Oxygen XML Editor Eclipse Plugin 17.1

Notice

Copyright

Oxygen XML Editor plugin User Manual

Syncro Soft SRL.

Contents

Chapter 1

Introduction

Welcome to the User Manual of Oxygen XML Editor plugin 17.1.

Oxygen XML Editor plugin is a cross-platform application designed to accommodate all of your XML editing, authoring, developing, and publishing needs. It is the best XML editor available for document development using structured mark-up languages such as XML, XSD, Relax NG, XSL, DTD. It is a comprehensive solution for authors who want to edit XML documents visually, with or 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 many innovative, user-friendly authoring features that make XML authoring easy and powerful.

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

This user guide is focused on describing features, functionality, the application interface, and to help you quickly get started. It also includes a vast amount of advanced technical information and instructional topics that are designed to teach you how to use Oxygen XML Editor plugin to accomplish your tasks. It is assumed that you are familiar with the use of your operating system and the concepts related to XML technologies and structured mark-up.

Chapter 2

Getting Started

Topics:

This chapter is designed to help you get started using Oxygen XML Editor plugin as quickly as possible and to provide you with a variety of resources to help you get the most out of the application. Typically, the first step of getting started with Oxygen XML Editor plugin would be to install the software. For detailed information about that process, see the *Installation* on page 33 chapter.

After installation, when you launch Oxygen XML Editor plugin for the first time, you are greeted with a **Welcome** dialog box. It presents upcoming events, useful video demonstrations, helpful resources, the tip of the day, and also gives you easy access to recently used files and projects and to create new ones.

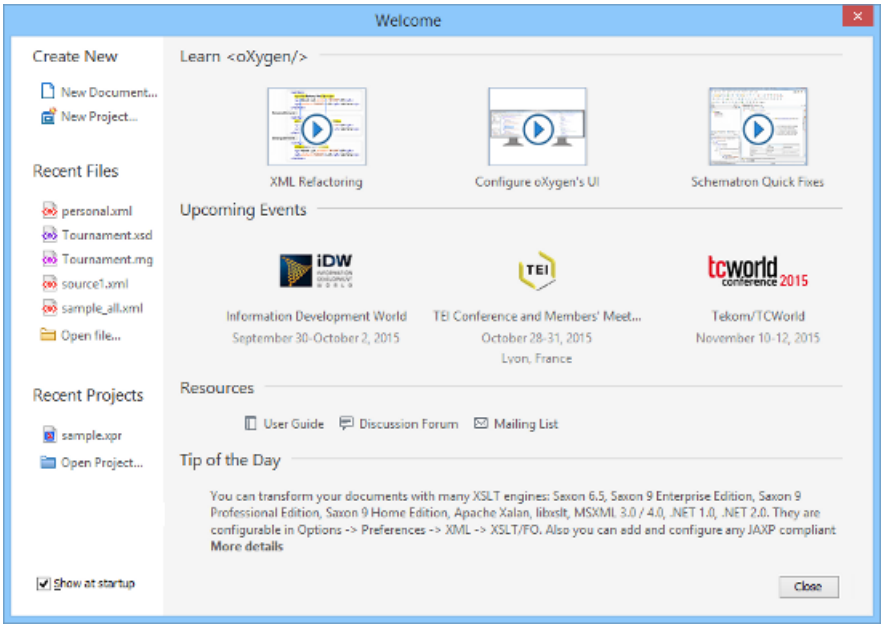


Figure 1: The Welcome Dialog Box

If you do not want it to be displayed every time you launch Oxygen XML Editor plugin, disable the **Show at startup** option. To display it any time, go to **Help** > **Welcome**.

What is Oxygen XML Editor plugin

Oxygen XML Editor plugin is the best XML editor available and is a complete XML development and authoring solution. It is designed to accommodate a large number of users, ranging from beginners to XML experts. It is the only XML tool that supports all of the XML schema languages and provides a large variety of powerful tools for editing and publishing XML documents.

