password` - Input characters are hidden.
    - `generic` - The input is treated as generic text that requires no special handling.
    - `relative_url` - Input is considered an URL. Oxygen XML Author tries to make the URL relative to that of the document you are editing.

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

    - `combobox` - Displays a dialog that contains a non-editable combo-box.
    - `editable_combobox` - Displays a dialog that contains an editable combo-box.
    - `radio` - Displays a dialog that contains radio buttons.

  - `'default-value'` - optional parameter. Provides a default value in the input text box;

    Examples:

    - `${ask('message')}` - Only the message displayed for the user is specified.
    - `${ask('message', generic, 'default')}` - `'message'` is displayed, the type is not specified (the default is string), the default value is `'default'`.

- ${ask('message', password)} - 'message' is displayed, the characters typed are masked with a circle symbol.
  - ${ask('message', password, 'default')} - same as before, the default value is 'default'.
  - ${ask('message', url)} - 'message' is displayed, the parameter type is URL.
  - ${ask('message', url, 'default')} - same as before, the default value is 'default'.

- *${timeStamp}* - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- *${uuid}* - Universally unique identifier;An unique sequence of 32 hexadecimal digits generated by the Java *UUID* class.
- *${id}* - Application-level unique identifier;A short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
- *${cfn}* - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- *${cfne}* - Current file name with extension. The current file is the one currently opened and selected.
- *${cf}* - Current file as file path, that is the absolute file path of the current edited document.
- *${cfd}* - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- *${frameworksDir}* - The path (as file path) of the [OXYGEN_DIR]/frameworksdirectory.
- *${pd}* - Current project folder as file path. Usually the current folder selected in the Project View.
- *${oxygenInstallDir}* - Oxygen XML Author installation folder as file path.
- *${homeDir}* - The path (as file path) of the user home folder.
- *${pn}* - Current project name.
- *${env(VAR_NAME)}* - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the *${system(var.name)}* editor variable.
- *${system(var.name)}* - Value of the var.name Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as -Dvar.name=var.value. If you are looking for operating system environment variables, use the *${env(VAR_NAME)}* editor variable instead.
- *${date(pattern)}* - Current date. The allowed patterns are equivalent to the ones in the *Java SimpleDateFormat class*. Example: yyyy-MM-dd;

  > **Note:** This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to *http://www.w3.org/TR/xmlschema-2/#date*. For details about xs:datetime, go to *http://www.w3.org/TR/xmlschema-2/#dateTime*.

Document Templates Preferences

Oxygen XML Author provides a selection of document templates that make it easier to create new documents in a variety of formats. The list of available templates is presented when you create a new document. You can add your own templates to this list by creating template files in a directory and adding that directory to the list of template directories that Oxygen XML Author uses. To add a template directory, *open the **Preferences** dialog* and go to **Editor** > **Templates** > **Document Templates**.

You can add new document template location folders and manage existing ones. You can also alter the order in which Oxygen XML Author looks into these directories by using the **Up** and **Down** buttons.

Spell Check Preferences

Oxygen XML Author provides spell check support in the *text* and *author* edit modes. To configure the **Spell Check** options, *open the **Preferences** dialog* and go to **Editor** > **Spell Check**.

The following options are available:

- **Spell checking engine** - Oxygen XML Author ships with two spell check engines, *Hunspell* and *Java spell checker*. The two engines come with different dictionaries. You can select the engine that provide the dictionaries that best

meet your needs. When you select an engine here, the list of languages in the **Default language** option changes based on the available dictionaries for the engine you have chosen.

- **Automatic Spell Check** - When selected, Oxygen XML Author checks spelling as you type and highlights misspelled words in the document.
- **Select editors** - You can select which editors (and therefore which file types) will be automatically spelled checked. File types for which automatic spell check is generally not useful, like CSS and DTD, are excluded by default.
- **Spell check highlight color** - Use this option to set the color used by the spell check engine to highlight spelling errors.
- **Default language** - The default language list allows you to choose the language used by the spell check engine when the language is not specified in the source file. You can *add additional dictionaries to the spell check engines*.
- **Use "lang" and "xml:lang" attributes** - When this option is selected, the contents of any element with one of the `lang` or `xml:lang` attributes is checked in that language, if a suitable dictionary is available. If a suitable dictionary is not available, the element is not checked.
- **XML spell checking in** - These options allow you to specify if the spell checker will check spelling inside XML comments, attribute values, text, and CDATA sections.
- **Case sensitive** - When selected, the spell checker reports capitalization errors, for example a word that starts with lowercase after *etc.* or *i.e.*.
- **Ignore mixed case words** - When selected, the spell checker does not check words containing mixed case characters, for example, *SpellChecker*.
- **Ignore words with digits** - When selected, the spell checker does not check words containing digits, for example, *b2b*).
- **Ignore Duplicates** - When selected, the spell checker does not signal two successive identical words as an error.
- **Ignore URL** - When selected, the spell checker ignores words looking like URLs or file names, for example, *www.oxygenxml.com* or *c:\boot.ini*.
- **Check punctuation** - When selected, the spell checker checks punctuation. Misplaced white space and unusual sequences, like a period following a comma, are highlighted as errors.
- **Allow compounds words** - When selected, 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 selected, 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 selected, the spell checker accepts any word ending with registered file extensions, for example, *myfile.txt*, *index.html*, etc.
- **Ignore acronyms** - When selected, acronyms are not reported as errors.
- **Ignore elements** - You can tell the spell checker to ignore words in XML elements that match a certain XPath expression. The following restricted set of XPath expressions are supported:

  - '/' and '//' separators;
  - '*' wildcard.

  An example of an allowed XPath expression is: */a/*/b*.

Dictionaries Preferences

To set Dictionaries preferences, *open the **Preferences** dialog* and go to **Editor** > **Spell Check** > **Dictionaries**. This page allows you to configure the dictionaries and term lists that Oxygen XML Author uses and to choose where to save new learned words.

> **Note:** A term list must have the `.tdi` extension.

The following options are available:

- **Dictionaries and term lists default folder** - Specifies the default location where the dictionaries and term lists that Oxygen XML Author uses are stored;
- **Include dictionaries and term list from** - Specifies a location where you can store dictionaries and term lists, different from the default one;

📄 **Note:** In case the additional location contains a file with the same name as one from the default location, the additional file is proffered.

📄 **Note:** The spell checker takes into account term lists, collected both from the default and additional locations.

- **Save learned words in the following location** - Specifies the target where the newly learned words are saved. By default, the target is the application's preferences folder, but you can also choose a custom location.
- **Delete learned words** - Opens the list of learned words, allowing you to select the items you want to remove, without deleting the dictionaries and term lists.

📄 **Note:** These options are valid when Oxygen XML Author uses the Hunspell spell checking engine.

Document Checking Preferences

To configure the **Document Checking** options, *open the **Preferences** dialog* and go to **Editor** > **Document Checking**. This preferences page contains preferences for configuring how a document is checked for both well-formedness errors and validation errors.

The following options are available:

- **Maximum number of validation highlights** - If validation generates more errors than the number from this option only the first errors up to this number are highlighted in editor panel and on stripe that is displayed at right side of editor panel. This option is applied both for *automatic validation* and *manual validation*.
- **Validation error highlight color** - The color used to highlight validation errors in the document.
- **Validation warning highlight color** - The color used to highlight validation warnings in the document.
- **Validation success color** - The color used to highlight in the vertical ruler bar the success of the validation operation.
- **Always show validation status** - If this option is selected the line at the bottom of the editor panel which presents the current validation error or warning is always visible. This is useful to avoid scrolling problems when **Automatic validation** is enabled and the vertical scroll bar may change position due to displaying an error message while the document is edited.
- **Enable automatic validation** - Validation of edited document is executed in background as the document is modified by editing in Oxygen XML Author.
- **Delay after the last key event (s)** - The period of keyboard inactivity which starts a new validation (in seconds).

Custom Validation Engines Preferences

To configure **Custom Validation Engines** options, *open the **Preferences** dialog* and go to **Editor** > **Custom Validations**.

If you want to add a new custom validation tool or edit the properties of an exiting one you can use the **Custom Validator** dialog displayed by pressing the **New** button or the **Edit** button.
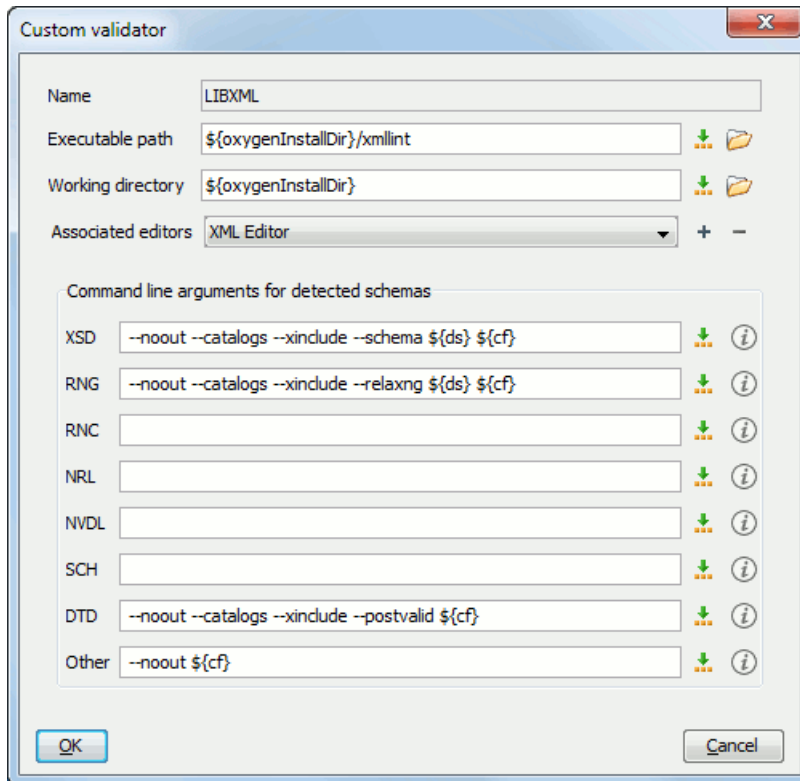
Figure 266: Edit a Custom Validator

The configurable parameters of a custom validator are the following:

- **Name** - Name of the custom validation tool displayed in the **Custom Validation Engines** toolbar.
- **Executable path** - Path to the executable file of the custom validation tool. You can insert here *editor variables* like *${home}*, *${pd}*, *${oxygenInstallDir}*, etc.
- **Working directory** - The working directory of the custom validation tool.
- **Associated editors** - The editors which can perform validation with the external tool: the XML editor, the XSL editor, the XSD editor, etc.
- **Command line arguments for detected schemas** - Command line arguments used in the commands that validate the current edited file against different types of schema: W3C XML Schema, Relax NG full syntax, Relax NG compact syntax, NVDL, Schematron, DTD, etc.. The arguments can include any custom switch (like -rng) and the following editor variables:

  - *${cf}* - Current file as file path, that is the absolute file path of the current edited document.
  - *${cfu}* - The path of the current file as a URL. The current file is the one currently opened and selected.
  - *${ds}* - The path of the detected schema as a local file path for the current validated XML document.
  - *${dsu}* - The path of the detected schema as an URL for the current validated XML document.

Increasing the stack size for validation engines

To prevent the appearance of a *StackOverflowException*, use one of the following methods:

- use the **com.oxygenxml.stack.size.validation.threads** property to increase the size of the stack for validation engines. The value of this property is specified in bytes. For example, to set a value of one megabyte specify 1x1024x1024=1048576;
- increase the value of the **-Xss** parameter.

  Note: Increasing the value of the **-Xss** parameter affects all the threads of the application.

CSS Validator Preferences

To configure the **CSS Validator** preferences, *open the **Preferences** dialog* and go to **CSS Validator**.

You can configure the following options for the built-in CSS validator of Oxygen XML Author:

- **Profile** - Selects one of the available validation profiles: **CSS 1**, **CSS 2**, **CSS 2.1**, **CSS 3**, **CSS 3 with Oxygen extensions**, **SVG**, **SVG Basic**, **SVG Tiny**, **Mobile**, **TV Profile**, **ATSC TV Profile**. The **CSS 3 with Oxygen extensions** profile includes all the CSS 3 standard properties and the *CSS extensions specific for Oxygen* that can be used in *Author mode*. That means all Oxygen specific extensions are accepted in a CSS stylesheet by *the built-in CSS validator* when this profile is selected.
- **Media type** - Selects one of the available mediums: **all**, **aural**, **braille**, **embossed**, **handheld**, **print**, **projection**, **screen**, **tty**, **tv**, **presentation**, **oxygen**.
- **Warning level** - Sets the minimum severity level for reported validation warnings. Can be one of: **All**, **Normal**, **Most Important**, **No Warnings**.
- **Ignore properties** - Here you can type comma separated patterns that match the names of CSS properties that will be ignored at validation. As wildcards you can use:
  - ***** to match any string;
  - **?** to match any character.

- **Recognize browser CSS extensions (applies also to content completion)** - If checked, Oxygen XML Author recognizes (no validation is performed) browser-specific CSS properties. The **Content Completion Assistant** lists these properties at the end of its list, prefixed with the following particles:
  - `-moz-` for Mozilla;
  - `-ms-` for Internet Explorer;
  - `-o-` for Opera;
  - `-webkit-` for Safari/Webkit.

XML Preferences

This section describes the panels that contain the user preferences related with XML.

XML Catalog Preferences

To configure the **XML Catalog** options, *open the **Preferences** dialog* and go to **XML** > **XML Catalog**.

The following options are available:

- **Prefer** - the prefer setting determines whether public identifiers specified in the catalog are used in favor of system identifiers supplied in the document. Suppose you have an entity in your document for which both a public identifier and a system identifier has been specified, and the catalog only contains a mapping for the public identifier (e.g., a matching public catalog entry). If **public** is selected, the URI supplied in the matching public catalog entry is used. If **system** is selected, the system identifier in the document is used.

  > **Note:** If the catalog contained a matching system catalog entry giving a mapping for the system identifier, that mapping would have been used, the public identifier would never have been considered, and the setting of override would have been irrelevant.

  Generally, the purpose of catalogs is to override the system identifiers in XML documents, so **Prefer** should usually be **public** in your catalogs.
- When using catalogs it is sometimes useful to see what catalog files are parsed, if they are valid or not, and what identifiers are resolved by the catalogs. The **Verbosity** option selects the detail level of such logging messages of the XML catalog resolver that will be displayed in the **Catalogs** view at the bottom of the window:
  - **None** - No message is displayed by the catalog resolver when it tries to resolve a URI reference, a SYSTEM one or a PUBLIC one with the XML catalogs specified in this panel.
  - **Unresolved entities** - Only the logging messages that track the failed attempts to resolve references are displayed.
  - **All messages** - The messages of both failed attempts and successful ones are displayed.

- If **Resolve schema locations also through system mappings** is enabled, Oxygen XML Author analyzes both *uri* and *system* mappings in order to resolve the location of schema.

- If **Process namespaces through URI mappings for XML Schema** is selected then the target namespace of the imported XML Schemas is resolved through the *uri* mappings. The namespace is taken into account only when the schema specified in the *schemaLocation* attribute was not resolved successfully.

- If the **Use default catalog** option is checked the first XML catalog which Oxygen XML Author 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 Author frameworks: DocBook, DITA, TEI, XHTML, SVG, etc.

You can also add or configure catalogs at framework level in the *Document Type Association* preferences page.

When you add, delete or edit an XML catalog to / from the list, reopen the currently edited files which use the modified catalog or run *the Validate action* so that the XML catalog changes take full effect.

XML Parser Preferences

To configure the **XML Parser** options, *open the Preferences dialog* and go to **XML** > **XML Parser**.

The configurable options of the built-in XML parser are the following:

- **Enable parser caching (validation and content completion)** - enables re-use of internal models when validating and provides content completion in opened XML files which reference the same schemas (grammars) like DTD, XML Schema, RelaxNG;
- **Ignore the DTD for validation if a schema is specified** - forces validation against a referred schema (W3C XML Schema, Relax NG schema) even if the document includes also a DTD DOCTYPE declaration. This option is useful when the DTD declaration is used only to declare DTD entities and the schema reference is used for validation against a W3C XML Schema or a Relax NG schema;

  > 📑 **Note:** Schematron schemas are treated as additional schemas. The validation of a document associated with a DTD and referring a Schematron schema is executed against both the DTD and the Schematron schema, regardless of the value of the **Ignore the DTD for validation if a schema is specified** option.

- **Enable XInclude processing** - enables XInclude processing. If checked, the XInclude support in Oxygen XML Author is turned on for validation, rendering in Author mode and transformation of XML documents;
- **Base URI fix-up** - according to the specification for XInclude, processors must add an `xml:base` attribute to elements included from locations with a different base URI. Without these attributes, the resulting infoset information would be incorrect.

  Unfortunately, these attributes make XInclude processing not transparent to Schema validation. One solution to this is to modify your schema to allow `xml:base` attributes to appear on elements that might be included from different base URIs.

  If the addition of `xml:base` and / or `xml:lang` is undesired by your application, you can disable base URI fix-up.

- **Language fix-up** - the processor will preserve language information on a top-level included element by adding an `xml:lang` attribute if its include parent has a different [language] property. If the addition of `xml:lang` is undesired by your application, you can disable the language fix-up;
- **DTD post-validation** - enable this option to validate an XML file against the associated DTD, after all the content merged to the current XML file using `XInclude` was resolved. In case you disable this option, the current XML file is validated against the associated DTD before all the content merged to the current XML file using `XInclude` is resolved.

XML Schema Preferences

To configure the **XML Schema** options, *open the Preferences dialog* and go to **XML** > **XML Parser** > **XML Schema**.

This preferences page allows you to configure the following options:

- **Default XML Schema version** - Allows you to select the version of W3C XML Schema: XML Schema 1.0 or XML Schema 1.1;

  📄 **Note:** You are also able to set the XML Schema version using the **Customize** option in *New dialog box*.

- **Default XML Schema validation engine** - Allows you to set the default XML Schema validation engine either to Xerces, or to Saxon EE;

Xerces validation features

- **Enable full schema constraint checking (http://apache.org/xml/features/validation/schema-full-checking)** - Sets the *schema-full-checking* feature to true. This enables a validation of the parsed XML document against a schema (W3C XML Schema or DTD) while the document is parsed;
- **Enable honour all schema location feature (http://apache.org/xml/features/honour-all-schema-location)** - Sets the *honour-all-schema-location* feature to true. All the files that declare W3C XML Schema components from the same namespace are used to compose the validation model. In case this option is disabled, only the first W3C XML Schema file that is encountered in the XML Schema import tree is taken into account;
- **Enable full XPath 2.0 in assertions and alternative types (http://apache.org/xml/features/validation/cta-full-xpath-checking)** - When selected, you can use the full XPath support in assertions and alternative types. Otherwise, there is available only the XPath support offered by the Xerces engine.
- **Assertions can see comments and processing instructions (http://apache.org/xml/features/validation/assert-comments-and-pi-checking)** - Controls whether comments and processing instructions are visible to the XPath expression used for defining an assertion in XSD 1.1;

Saxon validation features

- **Multiple schema imports** - Forces xs:import to fetch the referenced schema document. By default, the xs:import fetches the document only if no schema document for the given namespace has already been loaded. With this option in effect, the referenced schema document is loaded unless the absolute URI is the same as a schema document already loaded;
- **Assertions can see comments and processing instructions** - Controls whether comments and processing instructions are visible to the XPath expression used to define an assertion. By default (unlike Saxon 9.3), they are not made visible.

Relax NG Preferences

To configure the **Relax NG** options, *open the Preferences dialog* and go to **XML** > **XML Parser** > **Relax NG**.

The following options are available in this page:

- **Check feasibly valid** - checks whether Relax NG documents can be transformed into valid documents by inserting any number of attributes and child elements anywhere in the tree;

  📄 **Note:** Enabling this option disables the **Check ID/IDREF** option.

- **Check ID/IDREF** - checks the ID/IDREF matches when a Relax NG document is validated;
- **Add default attribute values** - default values are given to the attributes of documents validated using Relax NG. These values are defined in the Relax NG schema.

Schematron Preferences

To configure the **Schematron** options, *open the Preferences dialog* and go to **XML** > **XML Parser** > **Relax NG**.