You can use Oxygen XML Editor plugin to work with most XML-based standards and technologies. It is a cross-platform application available on all the major operating systems (Windows, Mac OS X, Linux, Solaris) and can be used either as a standalone application or as an Eclipse plugin.

For a list of many of the features and technologies that are included in Oxygen XML Editor plugin, see the *oXygen Website*.

Getting Familiar with the Layout

Oxygen XML Editor plugin includes several *perspectives* and *editing modes* to help you accomplish a wide range of tasks. Each perspective and editing mode also includes a large variety of helper view, menu actions, toolbars, and contextual menu functions.

Regardless of the *perspective* or *editing mode* that you are working with, the default layout is comprised of the following areas:

Menus

Menu driven access to all the features and functions available in Oxygen XML Editor plugin. Most of the menus are common for all types of documents, but Oxygen XML Editor plugin also includes some context-sensitive and framework-specific menus and actions that are only available for a specific context or type of document.

Toolbars

Easy access to common and frequently used functions. Each icon is a button that acts as a shortcut to a related function. Some of the toolbars are common for all perspectives, editing modes, and types of documents, while others are specific to the particular perspective or mode. Some toolbars are also framework-specific, depending on the type of document that is being edited.

Helper Views

Oxygen XML Editor plugin includes a large variety of views to assist you with editing, viewing, searching, validating, transforming, and organizing your documents. Many of the views also contain useful contextual menu actions, toolbar buttons, or menus. The most commonly used views for each perspective and editing mode are displayed by default and you can choose to display others to suite your specific needs.

Editor Pane

The main editing area in the center of the application. Each *editing mode* provides a main editor pane where you spend most of your time reading, editing, applying markup, and validating your documents. The editor pane in each *editing mode* also includes a variety of contextual menu actions and other features to help streamline your editing tasks.

Perspectives

Oxygen XML Editor plugin includes *several different perspectives* that you can use to work with your documents. The **<oXygen/> XML** perspective is the most commonly used perspective used for displaying and editing the content of your XML documents, and it is the default perspective when you start Oxygen XML Editor plugin for the first time. Oxygen XML Editor plugin also includes a **Database** perspective that allows you to manage databases and their connections and a few debugging perspectives that allow you to detect problems in XSLT or XQuery transformations.

Resources to Help You Get Started Using Oxygen XML Editor plugin

Configuring Oxygen XML Editor plugin

There are numerous ways that you can configure Oxygen XML Editor plugin to accommodate your specific needs.

- See the *Configuring Oxygen XML Editor plugin* on page 907 section for details on the various ways that you can configure the application and its features.

Video Tutorials

The Oxygen XML Editor plugin website includes numerous video demonstrations and webinars that present many of the features that are available in Oxygen XML Editor plugin and show you how to complete specific tasks or how to use the various features.

- Go to the *Oxygen XML Editor plugin Videos Page* to see the list of video tutorials and webinars.

Oxygen XML Editor plugin Documentation

The Oxygen XML Editor plugin documentation includes a plethora of sections and topics to provide you with a variety of information, ranging from basic authoring tasks to advanced developer techniques. You can, of course, search through the documentation using standard search mechanisms, but you can also place the cursor in any particular position in the interface and use the **F1** key to open a dialog box that presents a section in the documentation that is appropriate for the context of the current cursor position. Aside from the other topics in this *Getting Started* section, the following are links to other sections of the documentation that might be helpful for your specific needs:

- ***Text Editing Mode*** **on page 58 Section** - Provides information about the **Text** editor.
- ***Author Editing Mode*** **on page 75 Section** - Provides information about the visual WYSIWYG-like **Author** editing mode.
- ***Editing Documents*** **on page 107 Section** - Includes information about editing a large variety of different types of documents.
- ***DITA Authoring*** **on page 403 Section** - Provides information about using DITA to edit and structure your content.
- ***WebHelp System Output*** **on page 758 Section** - Provides information about the WebHelp system that can be used for publishing content.
- ***Importing Data*** **on page 877 Section** - Provides information about importing data from text files, MS Excel files, database data, and HTML files.