The following options are available in this preferences page:

- **Schematron XPath Version** - selects the version of XPath for the expressions that are allowed in Schematron assertion tests: 1.0 or 2.0. This option is applied both in standalone Schematron schemas and in embedded Schematron rules, both in Schematron 1.5 and in ISO Schematron;
- **Optimize (visit-no-attributes)** - in case your ISO Schematron assertion tests do not contain the attributes axis you should check this option for faster ISO Schematron validation;

- **Allow foreign elements (allow-foreign)** - enables support for `allow-foreign` on ISO Schematron. This option is used to pass non-Schematron elements to the generated stylesheet;
- **Use Saxon EE (schema aware) for xslt2 query binding** - when enabled, Saxon EE is used for `xslt2` query binding. In case this option is disabled, Saxon PE is used.

XProc Engines Preferences

Oxygen XML Author comes with a built-in XProc engine called *Calabash*. An external XProc engine can be configured in this panel.

When **Show XProc messages** is selected all messages emitted by the XProc processor during a transformation will be presented in the results view.

For an external engine the value of the **Name** field will be displayed in the XProc transformation scenario and in the command line that will start it.
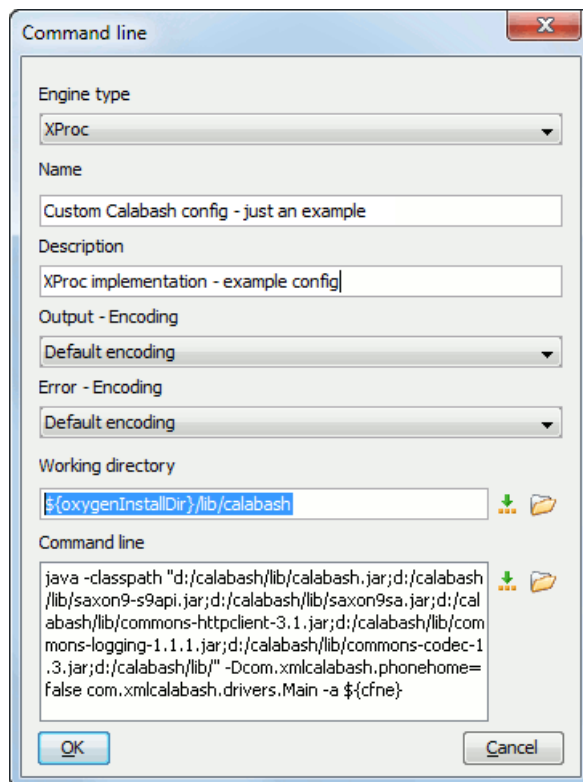
Figure 267: Creating an XProc external engine

Other parameters that can be set for an XProc external engine are the following: , and the error stream of the engine, the working directory of the command that will start the engine. The encodings will be used for reading and displaying the output of the engine. The working directory and

- a textual description that will appear as tooltip where the XProc engine will be used;
- the encoding for the output stream of the XProc engine, used for reading and displaying the output messages;
- the encoding for the error stream of the XProc engine, used for reading and displaying the messages from the error stream;
- the working directory for resolving relative paths;
- the command line that will run the XProc engine as an external process; the command line can use *built-in editor variables* and *custom editor variables* for parameterizing a file path.

> **Note:** You can configure the Saxon processor using the `saxon.config` file. For further details about configuring this file go to *http://www.saxonica.com/documentation/configuration/configuration-file.xml*.

> **Note:** You can configure Calabash using the `calabash.config` file.

> **Note:** These files are located in `[OXYGEN_DIR]\lib\xproc\calabash`. In case they do not exist, you have to create them.

XSLT-FO-XQuery Preferences

To configure the **XSLT/FO/XQuery** options, *open the **Preferences dialog*** and go to **XML** > **XSLT/FO/XQuery**. This panel contains only the most generic options for working with XSLT / XSL-FO / XQuery processors. The more specific options are grouped in other panels linked as child nodes of this panel in the tree of the **Preferences** dialog.

There is only one generic option available:

Create transformation temporary files in system temporary directory - It should be selected only when the temporary files necessary for performing transformations are created in the same folder as the source of the transformation (the default behavior, when this option is not selected) and this breaks the transformation. An example of breaking the transformation is when the transformation processes all the files located in the same folder as the source of the transformation, which will include the temporary files, and the result is incorrect or the transformation fails due to this fact.

XSLT Preferences

To configure the **XSLT** options, *open the **Preferences dialog*** and go to **XML** > **XSLT/FO/XQuery** > **XSLT**.

Oxygen XML Author gives you the possibility to use an XSLT transformer implemented in Java (other than the XSLT transformers that come bundled with Oxygen XML Author). To use a different XSLT transformer, specify the name of the transformer factory class. Oxygen XML Author sets this transformer factory class as the value of the Java property `javax.xml.transform.TransformerFactory`.

You can customize the following XSLT preferences:

- **Value** - Allows the user to enter the name of the transformer factory Java class;
- **XSLT 1.0 Validate with** - allows you to set the XSLT engine used for validation of XSLT 1.0 documents;
- **XSLT 2.0 Validate with** - allows you to set the XSLT Engine used for validation of XSLT 2.0 documents;
- **XSLT 3.0 Validate with** - allows you to set the XSLT Engine used for validation of XSLT 3.0 documents.

> **Note:** Saxon-HE does not implement any XSLT 3.0 features. Saxon-PE implements a selection of XSLT 3.0 (and XPath 3.0) features, with the exception of schema-awareness and streaming. Saxon-EE implements additional features relating to streaming (processing of a source document without constructing a tree in memory. For further details about XSLT 3.0 conformance, go to *http://www.saxonica.com/documentation/index.html#!conformance/xslt30*.

Saxon6 Preferences

To configure the **Saxon 6** options, *open the **Preferences dialog*** and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **Saxon** > **Saxon 6**.

XML / XSLT-FO-XQuery / XSLT / Saxon / 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")
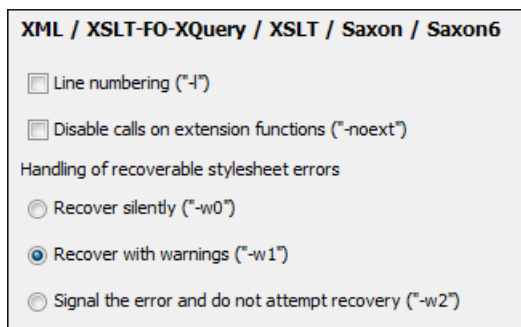
Figure 268: The Saxon 6 XSLT Preferences Panel

The built-in Saxon 6 XSLT processor can be configured with the following options:

- **Line numbering** - specifies whether line numbers are maintained and reported in error messages for the XML source document;
- **Disable calls on extension functions** - if enabled, external functions called is disallowed. Checking this is recommended in an environment where untrusted stylesheets may be executed. Also disables user-defined extension elements, together with the writing of multiple output files, all of which carry similar security risks;
- **Handling of recoverable stylesheet errors** - allows the user to choose how dynamic errors are handled. Either one of the following options can be selected:

  - **recover silently** - continue processing without reporting the error;
  - **recover with warnings** - issue a warning but continue processing;
  - **signal the error and do not attempt recovery** - issue an error and stop processing.

Saxon-HE/PE/EE Preferences

To configure the **Saxon HE/PE/EE** options, *open the **Preferences** dialog* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **Saxon** > **Saxon HE/PE/EE**.

Oxygen XML Author allows you to configure the following XSLT options for the Saxon 9.5.1.7 transformer (editions: Home Edition, Professional, and Enterprise):

- **Use a configuration file ("-config")** - Sets a Saxon 9.5.1.7 configuration file that is executed for XSLT transformation and validation processes.
- **Version warnings ("-versmsg")** - Warns you when the transformation is applied to an XSLT 1.0 stylesheet.
- **Line numbering ("-l")** - Error line number is included in the output messages.
- **Debugger trace into XPath expressions (applies to debugging sessions)** - Instructs the XSLT Debugger to step into XPath expressions.
- **Expand attributes defaults ("-expand")** - Specifies whether the attributes defined in the associated DTD or XML Schema that have default values are expanded in output of the transformation you are executing.
- **DTD validation of the source ("-dtd")** - The following options are available:

  - **On** - Requests *DTD* validation of the source file and of any files read using the document() function.
  - **Off** - (default setting) Suppresses DTD validation.
  - **Recover** - Performs DTD validation but treats the errors as non-fatal.

    > **Note:** Any external DTD is likely to be read even if not used for validation, because DTDs can contain definitions of entities.

- **Recoverable errors ("-warnings")** - Policy for handling recoverable errors in the stylesheet: Allows you to choose how dynamic errors are handled. One of the following options can be selected:

  - **Recover silently ("silent")**
  - **Recover with warnings ("recover")** - default setting
  - **Signal the error and do not attempt recovery ("fatal")**

- **Strip whitespaces ("-strip")** - Strip whitespaces feature can be one of the following three options:

  - **All ("all")** - Strips all whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes in the source document.
  - **Ignorable ("ignorable")** - Strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
  - **None ("none")** - default setting. No whitespaces are stripped before further processing. However, whitespace are stripped if this is specified in the stylesheet using `xsl:strip-space`.

- **Optimization level ("-opt")** - Set optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization); currently all values other than 0 result in full optimization but this is likely to change in the future. The default is full optimization; this feature allows optimization to be suppressed in cases where reducing compile

time is important, or where optimization gets in the way of debugging, or causes extension functions with side-effects to behave unpredictably. (Note however, that even with no optimization, the lazy evaluation may still cause the evaluation order to be not as expected).

The advanced options available only in Saxon PE / EE are:

- **Allow calls on extension functions ("-ext")** - If checked, the stylesheet is allowed to call external Java functions. This does not affect calls on integrated extension functions, including Saxon and EXSLT extension functions. This option is useful when loading an untrusted stylesheet, perhaps from a remote site using an http:// URL; it ensures that the stylesheet cannot call arbitrary Java methods and thus gain privileged access to resources on your machine.
- **Register Saxon-CE extension functions and instructions** - Registers the Saxon-CE extension functions and instructions when compiling a stylesheet using the Saxon 9.5.1.7 processors.

The advanced options available only in Saxon EE are:

- **XML Schema version** - Use this option to change the default XML Schema version. To change the default XML Schema version, *open the **Preferences** dialog* and go to **XML** > **XML Parser** > **XML Schema**.

  > 📝 **Note:** This option is available when you configure the Saxon EE advanced options from a transformation scenario.

- **Validation of the source file ("-val")** - Requests schema-based validation of the source file and of any files read using the document() or similar functions. Validation is available only with Saxon-EE, and this flag automatically switches on the -sa option. Available options:

  - **Schema validation ("strict")** - This mode requires an XML Schema and specifies that the source documents should be parsed with schema-validation enabled.
  - **Lax schema validation ("lax")** - This mode specifies if the source documents should be parsed with schema-validation enabled if an XML Schema is provided.
  - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.

- **Validation errors in the results tree treated as warnings ("-outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.
- **Write comments for non-fatal validation errors of the result document** - The validation messages are written (where possible) as a comment in the result document itself.
- **Generate bytecode ("--generateByteCode:(on|off)")** - When you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such treatment. For further details regarding this option, go to*http://www.saxonica.com/documentation/javadoc/*.
- **Initializer class** - Equivalent with the *-init* Saxon command-line argument. The value is the name of a user-supplied class that implements the net.sf.saxon.lib.Initializer interface; this initializer is called during the initialization process, and may be used to set any options required on the configuration programmatically. It is particularly useful for tasks such as registering extension functions, collations, or external object models, especially in Saxon-HE where the option cannot be set via a configuration file. Saxon only calls the initializer when running from the command line, but the same code may be invoked to perform initialization when running user application code.

  > ⚠ **Important:** The *advanced Saxon-HE/PE/EE options configured in a transformation scenario* override the Saxon-HE/PE/EE options defined globally.

Saxon HE/PE/EE Advanced Preferences

To configure **Saxon HE/PE/EE Advanced** preferences, *open the **Preferences** dialog* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **Saxon** > **Saxon HE/PE/EE** > **Advanced**.

Figure 269: The Saxon HE/PE/EE XSLT Advanced Preferences Panel

You can configure the following advanced XSLT options for the Saxon 9.5.1.7 transformer (all three editions: Home Edition, Professional Edition, Enterprise Edition):

- **URI Resolver class name ("-r")** - specifies a custom implementation for the URI resolver used by the XSLT Saxon 9.5.1.7 transformer (the -r option when run from the command line). The class name must be fully specified and the corresponding `jar` or `class` extension must be configured from *the dialog for configuring the XSLT extension* for the particular transformation scenario.
- **Collection URI Resolver class name ("-cr")** - specifies a custom implementation for the Collection URI resolver used by the XSLT Saxon 9.5.1.7 transformer (the -cr option when run from the command line). The class name must be fully specified and the corresponding `jar` or `class` extension must be configured from *the dialog for configuring the XSLT extension* for the particular transformation scenario.

XSLTProc Preferences

To configure **XSLTProc** options, *open the **Preferences** dialog* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **XSLTProc**.

The options of the XSLTProc processor are the same as the ones available in the command line:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when XSLTProc is used as transformer in *XSLT transformation scenarios*.
- **Skip loading the document's DTD** - If checked, the DTD specified in the DOCTYPE declaration will not be loaded.
- **Do not apply default attributes from document's DTD** - If checked, the default attributes declared in the DTD and not specified in the document are not included in the transformed document.
- **Do not use Internet to fetch DTD's, entities or docs** - If checked, the remote references to DTD's and entities are not followed.
- **Maximum depth in templates stack** - If this limit of maximum templates depth is reached the transformation ends with an error.
- **Verbosity** - If checked, the transformation will output detailed status messages about the transformation process in the **Warnings** view.
- **Show version of libxml and libxslt used** - If checked, Oxygen XML Author 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 Author will display in the **Warnings** view the URL of all the files loaded during transformation.
- **Show profile information** - If checked, Oxygen XML Author 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 Author will display in the **Warnings** view a list with all the registered extension functions, extension elements and extension modules.
- **Refuses to write to any file or resource** - If checked, the XSLTProc processor will not write any part of the transformation result to an external file on disk. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.

- **Refuses to create directories** - If checked, the XSLTProc processor will not create any directory during the transformation process. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.

MSXML Preferences

To configure the MSXML options, *open the **Preferences** dialog* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **MSXML**.

The options of the MSXML 3.0 and 4.0 processors are the same as *the ones available in the command line for the MSXML processors*:

- **Validate documents during parse phase** - If checked and either the source or stylesheet document has a DTD or schema against which its content can be checked, validation is performed.
- **Do not resolve external definitions during parse phase** - By default, MSXSL instructs the parser to resolve external definitions such as document type definition (DTD), external subsets or external entity references when parsing the source and style sheet documents. If this option is checked the resolution is disabled.
- **Strip non-significant whitespaces** - If checked, strips non-significant white space from the input XML document during the load phase. Enabling this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.
- **Show time information** - If checked, the relative speed of various transformation steps can be measured:

  - the time to load, parse, and build the input document
  - the time to load, parse, and build the stylesheet document
  - the time to compile the stylesheet in preparation for the transformation
  - the time to execute the stylesheet

- **Start transformation in this mode** - Although stylesheet execution usually begins in the empty mode, this default may be changed by specifying another mode. Changing the start mode allows execution to jump directly to an alternate group of templates.

MSXML.NET Preferences

To configure the **MSXML.NET** options, *open the **Preferences** dialog* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **MSXML.NET**.

The options of the MSXML.NET processor are:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when MSXML.NET is used as transformer in the *XSLT transformation scenario*.
- **Validate documents during parse phase** - If checked and either the source or stylesheet document has a DTD or schema against which its content can be checked, validation is performed.
- **Do not resolve external definitions during parse phase** - By default MSXML.NET resolves external definitions such as DTD external subsets or external entity references when parsing source XML document and stylesheet document. Using this option you can disable this behaviour. Note, that it may affect also the validation process for the XML document.
- **Strip non-significant whitespaces** - If checked, strips non-significant white space from the input XML document during the load phase. Enabling this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.
- **Show time information** - If checked, the relative speed of various transformation steps can be measured:

  - the time to load, parse, and build the input document
  - the time to load, parse, and build the stylesheet document
  - the time to compile the stylesheet in preparation for the transformation
  - the time to execute the stylesheet

- **Forces ASCII output encoding** - There is a known problem with .NET 1.X XSLT processor (`System.Xml.Xsl.XslTransform` class): it doesn't support escaping of characters as XML character references when they cannot be represented in the output encoding. That means that when you output a character that cannot be represented in output encoding, it will be outputted as '?'. Usually this happens when output encoding is set to

ASCII. With this option checked the output is forced to be ASCII encoded and all non-ASCII characters get escaped as XML character references (&#nnnn; form).

- **Allow multiple output documents** - This option allows to create multiple result documents using *the exsl:document extension element.*
- **Use named URI resolver class** - This option allows to specify a custom URI resolver class to resolve URI references in `xsl:import` and `xsl:include` instructions (during XSLT stylesheet loading phase) and in `document()` function (during XSL transformation phase).
- **Assembly file name for URI resolver class** - The previous option specifies partially or fully qualified URI resolver class name, e.g. `Acme.Resolvers.CacheResolver`. Such name requires additional assembly specification using this option or the next option, but fully qualified class name (which always includes an assembly specifier) is all-sufficient. See MSDN for more info about *fully qualified class names*. This option specifies a file name of the assembly, where the specified resolver class can be found.
- **Assembly GAC name for URI resolver class** - This option specifies partially or fully qualified name of the assembly in the *global assembly cache* (GAC), where the specified resolver class can be found. See MSDN for more info about *partial assembly names*. Also see the previous option.
- **List of extension object class names** - This option allows to specify *extension object* classes, whose public methods then can be used as extension functions in an XSLT stylesheet. It is a comma-separated list of namespace-qualified extension object class names. Each class name must be bound to a namespace URI using prefixes, similar to providing XSLT parameters.
- **Use specified EXSLT assembly** - MSXML.NET supports a rich library of the *EXSLT* and *EXSLT.NET extension functions* embedded or in a plugged in EXSLT.NET library. EXSLT support is enabled by default and cannot be disabled in this version. If you want to use an external EXSLT.NET implementation instead of a built-in one use this option.
- **Credential loading source xml** - This option allows to specify user credentials to be used when loading XML source documents. The credentials should be provided in the *username:password@domain* format (all parts are optional).
- **Credential loading stylesheet** - This option allows to specify user credentials to be used when loading XSLT stylesheet documents. The credentials should be provided in the *username:password@domain* format (all parts are optional).

FO Processors Preferences

Besides Apache FOP, the built-in formatting objects processor, you can configure other external processors and set them in the transformation scenarios for processing XSL-FO documents.

Oxygen XML Author provides an easy way to add two of the most used commercial FO processors: *RenderX XEP* and *Antenna House XSL Formatter*. You can easily add RenderX XEP as an external FO processor if you have the XEP installed. Also, if you have the Antenna House XSL Formatter v4 or v5, Oxygen XML Author uses the environmental variables set by the XSL Formatter installation to detect and use it for XSL-FO transformations. If the environmental variables are not set for the XSL Formatter installation, you can browse and choose the executable file just as you would for XEP. You can use these two external FO processors for *DITA OT transformations scenarios* and *XML with XSLT transformation scenarios*.

To configure the **FO Processors** options, *open the Preferences dialog* and go to **XML** > **XSLT/FO/XQuery** > **FO Processors**.
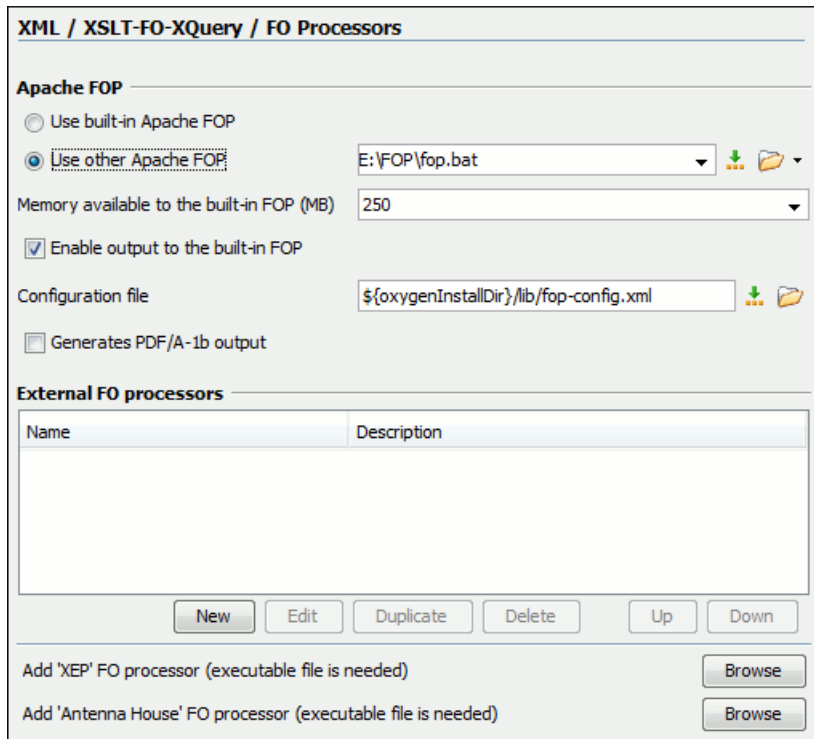
Figure 270: The FO Processors Preferences Panel

Apache FOP

The options for FO processors are the following:

- **Use built-in Apache FOP** - instructs Oxygen XML Author to use its built-in Apache FO processor;
- **Use other Apache FOP** - instructs Oxygen XML Author 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 Author window including warning messages about FO instructions not supported by Apache FOP;
- **Memory available to the built-in FOP** - if your Apache FOP transformations fail with an Out of Memory error (**OutOfMemoryError**) select from this combo box a larger value for the amount of memory reserved for FOP transformations;
- **Configuration file for the built-in FOP** - you should specify here the path to an Apache FOP configuration file, necessary for example to render to PDF a document containing Unicode content using a special *true type* font;
- **Generates PDF/A-1b output** - when selected PDF/A-1b output is generated;

  > **Note:** All fonts have to be embedded, even the implicit ones. More information about configuring metrics files for the embedded fonts can be found in *Add a font to the built-in FOP*.

  > **Note:** You cannot use the `<filterList>` key in the configuration file because FOP would generate the following error: *The Filter key is prohibited when PDF/A-1 is active*.

- **Add 'XEP' FO processor (executable file is needed)** - in case *RenderX XEP* is already installed on your computer, you can use this button to choose the XEP executable script (`xep.bat` for Windows, `xep` for Linux);
- **Add 'Antenna House' FO processor (executable file is needed)** - in case *Antenna House XSL Formatter* is already installed on your computer, you can use this button to choose the Antenna House executable script (AHFCmd.exe ,or XSLCmd.exe for Windows, AHFCmd.sh ,or XSLCmd.sh for Linux);

External FO processors

In this section you can manage the external FO processors you want to use in transformation scenarios. Press the **New** button to add a new external FO processor. The following dialog is displayed:
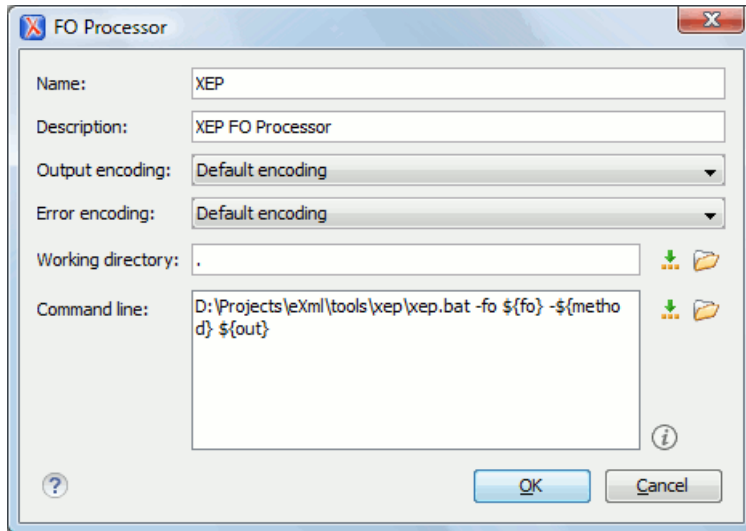
Figure 271: The External FO Processor Configuration Dialog

- **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 Author window;
- **Error Encoding** - the encoding of the FO processor error stream displayed in a results panel at the bottom of the Oxygen XML Author 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 Author 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;
  - **${frameworksDir}** - the path of the `frameworks` subdirectory of the Oxygen XML Author install directory;
  - **${oxygenInstallDir}** - the Oxygen XML Author installation directory;
  - **${ps}** - the platform-specific path separator. It is used between the library files specified in the class path of the command line.

XPath Preferences

To configure the **XPath** options, *open the Preferences dialog* and go to **XML** > **XSLT/FO/XQuery** > **XPath**.

Oxygen XML Author allows you to customize the following options:

- **Unescape XPath expression** - the entities of an XPath expressions that you type in the XPath/XQuery Builder and the *XPath toolbar* are unescaped during their execution. For example the expression

```
//varlistentry[starts-with(@os,'&#x73;')]
```

is equivalent with:

```
//varlistentry[starts-with(@os,'s')]
```

- **Multiple XPath results** - enable this option to display the results of an XPath in separate tabs in the *The Results View* on page 59.
- **No namespace** - if you enable this option, Oxygen XML Author considers unprefixed element names of the evaluated XPath 2.0 / 3.0 expressions as belonging to no namespace.
- **Use the default namespace from the root element** - if you enable this option, Oxygen XML Author considers unprefixed element names of the evaluated XPath expressions as belonging to the default namespace declared on the root element of the XML document you are querying.
- **Use the namespace of the root** - if you enable this option, Oxygen XML Author considers unprefixed element names of the evaluated XPath expressions as belonging to the same namespace as the root element of the XML document you are querying.
- **This namespace** - in this field you can enter the namespace of the unprefixed elements.
- **Default prefix-namespace mappings** - in this field you can associate prefixes with namespaces. Use these mappings when you want to define them globally, not for each document.

Custom Engines Preferences

You can configure and run XSLT and XQuery transformations with processors other than *the ones which come with the Oxygen XML Author distribution*.

📄 **Note:**

To configure the **Custom Engines** preferences, *open the Preferences dialog* and go to **XML** > **XSLT/FO/XQuery** > **Custom Engines**.

The following parameters can be configured for a custom engine:
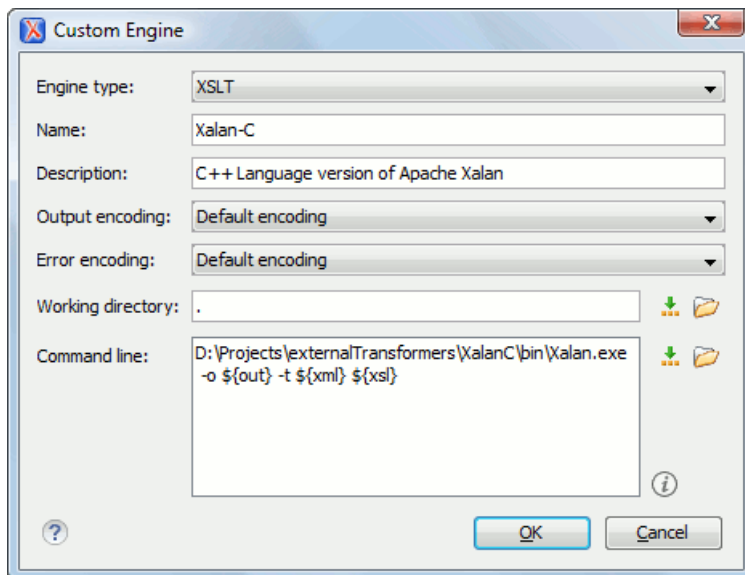
Figure 272: Parameters of a Custom Engine

- **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 Author install directory.

- **Command line** - The command line that must be executed by Oxygen XML Author 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.

Ant Preferences

To set Ant preferences, *open the **Preferences** dialog* and go to **XML** > **Ant**. This panel allows you to choose the directory containing the Apache Ant libraries (the so-called *Ant Home*) that Oxygen XML Author uses to handle Ant build files.

There are two options available:

- **Built-in** - the path to the Ant distribution that comes bundled with Oxygen XML Author installation kit;
- **Custom** - the path to an Ant distribution of your choice.

Import Preferences

To configure the **Import** options, *open the **Preferences** dialog* and go to **XML** > **Import**. This page allows you to configure how empty values and `null` values are handled when they are encountered in imported database tables or Excel sheets. Also you can configure the format of date / time values recognized in the imported database tables or Excel sheets.

The following options are available:

- **Create empty elements for empty values** - If checked, an empty value from a database column or from a text file is imported as an empty element.
- **Create empty elements for null values** - If checked, `null` values from a database column are imported as empty elements.
- **Escape XML content** - Enabled by default, this option instructs Oxygen XML Author to escape the imported content to an XML-safe form.
- **Add annotations for generated XML Schema** - If checked, the generated XML Schema contains an annotation for each of the imported table columns. The documentation inside the annotation tag contains the remarks of the database columns (if available) and also information about the conversion between the column type and the generated XML Schema type.

The section **Date / Time Format** specifies the format used for importing date and time values from Excel spreadsheets or database tables and in the generated XML schemas. The following format types are available:

- **Unformatted** - If checked, the date and time formats specific to the database are used for import. When importing data from Excel a string representation of date or time values are used. The type used in the generated XML Schema is `xs:string`.
- **XML Schema date format** - If checked, the XML Schema-specific format ISO8601 is used for imported date / time data (`yyyy-MM-dd'T'HH:mm:ss` for `datetime`, `yyyy-MM-dd` for `date` and `HH:mm:ss` for `time`). The types used in the generated XML Schema are `xs:datetime`, `xs:date` and `xs:time`.
- **Custom format** - If checked, the user can define a custom format for timestamp, date, and time values or choose one of the predefined formats. A preview of the values is presented when a format is used. The type used in the generated XML Schema is `xs:string`.

Date / Time Patterns Preferences

Table 8: Pattern letters

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am / pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am / pm (0-11)	Number	0
h	Hour in am / pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

Pattern letters are usually repeated, as their number determines the exact presentation:

- *Text* - If the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available.
- *Number* - The number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount.
- *Year* - If the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a number.
- *Month* - If the number of pattern letters is 3 or more, the month is interpreted as text; otherwise, it is interpreted as a number.
- *General time zone* - Time zones are interpreted as text if they have names. For time zones representing a GMT offset value, the following syntax is used:

- *GMTOffsetTimeZone* - GMT Sign Hours : Minutes
- *Sign* - one of + or -
- *Hours* - one or two digits
- *Minutes* - two digits
- *Digit* - one of 0 1 2 3 4 5 6 7 8 9

Hours must be between 0 and 23, and Minutes must be between 00 and 59. The format is locale independent and digits must be taken from the Basic Latin block of the Unicode standard.

- *RFC 822 time zone*: The RFC 822 4-digit time zone format is used:

  - *RFC822TimeZone* - Sign *TwoDigitHours* Minutes
  - *TwoDigitHours* - a number of two digits

  TwoDigitHours must be between 00 and 23.

XML Signing Certificates Preferences

Oxygen XML Author provides two types of keystores for certificates used for digital signatures of XML documents: Java KeyStore (JKS) and Public-Key Cryptography Standards version 12 (PKCS-12). A keystore file is protected by a password. To configure a certificate keystore, *open the **Preferences** dialog* and go to **XML** > **XML Signing Certificates**. You can customize the following parameters of a keystore:

Figure 273: The Certificates Preferences Panel

- **Keystore type** - the type of keystore that Oxygen XML Author uses;
- **Keystore file** - the location of the imported file;
- **Keystore password** - the password that is used for protecting the privacy of the stored keys;
- **Certificate alias** - the alias used for storing the key entry (the certificate and / or the private key) inside the keystore;
- **Private key password** - the private key password of the certificate. Required only for JKS keystores;
- **Validate** - press this button to verify the configured keystore and the validity of the certificate.

Data Sources Preferences

To configure the **Data Sources** preferences, *open the **Preferences** dialog* and go to **Data Sources**.

Data Sources Preferences

To configure the **Data Sources** preferences, *open the **Preferences** dialog* and go to **Data Sources**. In this preferences page you can configure data sources and connections to relational databases as well as native XML databases. You can check the list of drivers (*http://www.oxygenxml.com/database_drivers.html*) available for the major database servers.
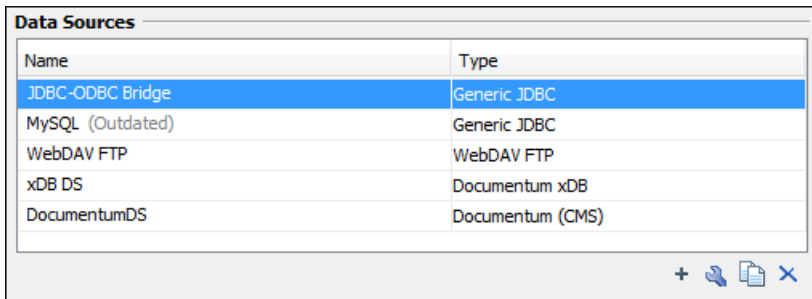
Figure 274: The Data Sources Preferences Panel

- **New** - opens the **Data Sources Drivers** dialog that allows you to configure a new database driver.
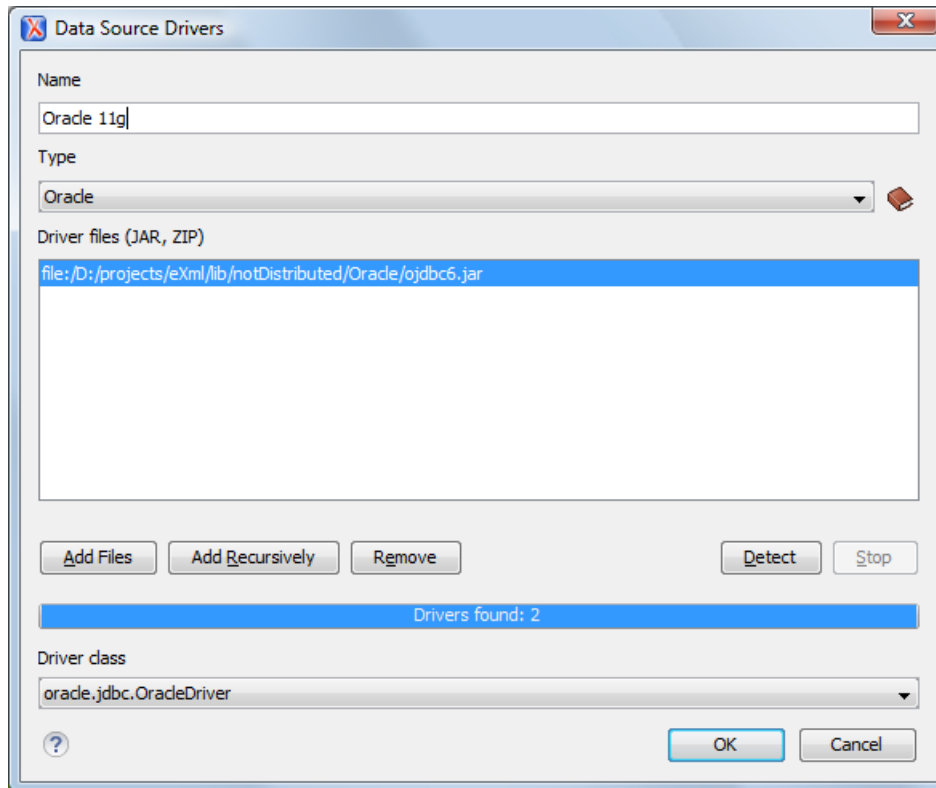
Figure 275: The Data Sources Drivers Dialog

The following options are available:

- **Name** - the name of the new data source driver that will be used for creating connections to the database;
- **Type** - selects the data source type from the supported driver types;
- **Help** - opens the User Manual at *the list of the sections* where the configuration of supported data sources is explained and the URLs for downloading the database drivers are specified;
- **Driver Class** - specifies the driver class for the data source driver;
- **Add** - adds the driver class library;
- **Remove** - removes the selected driver class library from the list;
- **Detect** - detects driver class candidates;
- **Stop** - stops the detection of the driver candidates;

- **Edit** - opens the **Data Sources Drivers** dialog for editing the selected driver. See above the specifications for the **Data Sources Drivers** dialog. In order to edit a data source, there must be no connections using that data source driver;

- **Delete** - deletes the selected driver. In order to delete a data source, there must be no connections using that data source driver;
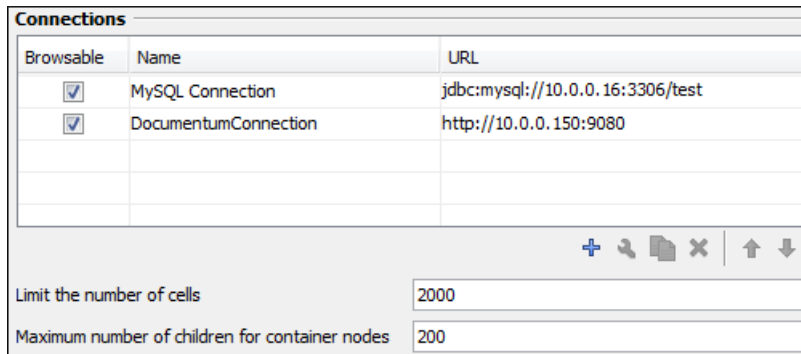
Figure 276: The Connections Preferences Panel

For performance issues, you can set the maximum number of cells that will be displayed in the **Table Explorer** view for a database table. Leave the field **Limit the number of cells** empty if you want the entire content of the table to be displayed. By default this field is set to 2,000. If a table having more cells than the value set here is displayed in the **Table Explorer** view, a warning dialog will inform you that the table is only partially shown.

In Oracle XML a container can hold millions of resources. If the node corresponding to such a container in the **Data Source Explorer** view would display all the contained resources at the same time the performance of the view would be very slow. To prevent such a situation only a limited number of the contained resources is displayed as child nodes of the container node. Navigation to other contained resources from the same container is enabled by the *Up* and *Down* buttons in the **Data Source Explorer** view. This limited number is set in the option **Maximum number of children for container nodes**. The default value is 200 nodes.

The actions of the buttons from the **Connections** panel are the following:

- **New** - opens the **Connection** dialog which has the following fields:
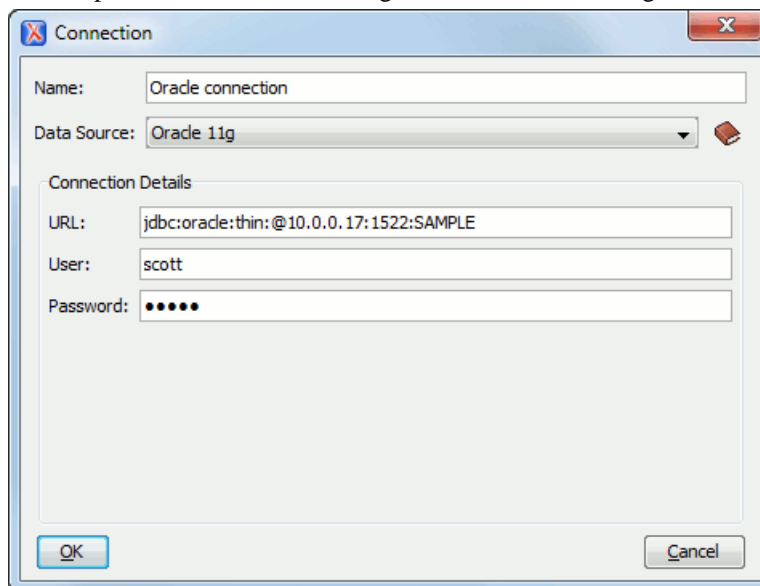
Figure 277: The Connection Dialog

- **Name** - the name of the new connection that will be used in transformation scenarios and validation scenarios;
- **Data Source** - allows selecting a data source defined in the **Data Source Drivers** dialog.

Depending upon the selected data source, you can set some of the following parameters in the **Connection details** area:

- **URL** - the URL for connecting to the database server;
- **User** - the user name for connecting to the database server;
- **Password** - the password of the specified user name;
- **Host** - the host address of the server;
- **Port** - the port where the server accepts the connection;
- **XML DB URI** - the database URI;
- **Database** - the initial database name;
- **Collection** - one of the available collections for the specified data source;
- **Environment home directory** - specifies the home directory (only for a Berkeley database);
- **Verbosity** - sets the verbosity level for output messages (only for a Berkeley database);
- **Use a secure HTTPS connection (SSL)** - allows you to establish a secure connection to an eXist database through the SSL protocol.

- **Edit** - opens the **Connection** dialog, allowing you to edit the selected connection. See above the specifications for the **Connection** dialog;
- **Duplicate** - creates a duplicate of the currently selected connection;
- **Delete** - deletes the selected connection.

Download Links for Database Drivers

You can find below the locations where you have to go to get the drivers necessary for accessing databases in Oxygen XML Author.