Sample Documents

Your installation of Oxygen XML Editor plugin includes a large variety of sample documents and projects that you can use as templates to get started and to experiment with the various features and technologies. They are located in the **samples** folder that is located in the installation directory of Oxygen XML Editor plugin. You will find files and folders for a variety of different types of documents, including the following:

- **`sample.xpr` file** - A sample project file that will allow you to experiment with how projects can be structured and used. When you open this project file, you will be able to see all the sample files and folders in the **Navigator** view.
- **`personal` files** - A collection of interrelated sample files that will allow you to experiment with the structure and relationship between XML files, stylesheets, and schemas.
- **Various document type folders** - The various folders contain sample files for numerous different types, such as CSS, DITA, DocBook, ePub, TEI, xhtml, and many others.

Other Resources

The following list includes links to various other resources that will help you get started using the features of Oxygen XML Editor plugin:

- See the *Oxygen XML Editor plugin Blog Site* for a large variety of current and archived blogs in regards to numerous features, requests, and instructional topics.
- Take advantage of the *Oxygen XML Editor plugin Forum* to see various announcements and learn more about specific issues that other users have experienced.

- If you are using DITA, see the incredibly helpful *DITA Style Guide Best Practices for Authors*.
- To learn about the WebHelp features in Oxygen XML Editor plugin, see the *Publishing DITA and DocBook to WebHelp* section of the website.
- For more information about various additional tools that are integrated into Oxygen XML Editor plugin, see the *Tools section*.
- See the *External Resource Page* for links to various other helpful resources, such as discussion lists, external tutorials, and more.
- See the *oXygen SDK* section for details about the SDK that allows you to extend and develop Oxygen XML Editor plugin frameworks and plugins, and to integrate Eclipse plugins.
- For a list of new features that were implemented in the latest version of Oxygen XML Editor plugin, see the *What's New Section of the Website*
- You can select the **Tip of the Day** action in the **Help** menu to display a dialog box that includes a variety of tips for using Oxygen XML Editor plugin.

Your First Document or Project

This section includes several topics that will help you get started with your first document or project.

Your First XML Document

To create your first XML document, select **File** > **New** > **Other** > **oXygen** or click the  **New** button on the toolbar. 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 wizard. You can also create one from a template by selecting **File** > **New** > **New from Templates** and choosing a template from the **Global templates** or **Framework templates** folders. If you are looking for a common document type, such as DITA or DocBook, you can find templates for these document types in the **Framework templates** folder. If your company has created its own templates, you can also find them there.

For some document types, you may find a number of different templates. For example, there are numerous templates for DocBook documents, and 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 Editor plugin interface changes to support editing that document type. This may include new menus, toolbar buttons, and items in the contextual menus.

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

Whether there is an **Author** mode view available for your document type depends on the type you choose and if there is a CSS stylesheet available to create the **Author** view. Oxygen XML Editor plugin includes default **Author** mode views for most of the document types it supports. If your company has created its own document types, **Author** mode stylesheets may have also been created for that type. However, if you create a plain XML file, or one based on a schema that is not included in the Oxygen XML Editor plugin built-in support, you need to edit it 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 do not lose any formatting when switching from **Author** to **Text** mode. **Text** and **Author** modes are just different views for 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 such as list items and table rows*.

If you use **Author** mode, you might find that it is similar to word processors that you are used to. Likewise, the **Text** mode is similar to many typical text editors. If you are new to XML, the biggest difference is that XML documents have a particular structure that you have to follow. Oxygen XML Editor plugin assists you with a continuous validation of the XML markup.

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, while others give you very few. In either case, you need to make sure that your document follows the particular structure for the document type you are creating. This means:

- At any given location in the document, there are only certain XML elements allowed. Oxygen XML Editor plugin helps you determine which elements are allowed. In **Author** mode, when you press **Enter**, Oxygen XML Editor plugin assumes that you want to enter a new element and shows you a list of elements that can be created in this location. Keep typing until the element you want is highlighted and press **Enter** to insert the element. If you want to view the overall structure of a document and see what is allowed (and where), you can use 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 of it. Your document will be invalid until you create those elements. Oxygen XML Editor plugin does 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 Editor plugin creates it for you. However, if there is more than one possibility you have to create the appropriate elements yourself. In many cases, Oxygen XML Editor plugin presents *XML Quick Fixes* that help you resolve errors by offering proposals to quickly fix problems such as missing required attributes or invalid elements.

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 Editor plugin 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 Editor plugin provides many tools to help you edit your structure and to keep your structure valid while editing text.

The Document Breadcrumbs

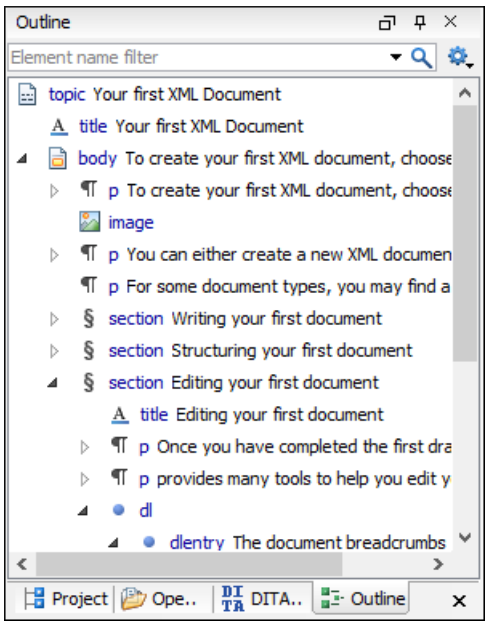
Across the top of the editor window (in **Text** mode), there is a set of breadcrumbs that shows you exactly were the insertion point is in the structure of the document. You can click any element in the breadcrumbs to select that entire element in the document.

Showing Tags

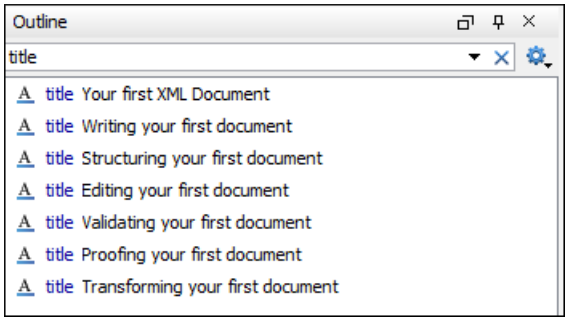
To see exactly where you are in the structure of the document, you can show the tags graphically in the **Author** view. There are several levels of tag visibility that you can choose using the  **Display tags mode** drop-down menu on the toolbar (the button may look a little different as it changes 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 it to select elements, or to move elements around in the document.



You can configure the **Outline** view to determine what is shown, such as element names, attributes, and comments. Certain choices may work better for particular document types. You can also filter the **Outline** view 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. 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 doing this correctly is to make sure that you pick up the right block of text in the first place. Using the breadcrumbs or **Outline** view, or showing tags and using them to select content, can 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 is not valid, Oxygen XML Editor plugin warns you and tries to suggest actions that make it valid (such as 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).

If you are using **Author** mode, you can also switch to **Text** mode 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 in the **Refactoring** menu of the contextual menu. More advanced refactoring operations are also available using the ***XML Refactoring*** *tool* that is available in the **XML Tools** menu.

Validating Your First Document

Validation is the process of making sure that an XML document abides by the rules of its schema. If Oxygen XML Editor plugin knows how to find the schema, it validates the document for you as you type. Oxygen XML Editor plugin finds the schema automatically for most of the document types created from templates. However, in some cases you may have to *tell it how to find the schema*.

When Oxygen XML Editor plugin validates as you type, there is a small bar at the right edge of the editor that shows you if the document is invalid and where 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 flag shows you where the errors are found. Click that flag to jump to the error. Remember that sometimes your document is invalid simply because the structure you are creating is not yet complete.