- **Berkeley DB XML database** - Copy the jar files from the Berkeley database install directory to the Oxygen XML Author 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 Author for *configuring a DB2 data source*.
- **eXist database** - Copy the jar files from the eXist database install directory to the Oxygen XML Author install directory as described in *the procedure* for configuring an eXist data source.
- **MarkLogic database** - Download the MarkLogic driver from *MarkLogic Community site*.
- **Microsoft SQL Server 2005 / 2008 database** - Both SQL Server 2005 and SQL Server 2008 are supported. For connecting to SQL Server 2005 you have to download the SQL Server 2005 JDBC driver file sqljdbc.jar from the *Microsoft website* and use it for *configuring an SQL Server data source*. For connecting to SQL Server 2008 you have to download the SQL Server 2008 JDBC 1.2 driver file sqljdbc_1.2\enu\sqljdbc.jar from the *Microsoft website* and use it for *configuring an SQL Server data source*. Please note that the SQL Server driver is compiled with a Java 1.6 compiler so you need to run Oxygen XML Author with a Java 1.6 virtual machine in order to use this driver.
- **Oracle 11g database** - Download the Oracle 11g JDBC driver called ojdbc6.jar from the *Oracle website* and use it for *configuring an Oracle data source*.
- **PostgreSQL 8.3 database** - Download the PostgreSQL 8.3 JDBC driver called postgresql-8.3-603.jdbc3.jar from the *PostgreSQL website* and use it for *configuring a PostgreSQL data source*.
- **Documentum xDb (X-Hive/DB) 10 XML database** - Copy the jar files from the Documentum xDb (X-Hive/DB) 10 database install directory to the Oxygen XML Author install directory as described in *the procedure* for configuring a Documentum xDb (X-Hive/DB) 10 data source.

Table Filters Preferences

to configure the **Table Filters** preferences, *open the Preferences dialog* and go to **Data Sources** > **Table Filters**.

Here you can choose which of the database table types will be displayed in the **Data Source Explorer** view.
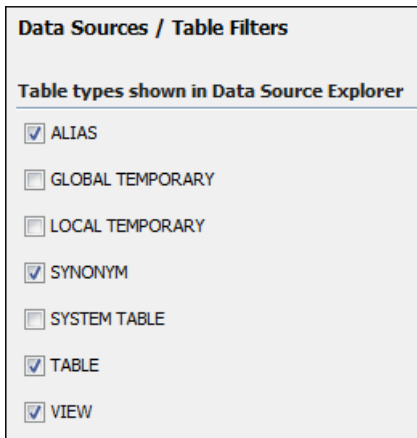
Figure 278: Table Filters Preferences Panel

SVN Preferences

To configure the **SVN** options, *open the **Preferences** dialog* and go to **SVN**. In this preferences page the user preferences for the embedded SVN client tool are configured. Some other preferences for the embedded SVN client tool can be set in the global files called `config` and `servers`, that is the files with parameters that act as defaults applied to all the SVN client tools that are used by the same user on his login account on the computer. These files can be opened for editing with the two edit actions available in the SVN client tool on the **Global Runtime Configuration** submenu of the **Options** menu.
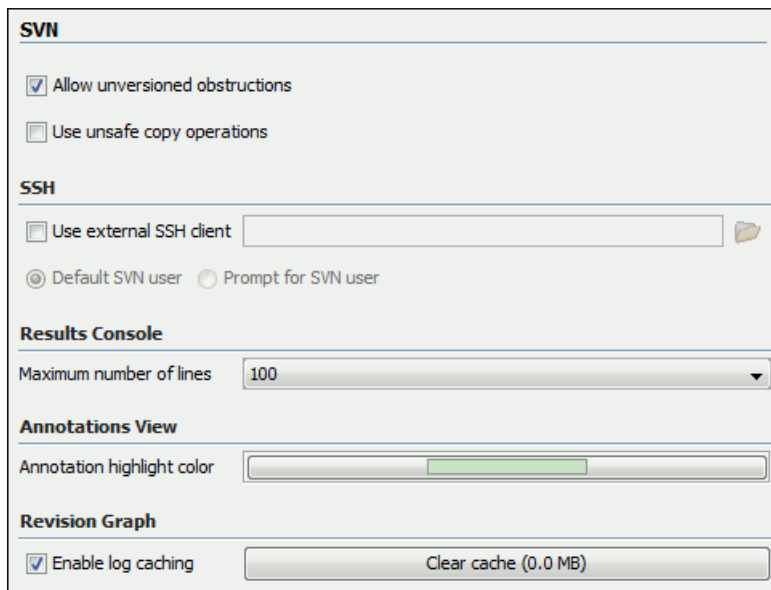
Figure 279: The SVN Preferences Panel

The SVN preferences are the following:

- **Enable symbolic link support** (*available only on Mac OS X and Linux*) - Apache Subversion™ has the ability to put a symbolic link under version control, via the usual SVN `add` command. The Subversion repository has no internal concept of a symbolic link, it stores a versioned symbolic link as an ordinary file with a `svn:special` property attached. The SVN client (on Unix) sees the property and translates the file into a symbolic link in the working copy.

  > **Note:** Windows file systems have no symbolic links, so a Windows client won't do any such translation: the object appears as a normal file.

If the symbolic link support is disabled then the versioned symbolic links, on Linux and OS X, are supported in the same way as on Windows, that is a text file instead of symbolic link is created.

> ⚠ **Important:** It is recommended to disable symbolic links support if you do not have versioned symbolic links in your repository, because the SVN operations will work faster. However, you should not disable this option when you do have versioned symbolic links in repository. In that case a workaround would be to refer to working copy by its real path, not a path that includes a symbolic link.

- **Allow unversioned obstructions** - controls how should be handled working copy resources being ignored / unversioned when performing an update operation and from the repository are incoming files with the same name, in the same location, that intersect with those being ignored / unversioned. If the option is enabled, then the incoming items will become BASE revisions of the ones already present in the working copy, and those present will be made versioned resources and will be marked as modified. Exactly as if the user first made the update operation and after that he / she modified the files. If the option is disabled, the update operation will fail when encountering files in this situation, possibly leaving other files not updated. By default, this option is enabled.
- **Use unsafe copy operations** - sometimes when the working copy is accessed through Samba and SVN client cannot make a safe copy of the committed file due to a delay in getting write permission the result is that the committed file will be saved with zero length (the content is removed) and an error will be reported. In this case this option should be selected so that SVN client does not try to make the safe copy.
- **HTTPS encryption protocols** - sets a specific encryption protocol to be used when the *application accesses a repository through HTTPS protocol*. You can choose one of the following values:

  - **SSLv3, TLSv1** (default value)
  - **SSLv3 only**
  - **TLSv1 only**

- **SSH** - specifies the command line for an external SSH client which will be used when connecting to a SVN+SSH repository. Absolute paths are recommended for the SSH client executable and the file paths given as arguments (if any). Depending on the SSH client used and your SSH server configuration you may need to specify in the command line the user name and / or private key / passphrase. Here you can also choose if the default user name (the same user name as the SSH client user) will be used for SVN repository operations or you should be prompted for a SVN user name whenever SVN authentication is required. For example on Windows the following command line uses the `plink.exe` tool as external SSH client for connecting to the SVN repository with SVN+SSH:

```
C:\plink-install-folder\plink.exe -l username -pw password -ssh -batch
host_name_or_IP_address_of_SVN_server
```

- **Results Console** - specifies the maximum number of lines displayed in the **Console** view. The default value is 1,000.
- **Annotations View** - sets the color used for highlighting in the editor panel all the changes contributed to a resource by the revision selected in *the **Annotations** view*.
- **Revision Graph** - enables caching for the action of computing a revision graph. When a new revision graph is requested one of the caches from the previous actions may be used which will avoid running the whole query again on the SVN server. If a cache is used it will finish the action much faster.

Working Copy Preferences

To configure the **Working Copy** preferences, *open the **Preferences** dialog* and go to **SVN** > **Working Copy**. The option that you are able to configure in this preferences page are specific to SVN working copies:

- **Working copies location** - Allows you to define a location where you keep your working copies. This location is automatically suggested when you checkout a new working copy.
- **Working copy administrative directory** - Allows you to customize the directory name where the svn entries are kept for each directory in the working copy.
- **When loading an old format working copy** - You can instruct Syncro SVN Client to do one of the following:

  - **Always ask** - You are notified when such a working copy is used and you are allowed to choose what action to be taken - to upgrade or not the format of the current working copy.
  - **Never upgrade** - Older format working copies are left untouched. No attempt to upgrade the format is made.

> **Note:** SVN 1.6 and older working copies still need to be upgraded before loading them.

- **Enable working copy caching** - If checked, the content of the working copies is cached for refresh operations.
- **Automatically refresh the working copy** - If checked, the working copy is refreshed from cache. Only the new changes (modifications with a date/time that follows the last refresh operation) are refreshed from disk. Disabled by default.
- **When synchronizing with repository** - The action that will be executed automatically after the **Synchronize** action. The possible actions are:

  - **Always switch to 'Modified' mode** - The **Synchronize** action is followed automatically by a switch to **Modified** mode of **Working Copy** view, if **All Files** mode is currently selected.
  - **Never switch to 'Modified' mode** - Keeps the currently selected view mode unchanged.
  - **Always ask** - The user is always asked if he wants to switch to **Modified** mode.

- **Application global ignores** - Allows setting file patterns that may include the `*` and `?` wildcards for unversioned files and folders that must be ignored when displaying the working copy resources in *the Working Copy view*. These patterns are case-sensitive. For example,`*.txt` matches `file.txt`, but does not match `file.TXT`.

Diff Preferences

To configure **Diff** options, *open the Preferences dialog* and go to **Diff**.
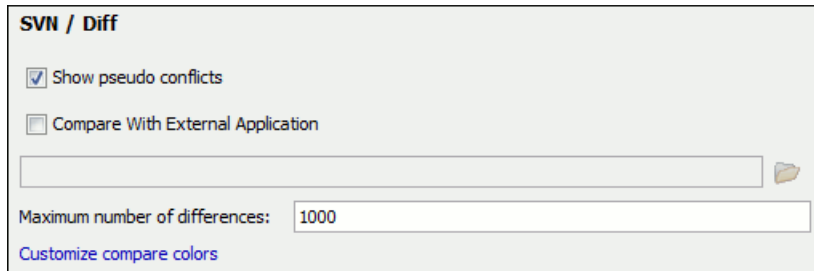
Figure 280: The SVN Diff Preferences Panel

The SVN diff preferences are the following:

- **Show pseudo conflicts** - specifies whether the *the Compare view* displays pseudo-conflicts . A pseudo-conflict occurs when two developers make the same change, for example when they both add or remove the same line of code.
- **Compare With External Application** - specifies an external application to be launched for compare operations in the following cases:

  - when two history revisions are compared;
  - when the working copy file is compared with a history revision;
  - when *a conflict is edited*.

  The parameters ${firstFile} and ${secondFile} specify the positions of the two compared files in the command line for the external diff application. The parameter ${ancestorFile} specifies the common ancestor (that is, the BASE revision of a file) in a three-way comparison: the working copy version of a file is compared with the repository version, with the BASE revision (the latest revision read from the repository by an Update or Synchronize operation) being the common ancestor of these two compared versions.

- **Maximum number of differences** - sets the maximum number of differences allowed in the view.

Messages Preferences

To configure the **Messages** options, *open the Preferences dialog* and go to **SVN** > **Messages**. This preferences page allows you to disable the following warning messages which may appear in the application:
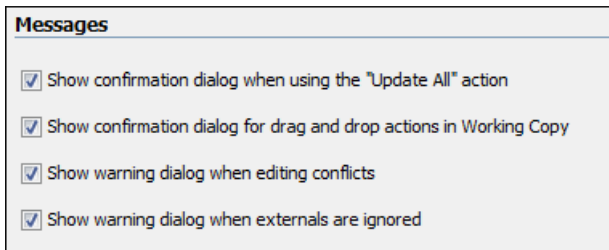
Figure 281: The Messages Preferences Panel

- **Show confirmation dialog when using the "Update All" action** - Allows you to avoid performing accidental update operations by requesting you to confirm them before execution.
- **Show confirmation dialog for drag and drop actions in Working Copy** - This option avoids doing a drag and drop when you just want to select multiple files in the Working Copy view.
- **Show warning dialog when editing conflicts** - When the **Edit Conflicts** action is executed, a warning dialog notifies you that the action overwrites the conflicted version of the file created by an update operation. The conflicted file is overwritten with the version of the same file which existed in the working copy before the update operation and then *proceeds with the visual editing of the conflicting file*.
- **Show warning dialog when "svn:externals" definitions are ignored** - A warning dialog is displayed when "svn:externals" definitions are ignored before performing any operation that updates resources of the working copy (like *Update* and *Override and Update*).

Diff Preferences

The Diff Preferences Page has sub-pages for configuring File Comparisons and Directory Comparisons.

Files Comparison Preferences

To configure the **Files Comparison** options, *open the **Preferences** dialog* and go to **Diff** > **Files Comparison**. This preferences page allows you to configure the following options:
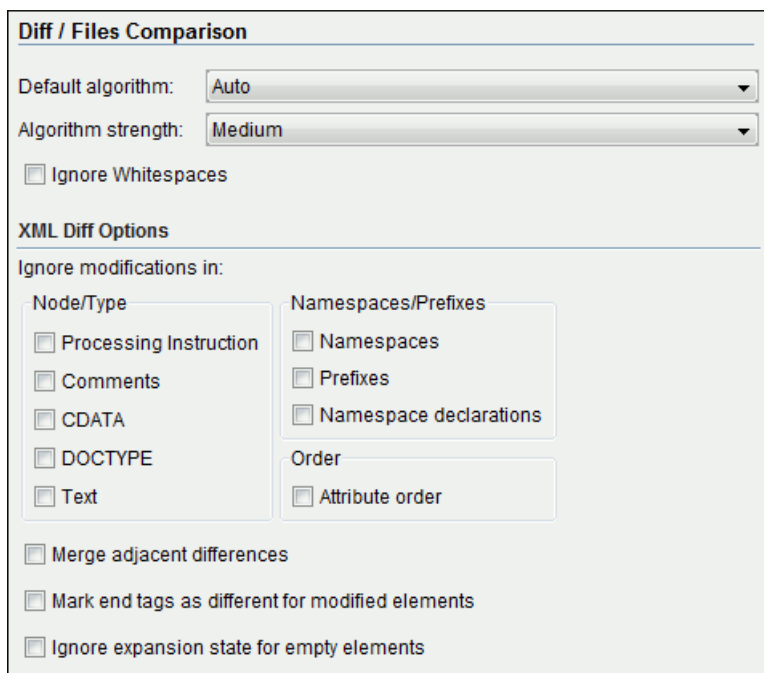
Figure 282: The Files Comparison Preferences Panel

- **Default algorithm** - the default algorithm used for comparing files. The following options are available:
  - **Auto** - automatic selection of the diff algorithm, based on the file content and its size;

- **Characters** - computes the differences at character level;
- **Words** - computes the differences at word level;
- **Lines** - computes the differences at line level;
- **Syntax aware** - for the file types known by Oxygen XML Author, this algorithm computes the differences taking into consideration the syntax (the specific types of tokens) of the documents;
- **XML Fast** - a diff algorithm well-suited for large XML documents (tens of MB). Sacrifices accuracy in favor of speed;
- **XML Accurate** - XML-tuned diff algorithm. It favors accuracy over speed.

- **Algorithm strength** - controls the amount of resources allocated to the application to perform the comparison. The algorithm stops searching more differences when reaches the maximum allowed resources. A dialog is shown when this limit is reached and partial results are displayed. Four settings are available: **Low**, **Medium** (default), **High** and **Very High**;
- **Ignore Whitespaces** - ignoring whitespaces means that before performing the comparison, the application normalizes the content (collapses any sequence of whitespace characters (space, newline or tab characters) into a single space) and trims its leading and trailing whitespaces;

  > **Note:** If the **Ignore Whitespaces** check box is selected, comparing the `a b` sequence with `a b`, Oxygen XML Author finds no differences, because after normalization, all whitespaces from the first sequence are collapsed into a single space character. However, when comparing `a b` with `ab` (no whitespace between `a` and `b`), Oxygen XML Author signals a difference.

- **XML Diff Options** - this set of options allows you to specify the types of XML nodes for which the differences will be ignored (will not be reported) by the **XML Fast** and **XML Accurate** algorithms;
- **Merge adjacent differences** - considers two adjacent differences as one when the differences are painted in the side-by-side editors. If unchecked, every difference is represented separately;
- **Mark end tags as different for modified elements** - end tags of modified elements are presented as differences too, otherwise only the start tags are presented as differences;
- **Ignore expansion state for empty elements** - empty elements in both expansion states are considered matched, that is `<element/>` and `<element></element>` are considered equal.

Appearance Preferences

To configure the **Files Comparison / Appearance** preferences, *open the **Preferences** dialog* and go to **Diff** > **Files Comparison** > **Appearance**. This preferences page offers the following options:
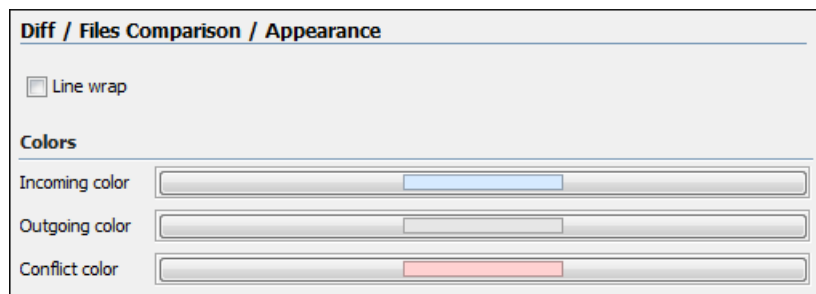
Figure 283: Files Comparison Appearance Preferences Panel

- **Line wrap** - wraps at the right margin of each panel the lines presented in the two diff panels, so no horizontal scrollbar is necessary;
- **Incoming color** - specifies the color used for incoming changes on the vertical bar, that shows the differences between the compared files;
- **Outgoing color** - specifies the color used for outgoing changes on the vertical bar, that shows the differences between the compared files;
- **Conflict color** - specifies the color used for conflicts on the vertical bar, that shows the differences between the compared files.

Directories Comparison Preferences

To configure the **Directories Comparison** preferences, *open the Preferences dialog* and go to **Diff** > **Directories Comparison**.
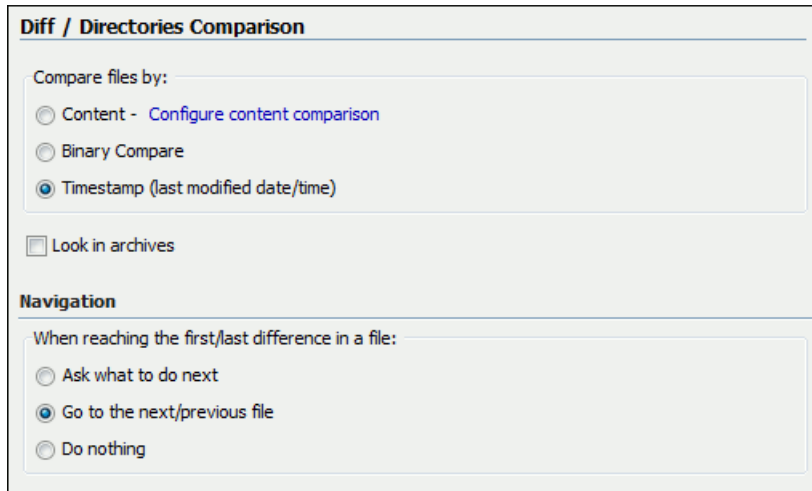
Figure 284: The Diff Preferences Panel

For the directories comparison, you can specify:

- **Compare files by** - Controls the method used for comparing two files:

  - **Content** - The file content is compared using the current *diff algorithm*. This option is applied for a pair of files only if that file type is associated with a built-in editor type (either associated by default or associated by the user when the user is prompted to do that on opening a file of that type for the first time).
  - **Binary Compare** - The files are compared at byte level.
  - **Timestamp (last modified date / time)** - The files are compared only by their last modified timestamp.

- **Look in archives** - If checked, *archives known by Oxygen XML Author* are considered directories and their content is compared just like regular files.
- **Navigation** - This options control the behaviour of the differences traversal actions (**Go to previous modification**, **Go to next modification**) when the first or last difference in a file is reached:

  - **Ask what to do next** - A dialog is displayed asking you to confirm that you want the application to display modifications from the previous or next file.
  - **Go to the next/previous file** - The application opens the next or previous file without waiting for your confirmation.
  - **Do nothing** - No further action is taken.