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

You can also validate your document at any time by selecting the **Validate** action from the **Validation** toolbar drop-down menu or the **XML** menu.. When you validate in this manner, if errors are found, the validation result opens in a new pane at the bottom of the editor that shows each validation error on a separate line. Clicking the error takes you to the location in your document where the error was detected.

Note: Be aware that the problem is sometimes in a different location from where the validator detects the error. To get more information about a validation error, right-click a validation error message, and select **Show Message**.

Proofing Your First Document

Oxygen XML Editor plugin includes an *automatic (as-you-type) spell checking feature*, as well as a manual spell checking action. To check the spelling of your document manually, use the **Check Spelling** action on the toolbar.

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. For example, 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 appropriate for your document type and the kind of output you want to generate.

Oxygen XML Editor plugin 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 Editor plugin or ones created by your organization.

To see the list of transformations available for your document, select the **Apply Transformation Scenario(s)** action from the toolbar or the **XML** menu. A list of available transformation scenarios are displayed. Choose one or more scenarios to apply, and click **Apply associated**. Exactly how your transformed content appears depends on how the transformation scenario is configured.

Your First DITA Topic

To create your first DITA topic, select **File** > **New** > **Other** > **Oxygen XML Editor plugin**, or click the **New** button on the toolbar, and select **New from Templates**. The ***New Document Wizard*** is displayed:

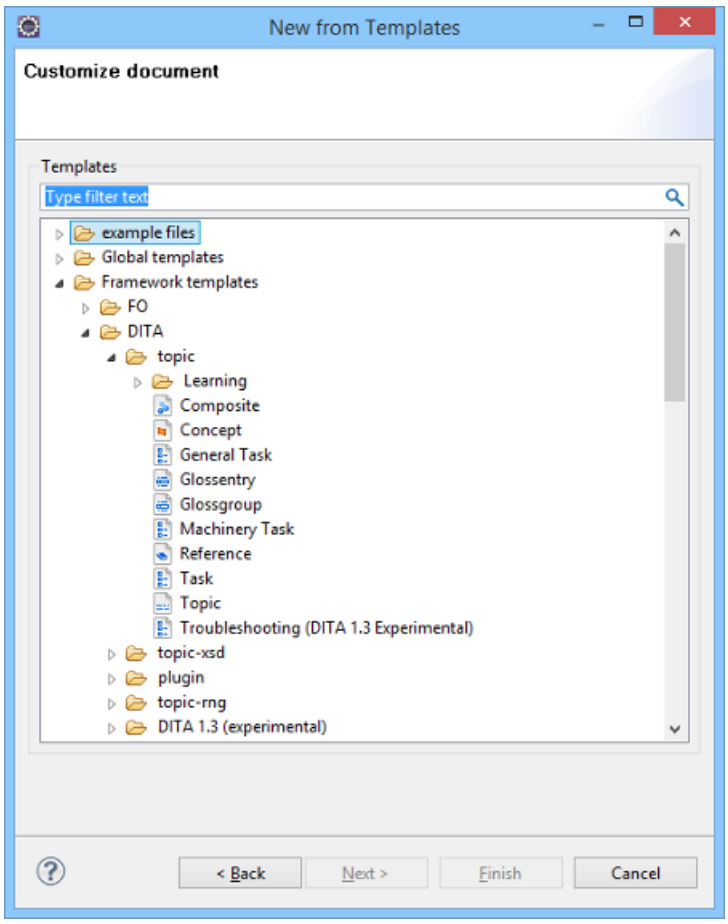


Figure 2: New from Templates Wizard

Go to **Framework templates** > **DITA** > **topic** and select the type of topic that you want to create.

Note: If your organization has created DITA customizations, the appropriate template files may be in another location, and various different types of topics 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, thus all *the editing features that Oxygen XML Editor plugin provides for editing XML documents* also apply to DITA topics. Oxygen XML Editor plugin also provides extensive additional support for *editing DITA topics*, *their associated DITA maps*, and for *creating DITA output*.