Appearance Preferences

to configure the **Directories Comparison / Appearance** options, *open the Preferences dialog* and go to **Diff** > **Directories Comparison** > **Appearance**.
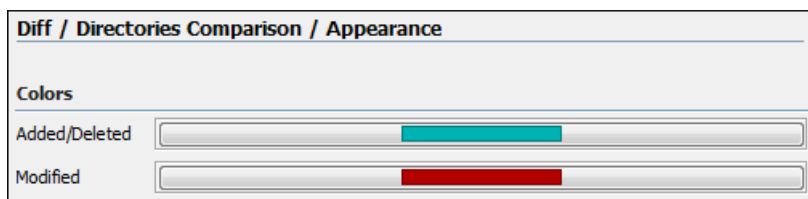
Figure 285: The Diff Appearance Preferences Panel

- **Added/Deleted** - Color used for marking added or deleted files and folders.
- **Modified** - Color used for marking modified files.

Archive Preferences

To configure **Archive** preferences, *open the **Preferences** dialog* and go to **Archive**.

The following options are available in the **Archive** preferences panel:

- **Archive backup options** - Controls if the application makes backup copies of the modified archives. The following options are available:

  - **Always create backup copies of modified archives** - When you modify an archive, its content is backed up.
  - **Never create backup copies of modified archives** - No backup copy is created.
  - **Ask for each archive once per session** - Once per application session for each modified archive, user confirmation is required to create the backup. This is the default setting.

    > **Note:** Backup files have the name `originalArchiveFileName.bak` and are located in the same folder as the original archive.

- **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 Author. You can edit an existing mapping or create a new one by associating your own list of extensions to an archive format.
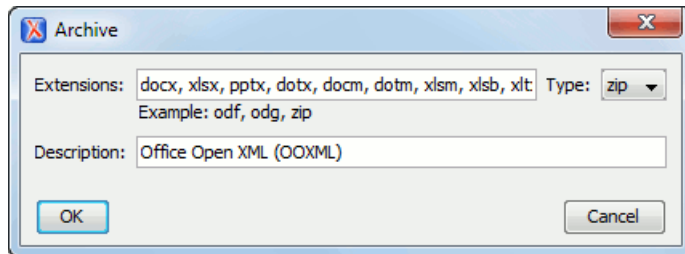
Figure 286: Edit Archive Extension Mappings

> **Important:** You have to restart Oxygen XML Author after removing an extension from the table in order for that extension to not be recognised anymore as an archive extension.

- **Store Unicode file names in Zip archives** - Use this option when you archive files that contain international (that is, non-English) characters in file names or file comments. If an archive is modified in any way with this option turned on, UTF-8 characters are used in the names of all files in the archive.

Plugins Preferences

You are able to add plugins that extend the functionality of Oxygen XML Author. The plugins are shipped as separate packages. To check for new plugins, go to *http://www.oxygenxml.com/oxygen_sdk.html*.

A plugin consists of a separate sub-folder in the `Plugins` folder of the Oxygen XML Author installation folder. This sub-folder must contain a valid `plugin.xml` file in accordance with the `plugin.dtd` file located in the `Plugins` folder.

Oxygen XML Author automatically detects and loads plugins installed correctly in the `Plugins` folder and displays them in the **Plugins** preferences page. To configure plugins, *open the **Preferences** dialog* and go to **Plugins**.

You can use the check-boxes in front of each plugin to enable or disable them. To display the properties of a plugin in the second section of the **Plugins** preferences page, click the name of the plugin.
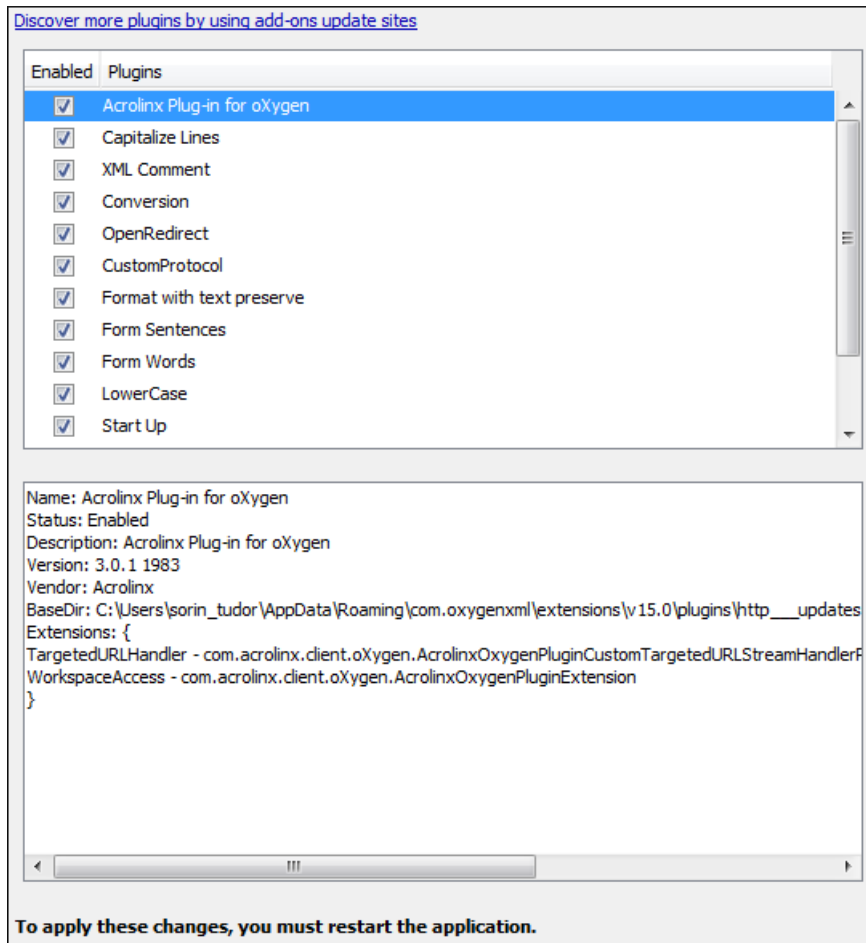
Figure 287: The Plugins Preferences Panel

Also, you are able to install a plugin as an add-on. For further details about this, go to *Deploying Add-ons*

External Tools Preferences

The **External Tools** preferences panel is opened from menu **Tools** > **External Tools** > **Configure...**.

A command-line tool can be started from the **External Tools** toolbar in the Oxygen XML Author user interface as if from the command line of the operating system shell. Such a tool must be configured first in this preferences panel. The parameters of an external tool are set in the following dialog:
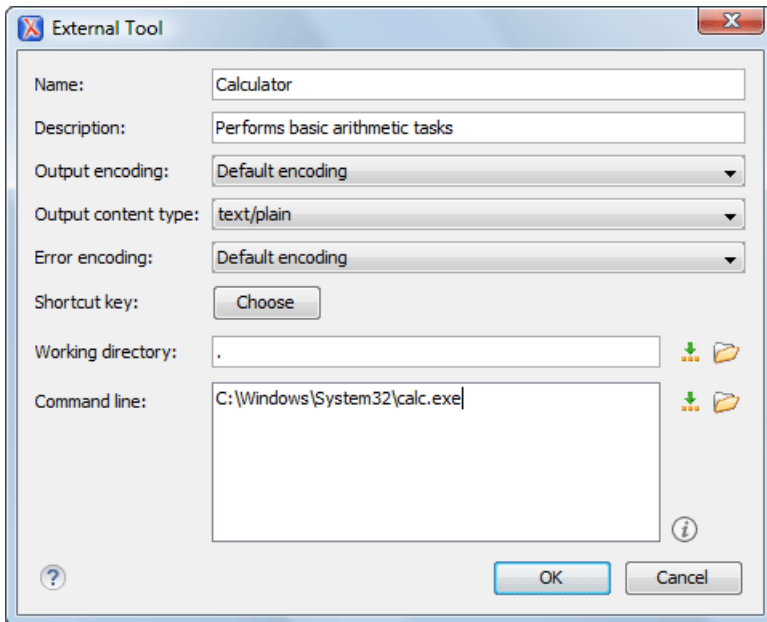
Figure 288: The External Tools Configuration Dialog

- **Name** - The name of the menu entry corresponding to this tool that will be displayed in the **Tools** > **External Tools** menu and in the **External Tools** toolbar.
- **Description** - The description of the tool displayed as tooltip where the tool name is used.
- **Output encoding** - The encoding of the output stream of the external tool that will be used by Oxygen XML Author to read the output of the tool.
- **Output content type** - A list of predefined content type formats that instruct Oxygen XML Author how to display the generated output. For example, setting the **Output content type** to text/xml enables the syntax coloring of XML output.
- **Error Encoding** - The encoding of the error stream of the external tool that will be used by Oxygen XML Author to read this error stream.
- **Shortcut key** - The keyboard shortcut that launches the external tool.
- **Working directory** - The directory the external tool will use to store intermediate and final results. Here you can use one of the following editor variables:

  - *${cfd}* - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
  - *${pd}* - Current project folder as file path. Usually the current folder selected in the Project View.
  - *${oxygenInstallDir}* - Oxygen XML Author installation folder as file path.
  - *${homeDir}* - The path (as file path) of the user home folder.
  - *${env(VAR_NAME)}* - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the *${system(var.name)}* editor variable.
  - *${system(var.name)}* - Value of the var.name Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as -Dvar.name=var.value. If you are looking for operating system environment variables, use the *${env(VAR_NAME)}* editor variable instead.
  - *${date(pattern)}* - Current date. The allowed patterns are equivalent to the ones in the *Java SimpleDateFormat class*. Example: yyyy-MM-dd;

    > **Note:** This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to *http://www.w3.org/TR/xmlschema-2/#date*. For details about xs:datetime, go to *http://www.w3.org/TR/xmlschema-2/#dateTime*.

- *${xpath_eval(expression)}* - Evaluates an XPath 3.0 expression. Depending on the context, the expression can be:

  - *static*, when executed in a non-XML context. For example, you can use such static expressions to perform string operations on other editor variables for composing the name of the output file in a transformation scenario's **Output** tab.

    Example:

    ```
    ${xpath_eval(upper-case(substring('${cfn}', 1, 4)))}
    ```

  - *dynamic*, when executed in an XML context. For example, you can use such dynamic expression in a code template or as a value of an author operation's parameter.

    Example:

    ```
    ${ask('Set new ID attribute', generic, '${xpath_eval(@id)}')}
    ```

- **Command line** - The command line that will start the external tool. Here you can use one of the following editor variables:

  - *${homeDir}* - The path (as file path) of the user home folder.
  - *${home}* - The path (as URL) of the user home folder.
  - *${cfn}* - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
  - *${cfne}* - Current file name with extension. The current file is the one currently opened and selected.
  - *${cf}* - Current file as file path, that is the absolute file path of the current edited document.
  - *${currentFileURL}* - Current file as URL, that is the absolute file path of the current edited document represented as URL.
  - *${cfd}* - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
  - *${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.
  - *${tsf}* - The transformation result file path. If the current opened file has an associated scenario which specifies a transformation output file, this variable expands to it.
  - *${pd}* - Current project folder as file path. Usually the current folder selected in the Project View.
  - *${pdu}* - Current project folder as URL. Usually the current folder selected in the Project View.
  - *${oxygenInstallDir}* - Oxygen XML Author installation folder as file path.
  - *${oxygenHome}* - Oxygen XML Author installation folder as URL.
  - *${frameworksDir}* - The path (as file path) of the `[OXYGEN_DIR]/frameworks`directory.
  - *${frameworks}* - The path (as URL) of the `[OXYGEN_DIR]` directory.
  - *${ps}* - Path separator, that is the separator which can be used on the current platform (Windows, OS X, Linux) between library files specified in the class path.
  - *${timeStamp}* - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
  - *${uuid}* - Universally unique identifier;An unique sequence of 32 hexadecimal digits generated by the Java *UUID* class.
  - *${id}* - Application-level unique identifier;A short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
  - *${afn}* - The file name (without parent directory and without file extension) of the zip archive that includes the current edited file.
  - *${afne}* - The file name (with file extension, for example `.zip` or `.epub`, but without parent directory) of the zip archive that includes the current edited file.
  - *${af}* - The local file path of the ZIP archive that includes the current edited document.
  - *${afu}* - The URL path of the ZIP archive that includes the current edited document.

- *${afd}* - The local directory path of the ZIP archive that includes the current edited document.
- *${afdu}* - The URL path of the directory of the ZIP archive that includes the current edited document.
- *${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}* - To prompt for values at runtime, use the *ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value'')* editor variable. You can set the following parameters:

  - `'message'` - The displayed message. Note the quotes that enclose the message.
  - `type` - Optional parameter. Can have one of the following values:

    - `url` - Input is considered an URL. Oxygen XML Author checks that the URL is valid before passing it to the transformation.
    - `password` - Input characters are hidden.
    - `generic` - The input is treated as generic text that requires no special handling.
    - `relative_url` - Input is considered an URL. Oxygen XML Author tries to make the URL relative to that of the document you are editing.

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

    - `combobox` - Displays a dialog that contains a non-editable combo-box.
    - `editable_combobox` - Displays a dialog that contains an editable combo-box.
    - `radio` - Displays a dialog that contains radio buttons.

  - `'default-value'` - optional parameter. Provides a default value in the input text box;

    Examples:

    - `${ask('message')}` - Only the message displayed for the user is specified.
    - `${ask('message', generic, 'default')}` - `'message'` is displayed, the type is not specified (the default is string), the default value is `'default'`.
    - `${ask('message', password)}` - `'message'` is displayed, the characters typed are masked with a circle symbol.
    - `${ask('message', password, 'default')}` - same as before, the default value is `'default'`.
    - `${ask('message', url)}` - `'message'` is displayed, the parameter type is URL.
    - `${ask('message', url, 'default')}` - same as before, the default value is `'default'`.

- *${dbgXML}* - The local file path to the XML document which is current selected in the Debugger source combo box (for tools started from the XSLT/XQuery Debugger).
- *${dbgXSL}* - The local file path to the XSL/XQuery document which is current selected in the Debugger stylesheet combo box (for tools started from the XSLT/XQuery Debugger).
- *${env(VAR_NAME)}* - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the *${system(var.name)}* editor variable.
- *${system(var.name)}* - Value of the `var.name` Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as `-Dvar.name=var.value`. If you are looking for operating system environment variables, use the *${env(VAR_NAME)}* editor variable instead.
- *${date(pattern)}* - Current date. The allowed patterns are equivalent to the ones in the *[Java SimpleDateFormat class](#)*. Example: yyyy-MM-dd;

  > **Note:** This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to *http://www.w3.org/TR/xmlschema-2/#date*. For details about xs:datetime, go to *http://www.w3.org/TR/xmlschema-2/#dateTime*.

- *${xpath_eval(expression)}* - Evaluates an XPath 3.0 expression. Depending on the context, the expression can be:

- *static*, when executed in a non-XML context. For example, you can use such static expressions to perform string operations on other editor variables for composing the name of the output file in a transformation scenario's **Output** tab.

  Example:

  ```
  ${xpath_eval(upper-case(substring('${cfn}', 1, 4)))}
  ```

- *dynamic*, when executed in an XML context. For example, you can use such dynamic expression in a code template or as a value of an author operation's parameter.

  Example:

  ```
  ${ask('Set new ID attribute', generic, '${xpath_eval(@id)}')}
  ```

Menu Shortcut Keys Preferences

To configure the **Menu Shortcut Keys** options, *open the Preferences dialog* and go to **Menu Shortcut Keys**. Alternatively you can go to **Options** > **Menu Shortcut Keys**. You can use this page to configure shortcut keys for the actions available in Oxygen XML Author. The shortcuts assigned to menu items are displayed in the below table.

Description	Category	Shortcut key
Find Next	Find	F3
Find Previous	Find	Shift+F3
Find/Replace	Find	Ctrl+F
Find/Replace in Files	Find	Ctrl+Shift+H
Go to	Find	Ctrl+L
Quick Find	Find	Alt+Shift+F
Close Other Folds	Folding	Ctrl+NumPad /
Collapse Child Folds	Folding	Ctrl+NumPad .
Expand All	Folding	Ctrl+NumPad *
Expand Child Folds	Folding	
Toggle Fold	Folding	
Decrease Editor Font	Font Size	Ctrl+NumPad -
Increase Editor Font	Font Size	Ctrl+NumPad +
Normal Editor Font	Font Size	Ctrl+NumPad-0
Array	Grid	
Append value as child	Grid Append Child	
Attribute	Grid Append Child	
CDATA	Grid Append Child	
Comment	Grid Append Child	
Doctype	Grid Append Child	
Doctype identifier	Grid Append Child	
Doctype subset	Grid Append Child	
Element	Grid Append Child	

Figure 289: The Menu Shortcut Keys Preferences Panel

The **Menu shortcut Keys** preferences page also contains the shortcuts that you define at *document type* level.

Note: A shortcut defined at *document type* level overwrites a default shortcut.

To find a specific action, you can use the filter to search through the **Description**, **Category**, and **Shortcut Key** columns:

- **Description** - this column provides a short description of the action;
- **Category** - this column provides a classification of the actions in categories for easier management. For example, you can distinguish the **Cut** operation available in the **Text** view from the one available in the **Tree** view by assigning them to different categories;
- **Shortcut key** - this column holds the combination of keys on your keyboard that launches the action. You can either double click a row of the **Shortcut key** column or press the **Edit** button to enter a new shortcut;
- **'Home' and 'End' keys are applied at line level** - Option available only on Mac OS X that controls the way the HOME and END keys are interpreted. If enabled, the default behavior of the Mac OS X HOME and END keys is overridden and the caret moves only on the current line.

File Types Preferences

Oxygen XML Author offers editing support for a wide variety of file types, but users are free to add new file extensions and associate them with the editor type which fits better. The associations set here between a file extension and the type of editor which opens the file for editing have precedence over the default associations available in *the File > New dialog*.

To configure the **File Types** options, *open the Preferences dialog* and go to **File Types**.
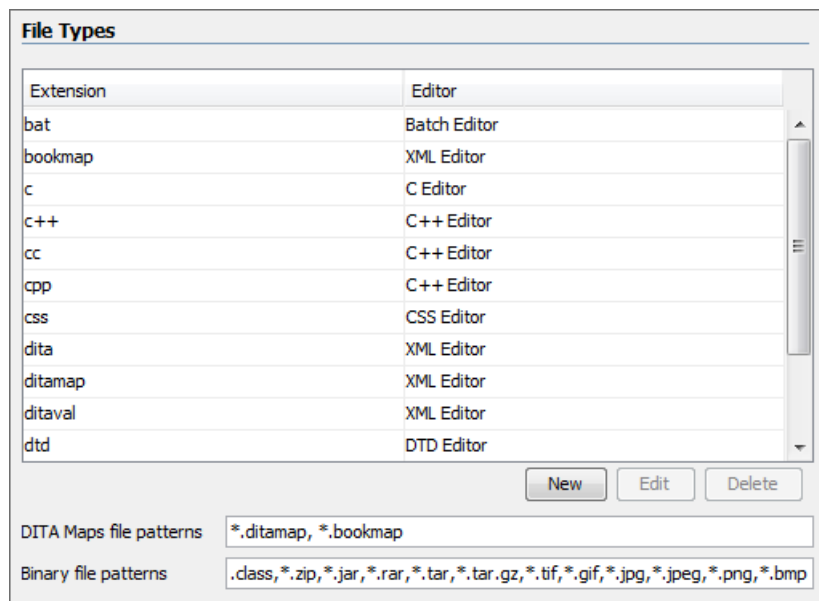
Figure 290: The File Types Preferences Panel

The table columns contain the following data:

- **Extension** - The extensions of the files that will be associated with an editor type.
- **Editor** - The type of editor which the extensions will be associated with. Some editors provide easy access to frequent operations via toolbars (e.g. XML editor, XSL editor, DTD editor) while others provide just a syntax highlight scheme (e.g. Java editor, SQL editor, Shell editor, etc.).

  If the editor set here is not one of the XML editors (XML editor, XSL editor, XSD editor, RNG editor, WSDL editor) then the encoding set in *the preference Encoding for non XML files* is used for opening and saving a file of this type.

You can also decide the files which will be handled as DITA maps when opened in Oxygen XML Author. If the files match the pattern that is specified in **DITA Maps file patterns** you will be prompted to open them in the **DITA Maps Manager** view.

The files that match the **Binary file patterns** patterns are handled as binary and opened in the associated system application. Also, they are excluded from the following actions available in the *Project view*: **File/Replace in Files**, **Check Spelling in Files**, **Validate**.

The Open/Find Resources Preferences Page

To configure the **Open/Find Resource** options, *open the Preferences dialog* and go to **Open/Find Resource**.

The following options are available in the **Open/Find Resource** preferences page:

- **Limit search results to** - specifies the maximum number of results that are displayed in the **Open/Find Resource** dialog or in the **Open/Find Resources** view.
- **Enable searching in content** - enable this option to activate the **In content** option of the **Open/Find Resource** view and **Open/Find Resource** dialog. In case this option is disabled you can only use the **Open/Find Resource** feature to search in file paths.
- **Ignore content of these files** - allows you to select specific directories, files, or file types that are ignored when you perform a search. For example, `*.txt` ignores all the `.txt` files, `*/topics/*` ignores all the files from the `topics` directory, regardless of their depth, and `file:/C:/tmp/*` ignores everything from the `tmp` directory.

  > **Note:** The specified pattern must begin with the desired protocol (in our case *file*) and also contain forward flash (`/`).

- **Index the content of remote resources** - controls the indexing of resources that are not local. For example, the resources referenced in a DITA Map opened from a remote server (from a CMS or from a WebDAV location) are not indexed by default. To index the content of these resources, enable this option.

  > **Note:** Enabling this option may lead to delays when the indexing is computed.

- **Exact matches** - this option controls whether the search results should exactly match the whole words that you introduce in the search field of the **Open/Find Resource** dialog or **Open/Find Resources** view.
- **Prefix matches** - this option controls whether the search results should match documents containing words starting with the searched terms. For instance searching for "pref page" will find documents containing also "preference pages".
- **Automatically join search terms using:** - select the default boolean operator that Oxygen XML Author applies when you perform a search. For example if the AND operator is selected and you enter "car assembly" in the dialog, the resulted documents must contain both of the words. If you choose OR, the resulted documents must contain one of the selected search terms. Results containing both words are promoted to the top of the list.
- **Enable XML-aware searching for files with size less than** - allows you to perform an *XML specific search* for XML elements and attributes.

  > **Note:** Enabling this option may slow down the indexing of your documents and increase the index size on the disk.

- **Stop Words** - list of comma separated stop words. Words added in this list are filtered out prior to processing a search query.

Custom Editor Variables Preferences

An editor variable is useful for making a transformation scenario, a validation scenario or an external tool independent of the file path on which the scenario / command line is applied. An editor variable is specified as a parameter in a transformation scenario, validation scenario or command line of an external tool. Such a variable is defined by a name, a string value and a text description. A custom editor variable is defined by the user and can be used in the same expressions as the built-in ones.
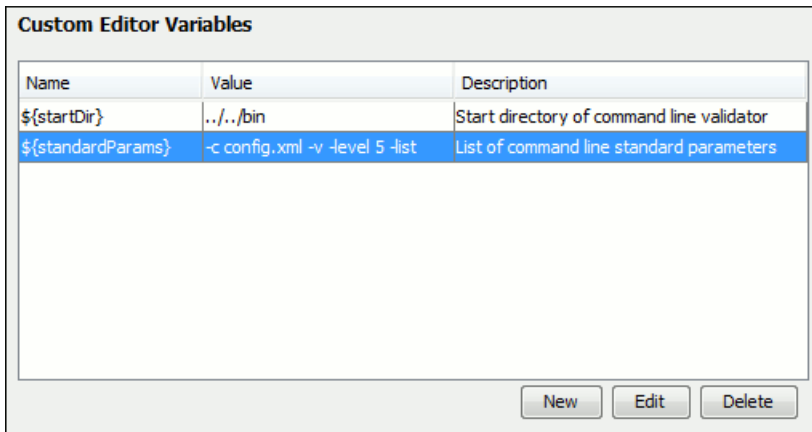
Figure 291: Custom Editor Variables

Network Connection Settings Preferences

This section presents the options available in the **Network Connection Settings** preferences pages.

Proxy Preferences

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 do not know the proxy parameters, consult your network administrator.

To configure the **Proxy** options, *open the Preferences dialog* and go to **Network Connection Settings** > **Proxy**. The following options are available:

- **Direct connection** - specifies whether the HTTP(S) connections go directly to the target host without going through a proxy server.
- **Use system settings** - specifies whether the HTTP(S) connections go through the proxy server set in the operating system. For example, on Windows the proxy settings are the ones that Internet Explorer uses.

  ⚠ **Attention:** The system settings for the proxy cannot be read correctly from the operating system on some Linux systems. The system settings option should work properly on Gnome based Linux systems, but it does not work on KDE based ones as the Java virtual machine does not offer the necessary support yet.

- **Manual proxy configuration** - specifies whether the HTTP(S) connections go through the proxy server specified in the **Address** and **Port** fields.
- **No proxy for** - specifies the hosts to which the connections must not go through a proxy server. A host needs to be written as a fully qualified domain name (e.g. *myhost.example.com*) or as a domain name (e.g. *example.com*). Use comma as a separator between multiple hosts.
- **User** - specifies the user necessary for authentication with the proxy server.
- **Password** - specifies the password necessary for authentication with the proxy server.
- **SOCKS Proxy** - In this section you set the host and port of a SOCKS proxy through which all the connections pass. If the **Address** field is empty the connections use no SOCKS proxy.

Using an automatic proxy configuration script (PAC)

If you have set up the path to an automatic proxy configuration script in your system (IE for Windows) Oxygen XML Author cannot detect this setting out of the box.

You can create a new folder `[OXYGEN_DIR]\lib\endorsed` in which you should copy two additional Java libraries: `deploy.jar` and `plugin.jar`. These libraries can be found in the folder `[OXYGEN_DIR]\jre\lib` if the application comes with a bundled Java VM or otherwise in the Java VM installation used to run the application.

HTTP(S)/WebDAV Preferences

To set the **HTTP(S)/WebDAV** preferences , *open the **Preferences** dialog* and go to **Network Connection Settings** > **HTTP(S)/WebDAV**. The following options are available:

* **Internal Apache HttpClient Version** - Oxygen XML Author uses the Apache HttpClient to establish connections to HTTP servers. To enable Oxygen XML Author to benefit from particular sets of features provided by different versions, you may choose between v3 and v4.

  > 📄 **Note:** For a full list of features, go to *http://hc.apache.org/httpclient-3.x/* and *http://hc.apache.org/httpcomponents-client-ga/*

* **Maximum number of simultaneous connections per host** - Defines the maximum number of simultaneous connections established by the application with a distinct host. Servers might consider multiple connections opened from the same source to be a **Denial of Service** attack. You can avoid that by lowering the value of this option.

  > 📄 **Note:** This option accepts a minimum value of 5.

* **Read Timeout (seconds)** - The period in seconds after which the application considers that an HTTP server is unreachable if it does not receive any response to a request sent to that server.
* **Enable HTTP 'Expect: 100-continue ' handshake for HTTP/1.1 protocol** - Activates *Expect: 100-Continue* handshake. The purpose of the *Expect: 100-Continue* handshake is to allow a client that is sending a request message with a request body to determine if the origin server is willing to accept the request (based on the request headers) before the client sends the request body. The use of the *Expect: 100-continue* handshake can result in noticeable performance improvement when working with databases. The *Expect: 100-continue* handshake should be used with caution, as it may cause problems with HTTP servers and proxies that do not support the HTTP/1.1 protocol.
* **Use the 'Content-Type' header field to determine the content type** - When checked, Oxygen XML Author tries to determine a resource type using the **Content-Type** header field. This header indicates the *Internet media type* of the message content, consisting of a type and subtype, for example:

```
Content-Type: text/xml
```

  When unchecked, the resource type is determined by analyzing its extension. For example, a file ending in `.xml` is considered to be an XML file.

* **Automatic retry on recoverable error** - If enabled, if an HTTP error occurs when Oxygen XML Author communicates with a server via HTTP, for example sending / receiving a SOAP request / response to / from a Web services server, and the error is recoverable, Oxygen XML Author tries to send again the request to the server.
* **Automatically accept a security certificate, even if invalid** - When enabled, the HTTPS connections that Oxygen XML Author attempts to establish will accept all security certificates, even if they are invalid.

  > ⚠ **Important:** By accepting an invalid certificate, you accept at your own risk a potential security threat, since you cannot verify the integrity of the certificate's issuer.

* **Encryption protocols (SVN Client only)** - this option is available only if you run the application with Java 1.6 or older. Sets a specific encryption protocol used when a repository is accessed through HTTPS protocol. You can choose one of the following values:

  * **SSLv3, TLSv1** (default value);
  * **SSLv3 only**;
  * **TLSv1 only**.

* **Lock WebDAV files on open** - If checked, the files opened through WebDAV are locked on the server so that they cannot be edited by other users while the lock placed by the current user still exists on the server.

(S)FTP Preferences

To configure the **(S)FTP** options, *open the **Preferences** dialog* and go to **Network Connection Settings** > **(S)FTP**. You can customize the following options:
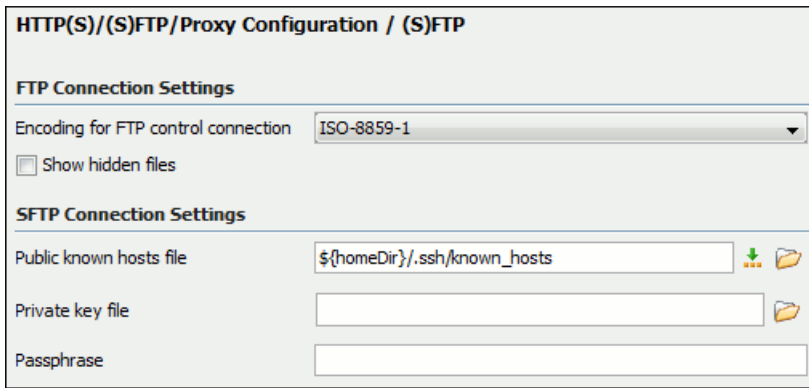
Figure 292: The (S)FTP Configuration Preferences Panel

- **Encoding for FTP control connection** - The encoding used to communicate with FTP servers: either ISO-8859-1 or UTF-8. If the server supports the UTF-8 encoding Oxygen XML Author will use it for communication. Otherwise it will use ISO-8859-1.
- **Public known hosts file** - File containing the list of all SSH server host keys that you have determined are accurate. The default value is `${homeDir}/.ssh/known_hosts`.
- **Private key file** - The path to the file containing the private key used for the private key method of authentication of the secure FTP (SFTP) protocol. 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*.

SSH Preferences

To configure the **SSH** options, *open the **Preferences** dialog* and go to **Connection settings** > **SSH**. The following options are available:

- **Use external SSH client** - Allows you to specify the command line for an external SSH client which is used when connecting to a SVN+SSH repository. Absolute paths are recommended for the SSH client executable and the file paths given as arguments (if any). Depending on the SSH client used and your SSH server configuration, specify the user name and / or private key / passphrase in the command line. Here you can also choose if the default user name (the same user name as the SSH client user) will be used for SVN repository operations or you should be prompted for a SVN user name whenever SVN authentication is required. For example, on Windows the following command line uses the `plink.exe` tool as external SSH client for connecting to the SVN repository with SVN+SSH:

```
C:\plink-install-folder\plink.exe -l username -pw password -ssh -batch
host_name_or_IP_address_of_SVN_server
```

XML Structure Outline Preferences

To configure the **XML Structure Outline** options, *open the **Preferences** dialog* and go to **XML Structure Outline** and contains the following preferences:

- **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.

Views Preferences

To configure the view options, *open the **Preferences** dialog* and go to **Views** and contains the following preferences:

- **Enable drag-and-drop in Project view** - Enables the drag and drop support in **Project** view. It should be disabled only if there is a possibility to accidentally change the structure of the project by such drag and drop actions.
- **Format and indent on save** - If selected, Oxygen XML Authorwill run the action **Format and Indent** (pretty-print) on saving a document edited in the tree editor.
- **Maximum number of lines** - Sets the maximum number of lines in the **Information** view.

Messages Preferences

To configure the **Messages** options, *open the Preferences dialog* and go to **Messages**. This preferences page allows you to disable the following warning messages which may appear in the application:

- **Show confirmation dialog when moving resources** - displays a confirmation dialog box when you move a resource in the Project view, DB Explorer, and Archive Browser. In the **Confirm** dialog box you are able to choose not to see this dialog in the future;
- **Show warning when adding resources already included in the project** - displays a dialog box that warns you in case you try to add already existing files in your project;
- **Show BIDI limit warning** - displays a warning dialog box when the opened file which contains bidirectional characters is too large and bidirectional support is disabled;
- **Show warning message when changing the text orientation in the editor** - displays a warning dialog box when you change the text orientation in the editor;
- **Show warning when editing long expressions in the XPath toolbar** - displays an information dialog box that asks you whether you want to use the *XPath/XQuery Builder* view when you edit long XPath expressions;
- **Show MathFlow recommendation** - displays an information dialog box that recommends using the **MathFlow Editor** to edit MathML equations;
- **Show SFTP certificate warning dialog** - displays a warning dialog box each time the authenticity of the SFTP server host cannot be established;
- **Convert DB Structure to XML Schema** - when tables from a database schema are selected in the **Select Database Table** dialog, after the *Convert DB Structure to XML Schema* is invoked, and another database schema is expanded, a confirmation is needed because the previous selection is discarded. This option controls whether you are always asked about the next action, the other database schema is always expanded without asking you, or it is never expanded.

Importing / Exporting Global Options

The import/export operations are located in the **Options** menu. These operations allow you to load or save global preferences as an XML file. You can use this file to reload saved options both on your computer and on others.

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

Reset Global Options

Restores the preference to the factory defaults, or to the *custom defaults*, if they are defined.

Import Global Options

Allows you to import a set of *Global Options* from an *options file* ;

Export Global Options

Allows you to export the entire set of *Global Options* to an options file.

Oxygen XML Author automatically stores your options in an `options` file. Depending on the platform you are using, this file is located in the following directories:

- [user-home-folder]\Application Data\com.oxygenxml for Windows XP
- [user-home-folder]\AppData\Roaming\com.oxygenxml for Windows Vista/7
- [user-home-folder]/Library/Preferences/com.oxygenxml for OS X
- [user-home-folder]/.com.oxygenxml for Linux

The name of the `options` file of Oxygen XML Author 16.1 is oxyAuthorOptionsSa16.1.xml.

Project Level User Options

You are able to set the Oxygen XML Author options either globally, or to bound them to a specific project by choosing the appropriate setting in the preferences pages:
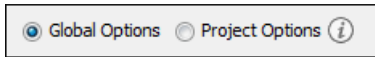
Figure 293: Controlling the Storage of the Preferences

By default, **Global Options** is selected, meaning that the options are stored on your computer, in your user home folder.

When you choose **Project Options**, the preferences are stored in the project file and can be shared with other users. For instance, if your project file and other files and folders are saved on a version control system (like SVN, CVS or Source Safe) or a shared folder, your team can use the same options/scenarios you have chosen to store in the project file.

> 📋 **Note:** Some pages do not have the **Project Options** switch, since the options they host might contain sensitive data (like passwords, for example), unsuitable for sharing with other users.

> You may decide that the default schema associations and catalogs must be shared with other team members. To do this, *open the **Preferences** dialog* and go to **XML** > **XML Catalog** and select **Project Options**. Now all the options set in the **XML Catalog** page are saved in your current project. At a later time, you can drop a preferences panel from being stored into the project by selecting **Global Options**.

The same mechanism is applied for saving transformation and validation scenarios.

Reset Global Options

Options > **Reset Global Options**

The project level preferences are not changed by this action.

The list of transformation scenarios will be reset to the default scenarios.

Customizing Default Options

Oxygen XML Author has an extensive set of options that you can configure. When Oxygen XML Author in installed, these options are set to default values. You can provide a different set of default values for an installation using an *options file*.

Creating an *options file*

To create an *options file*:

1. Open Oxygen XML Author. You may wish to use a fresh install for this procedure, to make sure that you do not copy personal option settings to the group.
2. *Open the **Preferences** dialog* .
3. Go through the options and set them to the desired defaults. Make sure that you are setting global options, not project options in each page.
4. Close the **Preferences** dialog.
5. Go to **Options** > **Export Global Options** and create an options file.
6. Go to back to the main preferences page and click **Export Global Options** to create an options file.

If you want to control exactly which options page will be stored in the default options file, you can choose to attach them to a *project file (.xpr file extension)* by following these steps:

• In *the Project view* create a project or open an already existing one.
• Switch to *Project Options* each **Preferences** page which contains settings of interest to you. All explicitly set values will be saved in the project file after you press either of the **OK** or **Apply** buttons.

> **Note:** Some pages do not have the **Project Options** switch, since the options they host might contain sensitive data (like passwords, for example), unsuitable for sharing with other users.

> **Note:** If you store the options to a *project file (.xpr file extension)* the file extension must be preserved as such (`xpr`) when the configuration file is distributed to other users.

Providing Default Option Values

Use either one of the following ways to configure an Oxygen XML Author installation to use customized default options from an XML configuration file:

• Set the path to the options file as the value of the `com.oxygenxml.default.options` *system property*.

  The path must be specified with an URL or a file path relative to the application installation folder:

  ```
  -Dcom.oxygenxml.default.options=options/default.xml
  ```

  > **Note:** If you are using the *Java Webstart distribution*, edit the `.jnlp file` that launches the application and set the `com.oxygenxml.default.options` parameter using a `property` element, for example:
  >
  > ```
  >  <property name="oxy:com.oxygenxml.default.options"
  > value="http://host/path/to/default-options.xml"/>
  > ```

• In the [OXYGEN_DIR] installation folder, create a folder called `preferences`. Copy the options file in the [OXYGEN_DIR]/preferences folder.

  > **Note:** The same procedure applies to a *Java Webstart distribution*.

> **Note:** Make sure that the options configuration file has either the `.xml` extension (for example: `default-options.xml`) or an `.xpr` extension depending on the way in which it was created (from the global options or saved at project level).

Scenarios Management

You can import, export, and reset the global transformation and validation scenarios using the following actions:

• To load a set of transformation scenarios from a properties file, go to **Options** > **Import Global Transformation Scenarios**
• To store a set of transformation scenarios in a properties file, go to **Options** > **Export Global Transformation Scenarios**
• To load a set of validation scenarios from a properties file, go to**Options** > **Import Global Validation Scenarios**
• To store all the global (not project-level) validation scenarios in a properties file, go to **Options** > **Export Global Validation Scenarios**

The **Export Global Transformation Scenarios** and **Export Global Validation Scenarios** options are used to store all the scenarios in a separate properties file. Associations between document URLs and scenarios are also saved in this file. You can load the saved scenarios using the **Import Global Transformation Scenarios** and **Import Global Validation Scenarios** actions. To distinguish the existing scenarios and the imported ones, the names of the imported scenarios contain the word **import**.

Editor Variables

An editor variable is a shorthand notation for context-dependent information, like a file or folder path, a time-stamp, or a date. It is used in the definition of a command (for example the input URL of a transformation, the output file path of a transformation, the command line of an external tool) to make a command or a parameter generic and reusable with other input files. When the same command is applied to different files, the notation is expanded at the execution of the command so that the same command has different effects depending on the actual file.

You can use the following editor variables in Oxygen XML Author commands of external engines or other external tools, in transformation scenarios, validation scenarios, and Author operations:

- *${oxygenHome}* - Oxygen XML Author installation folder as URL.
- *${oxygenInstallDir}* - Oxygen XML Author installation folder as file path.
- *${framework}* - The path (as URL) of the current framework, as part of the `[OXYGEN_DIR]/frameworks` directory.
- *${framework(fr_name)}* - The path (as URL) of the `fr_name` framework.
- *${frameworkDir(fr_name)}* - The path (as file path) of the `fr_name` framework.

  > **Note:** Because multiple frameworks might have the same name (although it is not recommended), for both *${framework(fr_name)}* and *${frameworkDir(fr_name)}* editor variables Oxygen XML Author employs the following algorithm when searching for a given framework name:
  >
  > - all frameworks are sorted, from high to low, according to their ***Priority*** setting from the framework configuration
  > - next, if the two or more frameworks have the same name and priority, a further sorting based on the **Storage** setting is made, in the exact following order:
  >   - frameworks stored in the internal Oxygen XML Author options
  >   - additional frameworks added in the ***Locations*** *preferences page*
  >   - frameworks installed using the add-ons support
  >   - frameworks found in the *main frameworks location* (**Default** or **Custom**)

- *${frameworks}* - The path (as URL) of the `[OXYGEN_DIR]` directory.
- *${frameworkDir}* - The path (as file path) of the current framework, as part of the `[OXYGEN_DIR]/frameworks` directory.
- *${frameworksDir}* - The path (as file path) of the `[OXYGEN_DIR]/frameworks`directory.
- *${home}* - The path (as URL) of the user home folder.
- *${homeDir}* - The path (as file path) of the user home folder.
- *${pdu}* - Current project folder as URL. Usually the current folder selected in the Project View.
- *${pd}* - Current project folder as file path. Usually the current folder selected in the Project View.
- *${pn}* - Current project name.
- *${cfdu}* - Current file folder as URL, that is the path of the current edited document up to the name of the parent folder, represented as a URL.
- *${cfd}* - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- *${cfn}* - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- *${cfne}* - Current file name with extension. The current file is the one currently opened and selected.
- *${cf}* - Current file as file path, that is the absolute file path of the current edited document.
- *${cfu}* - The path of the current file as a URL. The current file is the one currently opened and selected.
- *${af}* - The local file path of the ZIP archive that includes the current edited document.
- *${afu}* - The URL path of the ZIP archive that includes the current edited document.
- *${afd}* - The local directory path of the ZIP archive that includes the current edited document.
- *${afdu}* - The URL path of the directory of the ZIP archive that includes the current edited document.

- *${afn}* - The file name (without parent directory and without file extension) of the zip archive that includes the current edited file.
- *${afne}* - The file name (with file extension, for example `.zip` or `.epub`, but without parent directory) of the zip archive that includes the current edited file.
- *${currentFileURL}* - Current file as URL, that is the absolute file path of the current edited document represented as URL.
- *${ps}* - Path separator, that is the separator which can be used on the current platform (Windows, OS X, Linux) between library files specified in the class path.
- *${timeStamp}* - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- *${caret}* - The position where the caret is inserted. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- *${selection}* - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** operations, or in a selection plugin.
- *${id}* - Application-level unique identifier;A short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
- *${uuid}* - Universally unique identifier;An unique sequence of 32 hexadecimal digits generated by the Java *UUID* class.
- *${env(VAR_NAME)}* - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the *${system(var.name)}* editor variable.
- *${system(var.name)}* - Value of the `var.name` Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as `-Dvar.name=var.value`. If you are looking for operating system environment variables, use the *${env(VAR_NAME)}* editor variable instead.
- *${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}* - To prompt for values at runtime, use the *ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value'')* editor variable. You can set the following parameters:

  - `'message'` - The displayed message. Note the quotes that enclose the message.
  - `type` - Optional parameter. Can have one of the following values:

    - `url` - Input is considered an URL. Oxygen XML Author checks that the URL is valid before passing it to the transformation.
    - `password` - Input characters are hidden.
    - `generic` - The input is treated as generic text that requires no special handling.
    - `relative_url` - Input is considered an URL. Oxygen XML Author tries to make the URL relative to that of the document you are editing.

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

    - `combobox` - Displays a dialog that contains a non-editable combo-box.
    - `editable_combobox` - Displays a dialog that contains an editable combo-box.
    - `radio` - Displays a dialog that contains radio buttons.

  - `'default-value'` - optional parameter. Provides a default value in the input text box;

    Examples:

    - `${ask('message')}` - Only the message displayed for the user is specified.
    - `${ask('message', generic, 'default')}` - `'message'` is displayed, the type is not specified (the default is string), the default value is `'default'`.
    - `${ask('message', password)}` - `'message'` is displayed, the characters typed are masked with a circle symbol.
    - `${ask('message', password, 'default')}` - same as before, the default value is `'default'`.
    - `${ask('message', url)}` - `'message'` is displayed, the parameter type is URL.
    - `${ask('message', url, 'default')}` - same as before, the default value is `'default'`.

- *${date(pattern)}* - Current date. The allowed patterns are equivalent to the ones in the *Java SimpleDateFormat class*. Example: yyyy-MM-dd;

  > 📄 **Note:** This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to *http://www.w3.org/TR/xmlschema-2/#date*. For details about xs:datetime, go to *http://www.w3.org/TR/xmlschema-2/#dateTime*.

- *${dbgXML}* - The local file path to the XML document which is current selected in the Debugger source combo box (for tools started from the XSLT/XQuery Debugger).
- *${dbgXSL}* - The local file path to the XSL/XQuery document which is current selected in the Debugger stylesheet combo box (for tools started from the XSLT/XQuery Debugger).
- *${tsf}* - The transformation result file path. If the current opened file has an associated scenario which specifies a transformation output file, this variable expands to it.
- *${dsu}* - The path of the detected schema as an URL for the current validated XML document.
- *${ds}* - The path of the detected schema as a local file path for the current validated XML document.
- *${cp}* - Current page number. Used to display the current page number on each printed page in the **Editor / Print** Preferences page.
- *${tp}* - Total number of pages in the document. Used to display the total number of pages on each printed page in the **Editor / Print** Preferences page.
- *${xpath_eval(expression)}* - Evaluates an XPath 3.0 expression. Depending on the context, the expression can be:

  - *static*, when executed in a non-XML context. For example, you can use such static expressions to perform string operations on other editor variables for composing the name of the output file in a transformation scenario's **Output** tab.

    Example:

    ```
    ${xpath_eval(upper-case(substring('${cfn}', 1, 4)))}
    ```

  - *dynamic*, when executed in an XML context. For example, you can use such dynamic expression in a code template or as a value of an author operation's parameter.

    Example:

    ```
    ${ask('Set new ID attribute', generic, '${xpath_eval(@id)}')}
    ```

- *${i18n(key)}* - Editor variable used only at document type/framework level to allow translating names and descriptions of Author actions in multiple actions. More details here:*Localizing Frameworks* on page 348

Custom Editor Variables

An editor variable can be created by the user and included in any user defined expression where a built-in editor variable is also allowed. For example a custom editor variable may be necessary for configuring the command line of an external tool, the working directory of a custom validator, the command line of a custom XSLT engine, a custom FO processor, etc. All the custom editor variables are listed together with the built-in editor variables, for example when editing the working folder or the command line of an external tool or of a custom validator, the working directory, etc.

Creating a custom editor variable is very simple: just specify the name that will be used in user defined expressions, the value that will replace the variable name at runtime and a textual description for the user of that variable.

You can configure the custom editor variables in the **Preferences** page.

Configure Toolbars

You can configure the toolbars that appear in the current editor mode (Author, Text, Grid) using the **Configure toolbars** dialog that is opened from menu **Window** > **Configure toolbars ...**.
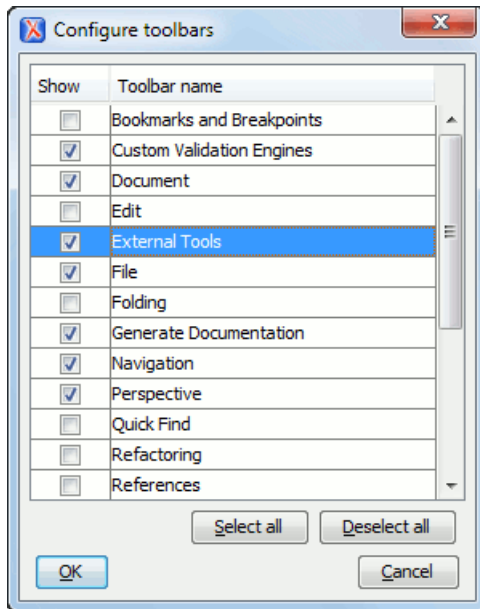
Figure 294: Configure Toolbars Dialog

Custom System Properties

A number of Java system properties can be set in the application to influence its behavior:

Table 9: Custom System Properties

Property	Allowed values	Purpose
com.oxygenxml.disable.http.protocol.handlers	true/false (false by default)	By default Oxygen XML Author uses for HTTP(S) connections the open source Apache HTTP Client software. If you disable this feature, the default Java SUN protocols HTTP(S) will be used instead. You will also lose **WebDAV** support and probably other related features.
com.oxygenxml.default.options	An URL-like relative or absolute path.	Provides the path to an XML file containing default application options. Read this topic for more details: *Customizing Default Options* on page 802.
com.oxygenxml.customOptionsDir	A file system absolute path pointing to a folder.	Sets a different folder which will be used by the application to load and save preference files. The default place where the options are saved varies according to the operating system: *Importing / Exporting Global Options* on page 801
com.oxygenxml.ApplicationDataFolder	A file system absolute path pointing to a folder.	When the application runs on Windows you can set this property to it to change the place where the application considers that the *APPDATA* folder is located.
com.oxygenxml.editor.frameworks.url	An URL-like absolute path.	Changes the folder where the application considers that the main frameworks/document types are installed. It has the same effect as

Property	Allowed values	Purpose
		changing the custom frameworks directory value in the *Locations Preferences* on page 729 Preferences page.
com.oxygenxml.MultipleInstances	true/false	If **true** allows multiple instances of the application to be started. By default it is **true** on Linux and **false** on Windows.
com.oxygenxml.xep.location	A file system absolute path pointing to a folder.	Points to a folder where RenderX XEP is installed. Has the same effect as configuring XEP in the *FO Processors Preferences* on page 774 Preferences page.
com.oxygenxml.additional.classpath	A list of JAR-like resources separated by the platform file separator.	An additional list of libraries to be used in the application's internal class loader in addition to the libraries specified in the `lib` folder.
com.oxygenxml.user.home	A file system absolute path pointing to a folder.	Overwrites the user home which was implicitly detected for the application. Used only when the application is running on Windows.
com.oxygenxml.use.late.delegation.for.author.extensions	true/false (true by default)	All Java extensions in a document type configuration are instantiated in a separate class loader. When **true**, the JAR libraries used in a certain document type configuration will have priority to resolve classes before delegating to the parent class loader. This is the default behavior. When **false** the parent class loader will take precedence.
com.oxygenxml.stack.size.validation.threads	The number of bytes used to validation threads.	Some parts of the application (validation, content completion) which use the Relax NG parser require sometimes a larger Thread stack size in order to parse complex schemas. The default value is (5 * 1024 * 1024) which should be more than enough.
com.oxygenxml.jing.skip.validation.xhtml.data.attrs	true/false (true by default)	By default the Relax NG validation was configured to skip validation for XHTML attributes starting with "data-" which should be skipped from validation according to the XHTML 5 specification.
com.oxygenxml.report.problems.url	User defined URL	The URL where a problem reported through the **Report Problem** dialog is sent. The report is sent in XML format using the `report` parameter with the POST HTTP method.

Localizing the User Interface

You can select the language of the Oxygen XML Author user interface. Oxygen XML Author ships with the following languages: English, French, German, Japanese, Dutch, and Italian. To change the interface language, go to

You can also localize in your own language by creating an interface localization file.

Creating an interface localization file

You can change the language of the Oxygen XML Author user interface by creating an interface localization file. The example in this procedure is based on the Spanish language, and a standard Oxygen XML Author Windows distribution.

1. Identify the ISO code for the new language you wish to translate the interface. In this example the language code is **es_ES**.
2. Write an email to the Oxygen XML Author support team and ask them to send you the *translation.xml* sample file.
3. Open *translation.xml* in Oxygen XML Author. The file contains all the interface messages that can be translated and is updated at every new release with the latest additions. Here is a sample of its content:

```
<translation>
    <languageList>
        <language description="English" lang="en_US"/>
    </languageList>
    <key value="New">
        <comment>The File/New action. Creates a new document.</comment>
        <val lang="en_US">New</val>
    </key>
    <key value="New_folder">
        <comment>Creates a folder in the Project View.</comment>
        <val lang="en_US">New Folder</val>
    </key>
    .....
</translation>
```

4. Update the `language` element to reflect the new language. Set the `description` attribute to `Spanish` and the `lang` attribute to `es_ES`.
5. For each `key` element translate the `comment` (optional) and `val` elements. Set the `lang` attribute to `es_ES`.

   📋 **Note:** After you are finished, the file should look like:

   ```
   <translation>
       <languageList>
           <language description="Español" lang="es_ES"/>
       </languageList>
       <key value="New">
         <comment>El Archivo / Nueva acción. Crea un nuevo documento.</comment>

           <val lang="es_ES">Nuevo</val>
       </key>
       <key value="New_folder">
           <comment>Crea una carpeta en la vista del proyecto.</comment>
           <val lang="es_ES">Nueva carpeta</val>
       </key>
       .....
   </translation>
   ```

6. *Open the Preferences dialog* and go to **Global** and enable the **Other language** option. Browse for the *translation.xml* file.
7. Restart the application.

Setting a Java Virtual Machine Parameter in the Launcher Configuration File / Start-up Script

There are two ways you can set new Java Virtual Machine parameters:

- *Setting parameters for the Oxygen XML Author launchers*
- *Setting parameters in the command line scripts*

Setting Parameters for the Application Launchers

Increasing the amount of memory that Oxygen XML Author uses on Windows

To increase the memory available to Oxygen XML Author on Windows:

- navigate to the installation directory of Oxygen XML Author;

- locate the -Xmx parameter in the `oxygenAuthor16.1.vmoptions` file;

  📄 **Note:** For 32-bit Windows modify the parameter to -Xmx1024m or larger, but not over -Xmx1200m. Make sure you do not exceed your physical RAM. For 64-bit Windows modify the parameter to a larger value (for example -Xmx2048m). We recommended you to not use more than half of your existing physical RAM.

Restart Oxygen XML Author. Go to **Help** > **About** and verify the amount of memory that is actually available (see the last row in the **About** dialog). In case Oxygen XML Author does not start and you receive and error message saying that it could not start the JVM, decrees the -Xmx parameter and try again.

For Windows Vista/7, copy the `oxygenAuthor16.1.vmoptions` to your desktop (or to any other folder with write access), modify it there, then copy it back to the Oxygen XML Author installation folder.

📄 **Note:** The parameters from the `.vmoptions` file are used when you start Oxygen XML Author with the *oxygen* launcher (or with the desktop shortcut). In case you use the command line script `oxygen.bat/oxygen.sh`, modify the -Xmx parameter in the script file.

Increasing the amount of memory that Oxygen XML Author uses on Mac OS X

To increase the memory available to Oxygen XML Author on Mac OS X:

- <u>**Ctrl+Click (Command+Click on OS X)**</u> (or right click) the Oxygen XML Author icon in **Finder**;
- from the pop-up menu, select **Show Package Contents**;
- navigate to the `Contents` directory and open for editing the `Info.plist` file;

  📄 **Note:** You can open this file either with the **Property List Editor**, or the **TextEdit**.

- Look for the **VMOptions** key and adjust the -Xmx parameter to a larger value (for example -Xmx1500m)

  📄 **Note:** Try not to use more than half of your existing physical RAM if possible.

Setting a system property

To set a system property, you have to provide a parameter of the following form:

```
-Dproperty.name=value
```

You can also set a system property through a parameter prefixed with `-Doxy` in the command line used to start the application:

```
oxygenAuthor16.1.exe "-Doxyproperty.name=value"
```

All system properties are displayed in the **System properties** tab of the **About** dialog.

To view the list of all system properties, go to *Custom System Properties* on page 807.

Setting Parameters in the Command Line Scripts

If you start Oxygen XML Author with the `oxygenAuthor.bat` script, you have to add or modify the parameter to the java command at the end of the script.

For example, to set the maximum amount of Java memory to 600 MB on **Windows**, modify the **-Xmx** parameter like this:

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

on **Mac OS X** the java command should look like:

```
java "-Xdock:name=Author"\
 -Dcom.oxygenxml.editor.plugins.dir="$AUTHOR_HOME/plugins"\
```

```
 -Xmx600m\
 ...
```

and on **Linux** the Java command should look like:

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

Chapter

20

Performance Problems

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

Large Documents

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

⚠️ **Attention:**

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

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

External Processes

The *Memory available to the built-in FOP* option controls the amount of memory allocated to generate PDF output with the built-in Apache FOP processor. If Oxygen XML Author throws an *Out Of Memory* error, *open the **Preferences dialog*** , go to **XML** > **XSLT-FO-XQuery** > **FO Processors**, and increase the value of the *Memory available to the built-in FOP* option.

For external XSL-FO processors, XSLT processors, and external tools, the maximum value of the allocated memory is set in the command line of the tool using the -Xmx parameter set to the Java virtual machine.

Display Problems on Linux or Solaris

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

Chapter

21

Common Problems

This chapter presents common problems that may appear when running the application and the solutions for these problems.

- *Damaged File Associations on OS X*
- *I Cannot Connect to SVN Repository From Repositories View*
- *Problem Report Submitted on the Technical Support Form*
- *Signature verification failed error on open or edit a resource from Documentum*
- *Cannot Cancel a System Shutdown*
- *Compatibility Issue Between Java and Certain Graphics Card Drivers*
- *An Image Appears Stretched Out in the PDF Output*
- *The DITA PDF Transformation Fails*
- *Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x*
- *JPEG CMYK Color Space Issues*
- *SVG Rendering Issues*
- *MSXML 4.0 Transformation Issues*

XML Document Takes a Long Time to Open

If Oxygen XML Author takes a long time to open a document, check the following:

It takes longer to open an XML document if the whole content of your document is on a single line or if the document size is very large.

If the content is on a single line, you can speed up loading by enabling the **Format and indent the document on open** option. To do so, *open the Preferences dialog* and go to *Editor > Format > Format and indent the document on open*.

If the document is very large (above 30 MB) make sure that the value of *Optimize loading in the Text edit mode for files over*) is greater than the size of your document.

If that fails and you get an Out Of Memory error (**OutOfMemoryError**) you can *increase the memory available to Oxygen XML Author.*

Oxygen XML Author Takes Several Minutes to Start on Mac

If Oxygen XML Author takes several minutes to start, the Java framework installed on the Mac may have a problem. One solution for this is to update Java to the latest version: go to **Apple symbol** > **Software Update**. After it finishes to check for updates, click **Show Details**, select the Java Update (if one is available) and click **Install**. If no Java updates are available, reset the Java preferences to their defaults. Start **Applications > Utilities > Java Preferences** and click **Restore Defaults**.

Out Of Memory Error When I Open Large Documents

I am trying to open a file larger than 100 MB to edit it in Oxygen XML Author, but it keeps telling me it runs out of memory (**OutOfMemoryError**). What can I do?

You should make sure that the minimum limit of document size that enables the support for editing large documents (the value of *the option Optimize loading in the Text edit mode for files over*) is less than the size of your document. That will enable the optimized support for large documents. If that fails and you still get an Out Of Memory error you should *increase the memory available to Oxygen XML Author.*

Other tips:

• Make sure that you close other files before opening the large file.
• You can set the default editing mode *in the Preferences dialog*. The *Text editing mode* uses less memory than other editing modes.
• If the file is too large for the editor to handle, you can *open it in for viewing in Large File Viewer*.

Special Characters Are Replaced With a Square in Editor

My file was created with other application and it contains special characters like é, ©, ®, etc. Why does Oxygen XML Author display a square for these characters when I open the file in Oxygen XML Author?

You must set a font able to render the special characters in the *Font preferences*. If it is a text file you must set also *the encoding used for non XML files.* If you want to set a font which is installed on your computer but that font is not accessible in the **Font** preferences that means the Java virtual machine is not able to load the system fonts, probably because it is not a True Type font. It is a problem of the Java virtual machine and a possible solution is to copy the font file in the [JVM_DIR]/lib/fonts folder. [JVM_DIR] is the value of the property *java.home* which is available in the **System properties** tab of the **About** dialog that is opened from menu **Help** > **About**.

The Scroll Function of my Notebook's Trackpad is Not Working

I got a new notebook (Lenovo Thinkpad™ with Windows) and noticed that the scroll function of my trackpad is not working in Oxygen XML Author.

It is a problem of the Synaptics™ trackpads which can be fixed by adding the following lines to the `C:\Program Files\Synaptics\SynTP\TP4table.dat` file:

```
*,*,oxygen16.1.exe,*,*,*,WheelStd,1,9
*,*,oxygenAuthor16.1.exe,*,*,*,WheelStd,1,9
*,*,oxygenDeveloper16.1.exe,*,*,*,WheelStd,1,9
*,*,syncroSVNClient.exe,*,*,*,WheelStd,1,9
*,*,diffDirs.exe,*,*,*,WheelStd,1,9
*,*,diffFiles.exe,*,*,*,WheelStd,1,9
```

`NullPointerException` at Startup on Windows XP

When I start Oxygen XML Author on Windows XP I get the following error. What can I do?

```
Cannot start Oxygen XML Author.
Due to:java.lang.NullPointerException
java.lang.NullPointerException
at com.sun.java.swing.plaf.windows.XPStyle.getString(Unknown Source)
at com.sun.java.swing.plaf.windows.XPStyle.getString(Unknown Source)
at com.sun.java.swing.plaf.windows.XPStyle.getDimension(Unknown Source)
at com.sun.java.swing.plaf.windows.WindowsProgressBarUI.
getPreferredInnerHorizontal(Unknown Source)
```

The error is caused by a bug in the Java runtime from Sun Microsystems. You can avoid it by setting the Java system property *com.oxygenxml.no.xp.theme* to the value *true* in the startup script, that is *adding the startup parameter -Dcom.oxygenxml.no.xp.theme=true*. If you start Oxygen XML Author with the author.bat script just add the parameter-Dcom.oxygenxml.no.xp.theme=true to the java command in the script. If you start Oxygen XML Author from the Start menu shortcut you have to add the same parameter on a new line in the file [oXygen-install-folder]\oxygenAuthor16.1.vmoptions .

Crash at Startup on Windows with an Error Message About a File `nvoglv32.dll`

I try to start Oxygen XML Author on Windows but it crashed with an error message about "Fault Module Name: nvoglv32.dll". What is the problem?

It is an OpenGL driver issue that can be avoided by creating an empty file called `opengl32.dll` in the Oxygen XML Author install folder (if you start Oxygen XML Author with the shortcut created by the installer on the Start menu or on Desktop) or in the subfolder `bin` of the home folder of the Java virtual machine that runs Oxygen XML Author (if you start Oxygen XML Author with the `oxygen.bat` script). The home folder of the Java virtual machine that runs Oxygen XML Author is the value of the property `java.home` which is available in the **System properties** tab of the **About** dialog that is opened from menu **Help** > **About**.

Oxygen XML Author Crashed on My Mac OS X Computer

Oxygen XML Author crashed the Apple Java virtual machine/Oxygen XML Author could not start up on my OS X computer due to a JVM crash. What can I do?

Usually it is an incompatibility between the Apple JVM and a native library of the host system. More details are available in the crash log file generated by OS X and reported in the crash error message.

Wrong Highlights of Matched Words in a Search in User Manual

When I do a keyword search in the User Manual that comes with the Oxygen XML Author application the search highlights the wrong word in the text. Sometimes the highlighted word is several words after the matched word. What can I do to get correct highlights?

This does not happen when Oxygen XML Author runs with a built-in Java virtual machine, that is a JVM that was installed by Oxygen XML Author in a subfolder of the installation folder, for example on Windows and Linux when installing Oxygen XML Author with the installation wizard specific for that platform. When Oxygen XML Author runs from an All Platforms installation it uses whatever JVM was found on the host system which may be incompatible with the JavaHelp indexer used for creating the built-in User Manual. Such a JVM may offset the highlight of the matched word with several characters, usually to the right of the matched word. In order to see the highlight exactly on the matched word it is recommended to install the application with the specific installation wizard for your platform (available only for Windows and Linux).

Keyboard Shortcuts Do Not Work

The keyboard shortcuts listed in the *Menu Shortcut Keys preferences* do not work. What can I do?

Usually this happens when a special keyboard layout is set in the operating system which generates other characters than the usual ones for the keys of a standard keyboard. For example if you set the extended Greek layout for your keyboard you should return to the default Greek layout or to the English one. Otherwise the Java virtual machine that runs the application will receive other key codes than the usual ones for a standard keyboard.

After Installing Oxygen XML Author I Cannot Open XML Files in Internet Explorer Anymore

Before installing Oxygen XML Author I had no problems opening XML files in Internet Explorer. Now when I try to open an XML file in Internet Explorer it opens the file in Oxygen XML Author. How can I load XML files in Internet Explorer again?

XML files are opened in Oxygen XML Author because Internet Explorer uses the Windows system file associations for opening files and you associated XML files with Oxygen XML Author in the installer panel called **File Associations** therefore Internet Explorer opens XML files with the associated Windows application.

By default the association with XML files is disabled in the Oxygen XML Author installer panel called **File Associations**. When you enabled it the installer displayed a warning message about the effect that you experience right now.

For opening XML files in Internet Explorer again you have to set Internet Explorer as the default system application for XML files, for example by right-clicking on an XML file in Windows Explorer, selecting **Open With** > **Choose Program**, selecting IE in the list of applications and selecting the checkbox **Always use the selected program**. Also you have to run the following command from a command line:

```
wscript revert.vbs
```

where `revert.vbs` is a text file with the following content:

```
function revert()
  Set objShell = CreateObject("WScript.Shell")
  objShell.RegWrite "HKCR\.xml\", "xmlfile", "REG_SZ"
  objShell.RegWrite "HKCR\.xml\Content Type", "text/xml", "REG_SZ"
end function

revert()
```

I Cannot Associate Oxygen XML Author With a File Type on My Windows Computer

I cannot associate the Oxygen XML Author application with a file type on my Windows computer by right clicking on a file in Windows Explorer, selecting **Open With** > **Choose Program** and browsing to the file oxygenAuthor16.1.exe. When I select the file oxygenAuthor16.1.exe in the Windows file browser dialog the Oxygen XML Author application is not added to the list of applications in the **Open With** dialog. What can I do?

The problem is due to some garbage Windows registry entries remained from versions of Oxygen XML Author older than version 9.0. Please uninstall all your installed versions of Oxygen XML Author and run a registry cleaner application for cleaning these older entries. After that just reinstall your current version of Oxygen XML Author and try again to create the file association.

The Files Are Opened in Split Panels When I Restart Oxygen XML Author

When I close the Oxygen XML Author application with multiple files open and then restart it, every file opens in a split panel of the editing area instead of a tab sharing with the other opened files the same editing area which organizes the editors in a tabbed pane. I want to have the files arranged as a tabbed pane as they used to be arranged before closing the Oxygen XML Author application.

This happens sometimes when several files are opened automatically on startup. It is a problem of the JIDE docking views library used in Oxygen XML Author for docking and floating views. The workaround is to run the action **Window** > **Reset Layout**. To preserve specific layout of the Oxygen XML Author when running this action, *open the **Preferences** dialog* , go to **Perspectives Layout**, and set the option **Use fixed layout**.

Grey Window on Linux With the Compiz / Beryl Window Manager

I try to run Oxygen XML Author on Linux with the Compiz / Beryl window manager but I get only a grey window which does not respond to user actions. Sometimes the Oxygen XML Author window responds to user actions but after opening and closing an Oxygen XML Author dialog or after resizing the Oxygen XML Author window or a view of the Oxygen XML Author window the content of this window becomes grey and it does not respond to user actions. What is wrong?

Sun Microsystems' Java virtual machine *does not support the Compiz window manager and the Beryl one very well*. It is expected that better support for Compiz / Beryl will be added in future versions of their Java virtual machine. You should turn off the special effects of the Compiz / Beryl window manager before starting the Oxygen XML Author application or switch to other window manager.

Drag and Drop Without Initial Selection Does Not Work

When I try to drag with the mouse an unselected file from the **Project** view or the **DITA Maps Manager** view, the drag never starts, it only selects the resource. I need to drag the resource again after it becomes selected. As a result any drag and drop without initial selection becomes a two step operation. How can I fix this?

This is *a bug* present in JVM versions prior to 1.5.0_09. This issue is fixed in 1.5.0_09 and newer versions. See *the installation instructions* for setting a specific JVM version for running the Oxygen XML Author application.

Set Specific JVM Version on Mac OS X

How do I configure Oxygen XML Author to run with the version X of the Apple Java virtual machine on my Mac OS X computer?

Oxygen XML Author uses the first JVM from the list of preferred JVM versions set on your Mac computer that has a version number 1.6.0 or higher. You can move your desired JVM version up in the preferred list by dragging it with the mouse on a higher position in the list of JVMs available in the **Java Preferences** panel that is opened from **Applications** > **Utilities** > **Java** > **Java Preferences**.

Segmentation Fault Error on Mac OS X

On my Mac OS X machine the application gives a *Segmentation fault* error when I double-click on the application icon. Sometimes it gives no error but it does not start. What is the problem?

Please make sure you have the latest Java update from the Apple website installed on your Mac OS X computer. If installing the latest Java update doesn't solve the problem please copy the file JavaApplicationStub from the /System/Frameworks/JavaVM.framework folder to the OxygenAuthor.app/Contents/MacOS folder. For browsing the .app folder you have to **(CMD+click)** on the Oxygen XML Author icon and select **Show Package Contents**.

Damaged File Associations on OS X

After upgrading OS X to version 10.4.x / Oxygen XML Author to version 6.x Oxygen XML Author is not associated anymore to the file types XML, XSL, XSD, etc. This worked in the previous version of Oxygen XML Author. How can I create the file associations again?

The upgrade damaged the file associations in the LaunchService Database on your OS X machine. Please rebuild the LaunchService Database with the following procedure. This will reset all file associations and will rescan the entire file system searching for applications that declare file associations and collecting them in a database used by Finder.

1. Find all the Oxygen XML Author installations on your hard drive.
2. Delete them by dragging them to the Trash.
3. Clear the Trash.
4. Unpack the Oxygen XML Author installation kit on your desktop.
5. Copy the contents of the archive into the folder / Applications / Oxygen.
6. Run the following command in a Terminal:

```
/System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/LaunchServices.framework/Versions/A/Support/lsregister
 -kill -r -domain local -domain system -domain user
```

7. Restart Finder with the following command:

```
killall Finder
```

8. Create a XML or XSD file on your desktop.
   It should take the Oxygen XML Author icon.
9. Double click the file.
10. Accepting the confirmation dialog.

Oxygen XML Author will start up and the file associations work correctly.

I Cannot Connect to SVN Repository From Repositories View

I cannot connect to a SVN repository from the **Repositories** view of SVN Client. How can I find more details about the error?

First check that you entered the correct URL of the repository in the **Repositories** view. Also check that a SVN server is running on the server machine specified in the repository URL and is accepting connections from SVN clients. You can check that the SVN server accepts connections with the command line SVN client from CollabNet.

If you try to access the repository with a `svn+ssh` URL also check that a SSH server is running on port 22 on the server machine specified in the URL.

If the above conditions are verified and you cannot connect to the SVN repository please generate a logging file on your computer and send the logging file to support@oxygenxml.com. For generating a logging file you need to create a text file called `log4j.properties` in the install folder with the following content:

```
log4j.rootCategory= debug, R2

log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=logging.log
log4j.appender.R2.MaxFileSize=12000KB
log4j.appender.R2.MaxBackupIndex=20
log4j.appender.R2.layout=org.apache.log4j.PatternLayout
log4j.appender.R2.layout.ConversionPattern=%r %p [ %t ] %c - %m%n
```

Restart the application, reproduce the error, close the application and send the file `logging.log` generated in the install directory to support@oxygenxml.com.

Problem Report Submitted on the Technical Support Form

What details should I add to my problem report that I enter on the Technical Support online form of the product website?

For problems like server connection error, unexpected delay while editing a document, a crash of the application, etc for which the usual details requested on the Technical Support online form are not enough you should generate a log file and attach it to the problem report. In case of a crash you should also attach the crash report file generated by your operating system. For generating a logging file you need to create a text file called `log4j.properties` in the install folder with the following content:

```
log4j.rootCategory= debug, R2

log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=logging.log
log4j.appender.R2.MaxFileSize=12000KB
log4j.appender.R2.MaxBackupIndex=20
log4j.appender.R2.layout=org.apache.log4j.PatternLayout
log4j.appender.R2.layout.ConversionPattern=%r %p [ %t ] %c - %m%n
```

Restart the application, reproduce the error and close the application. The log file is called `logging.log` and is located in the install folder.

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.

Set the `-Drequire.signed.ucf.jars=false` parameter, as explained in the *Setting a Parameter in the Launcher Configuration File / Startup Script* topic.

```
-vmargs
-Xms40m
-Xmx256m
-Drequire.signed.ucf.jars=false
```

Cannot Cancel a System Shutdown

If I try to shutdown my Win XP when there is at least one modified document open in Oxygen XML Author, I am prompted to cancel the shutdown or force quit the application. If I choose **Cancel**, the system shuts down anyway. Why is that?

This problem was reported on Windows XP systems only. The known workaround is to start Oxygen XML Author using the `author.bat` script.

Compatibility Issue Between Java and Certain Graphics Card Drivers

Under certain settings, a compatibility issue can appear between Java and some graphics card drivers, which results in the text from the editor (in **Author** or **Text** mode) being displayed garbled. In case you encounter this problem, update your graphics card driver. Another possible workaround is, *open the **Preferences** dialog* and go to **Fonts** > **Text antialiasing** and set the value of **Text antialiasing** option to ON.

> 📝 **Note:** If this workaround does not resolve the problem, set the **Text antialiasing** option to other values than ON.

An Image Appears Stretched Out in the PDF Output

Sometimes, when publishing XML content (DITA, DocBook, etc), images are scaled up in the PDF outputs but are displayed perfectly in the HTML (or WebHelp) output.

PDF output from XML content is obtained by first obtaining a intermediary XML format called XSL-FO and then applying an XSL-FO processor to it to obtain the PDF. This stretching problem is caused by the fact that all XSL-FO processors take into account the DPI (dots-per-inch) resolution when computing the size of the rendered image.

The PDF processor which comes out of the box with the application is the open-source Apache FOP processor. Here is what Apache FOP does when deciding the image size:

1. If the XSL-FO output contains width, height or a scale specified for the image `external-graphic` tag, then these dimensions are used. This means that if in the XML (DITA, DocBook, etc) you set explicit dimensions to the image they will be used as such in the PDF output.
2. If there are no sizes (width, height or scale) specified on the image XML element, the processor looks at the image resolution information available in the image content. If the image has such a resolution saved in it, the resolution will be used and combined with the image width and height in order to obtain the rendered image dimensions.
3. If the image does not contain resolution information inside, Apache FOP will look at the FOP configuration file for a default resolution. The FOP configuration file for XSLT transformations which output PDF is located in the `[OXYGEN_DIR]/lib/fop.xconf`. DITA publishing uses the DITA Open Toolkit which has the Apache FOP configuration file located in `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/org.dita.pdf2/fop/conf/fop.xconf`. The configuration file contains two XML elements called `source-resolution` and `target-resolution`. The values set to those elements can be increased, usually a DPI value of 110 or 120 should render the image in PDF just like in the HTML output.

The commercial **RenderX XEP** XSL-FO processor behaves similarly but as a fallback it uses 120 as the DPI value instead of using a configuration file.

> ℹ️ **Tip:**
>
> As a conclusion, it is best to save your images without any DPI resolution information in them. For example the open-source GIMP image editor allows you when saving a PNG image whether to save the resolution to it or not:
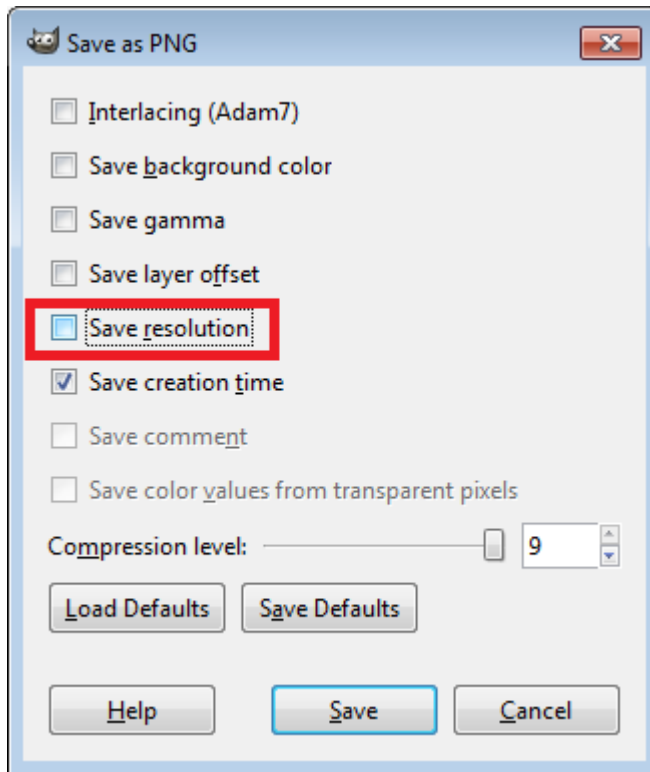
Having images without any resolution information saved in them allows you to control the image resolution from the configuration file for all referenced images.

The DITA PDF Transformation Fails

To generate the PDF output, Oxygen XML Author uses the DITA Open Toolkit.

If your transformation fails you can detect some of the problems that caused the errors by running *the Validate and Check for Completeness action*. Depending on the options you select when you run it, this action reports errors such as topics referenced in other topics but not in the DITA Map, broken links, and missing external resources.

You can analyse the **results** tab of the DITA transformation and search for messages that contain text similar to `[fop]` `[ERROR]`. If you encounter this type of error message, edit the transformation scenario you are using and set the **clean.temp** parameter to **no** and the **retain.topic.fo** parameter to **yes**. Run the transformation, go to the temporary directory of the transformation, open the `topic.fo` file and go to the line indicated by the error. Depending on the XSL FO context try to find the DITA topic that contains the text which generates the error.

If none of the above methods helps you, go to **Help** > **About** > **Components** > **Frameworks** and check what version of the DITA Open Toolkit you are using. Copy the whole output from the DITA OT console output and either report the problem on the DITA User List or to support@oxygenxml.com.

Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x

On some Linux systems based on Gnome 3.x (e.g. Ubuntu 11.x, 12.x) the main menu of Oxygen XML Author has alignment issues when you navigate it using your mouse.

This is a known problem caused by Java SE 6 1.6.0_32 and earlier. You can resolve this problem using the latest Java SE 6 JRE from Oracle. To download the latest version, go to *http://www.oracle.com/technetwork/java/javase/downloads/index.html*.

To bypass the JRE bundled with Oxygen XML Author, go to the installation directory of Oxygen XML Author and rename or move the `jre` folder. If Oxygen XML Author does not seem to locate the system JRE, either set the JAVA_HOME environment variable to point to the location where you have installed the JRE, or you can simply copy that folder with the JRE to the installation directory and rename it to `jre` to take the place of the bundled JRE.

JPEG CMYK Color Space Issues

JPEG images with CMYK color profile having the color profiles embedded in the image should be properly rendered in the **Author** mode.

If the color profile information is missing from the JPEG image but you have the ICC file available, you can copy the `profileFileName.icc` to the `[OXYGEN_DIR]\lib` directory.

If the color space profile is missing, JPEG images that have the CMYK color space are rendered without taking the color profile into account. The **Unsupported Image Type** message is displayed above the image.

SVG Rendering Issues

Oxygen XML Author uses the **Apache Batik** open source library to render SVG images. The **Batik** library only has partial support for SVG 1.2: *http://xmlgraphics.apache.org/batik/dev/svg12.html*.

For example, if you are using the *Inkscape* SVG editor, it is possible that it saves the SVG as 1.1 but it actually uses SVG 1.2 elements like `flowRoot` inside it. This means that the image will not be properly rendered inside the application.

SVG images shown in the **Author** visual editing mode are rendered as static images, without support for animations and Javascript.

MSXML 4.0 Transformation Issues

In case the latest MSXML 4.0 service pack is not installed on your computer, you are likely to encounter the following error message in the **Results** panel when you run a transformation scenario that uses the MSXML 4.0 transformer.

Error Message

```
Could not create the 'MSXML2.DOMDocument.4.0' object.
Make sure that MSXML version 4.0 is correctly installed on the machine.
```

To fix this issue, go to the Microsoft website and get the latest MSXML 4.0 service pack.

Glossary

Java Archive

JAR (Java ARchive) is an archive file format. JAR files are built on the ZIP file format and have the .jar file extension. Computer users can create or extract JAR files using the `jar` command that comes with a JDK.

Java Archive (JAR)

JAR

Apache Ant

Apache Ant (Another Neat Tool) is a software tool for automating software build processes.

Ant

Active cell

The selected cell in which data is entered when you begin typing. Only one cell is active at a time. The active cell is bounded by a heavy border.

Block element

A block element is an one that is intended to be visually separated from its siblings, usually vertically. For instance, a paragraph or a list item are block elements. It is distinct from a *inline element* which has no such separation.

Inline element

An inline element is one that is intended to be displayed in the same line of text as its siblings or the surrounding text. For instance, strong and emphasis in HTML are inline elements. It is distinct from a *block element*, which is visually separated from its siblings.

DITA map

A DITA map is a hierarchical collection of DITA topics that can be processed to form an output. Maps do not contain the content of topics, but only references to them. These are known as topic references. Usually the maps are saved on disk or in a CMS with the extension '.ditamap'.

Maps can also contain relationship tables that establish relationships between the topics contained within the map. Relationship tables are also used to generate links in your published document.

You can use your map or bookmap to generate a deliverable using an output type such as XHTML, PDF, HTML Help or Eclipse Help.

Bookmap

A bookmap is a specialized *ditamap* used for creating books. A bookmap supports book divisions such as chapters and book lists such as indexes.

Index