Understanding DITA Topics

It is important to understand the role that a *DITA topic plays in a DITA project*. A DITA topic is not associated with a single published document. It is a separate entity that can potentially be included in many different books, help systems, or websites. Therefore, when you write a DITA topic you are not writing a book, a help system, or a website. You are writing an individual piece of content. This affects how you approach the writing task and how Oxygen XML Editor plugin works to support you as you write.

Most of your topics are actually related to other topics, and those relationships can affect how you write and handle things such as links and content reuse. Oxygen XML Editor plugin helps you manage those relationships. Depending on how your topics are related, you can use the tools provided in Oxygen XML Editor plugin, along with the features of DITA, in a variety of different ways.

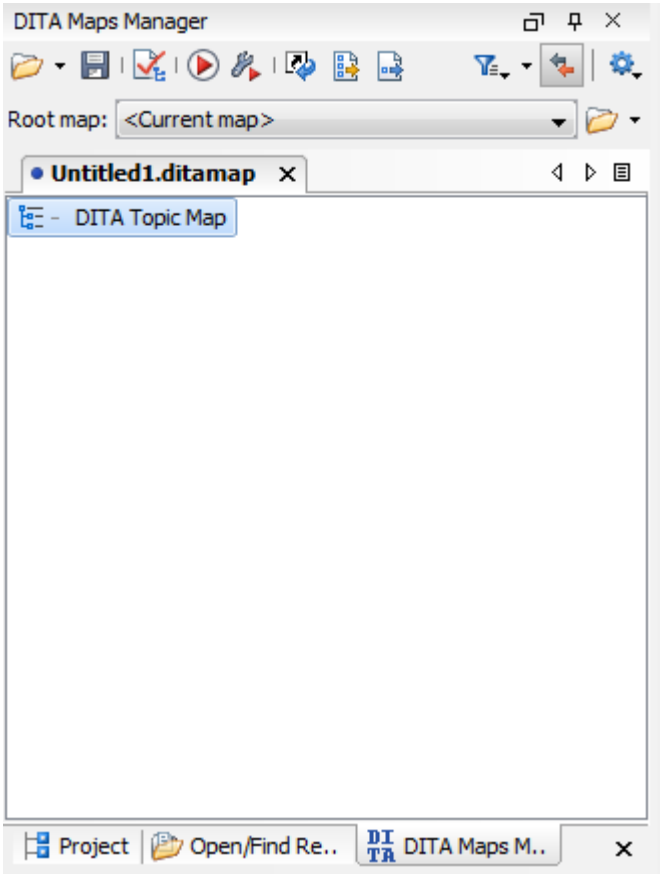
The Role of Maps

The basic method that DITA uses to express the relationship between topics is *through a DITA map*. Other relationships between topics, such as cross references, generally need to be made between topics in the same map. DITA uses maps to determine which topics are part of any output that you create. While customized DITA solutions can use other mechanisms, generally DITA is not used as a way to publish individual topics. Output is created from a map and includes all 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 organize the chapters 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 in a help system. This allows you to reuse the content of a map in multiple projects.

Creating a Map

To add topics to a map, you must first *create the map*. A map is an XML document, similar to a topic. To create a map, select **File** > **New** > **Other** > **Oxygen XML Editor plugin**, or click the **New** button on the toolbar, select **New from Templates**, go to **Framework templates** > **DITA Map** > **map** and select the type of map you want to create. Oxygen XML Editor plugin asks if you want to open your map in the editor or in the **DITA Maps Manager**. Usually, opening it in the **DITA Maps Manager** is the best choice, but you can also open the map in the editor from the **DITA Maps Manager**. The **DITA Maps Manager** presents a view of the DITA map that is similar to a table of contents.



Adding Topics to a Map

To *add a topic to a map*, add a topic reference to the map using a `topicref` element. The easiest way to do this is to open the topic in the editor, then right-click the DITA map in the **DITA Maps Manager** view and choose **Reference to the currently edited file** from the **Append child** or **Insert After** submenus. This opens the ***Insert Reference** dialog box* with all of the required fields already filled in for you. You can fill in additional information in the various tabs in

this dialog box or add it to the map later. When you select **Insert and close**, a reference to your topic is added to the map.

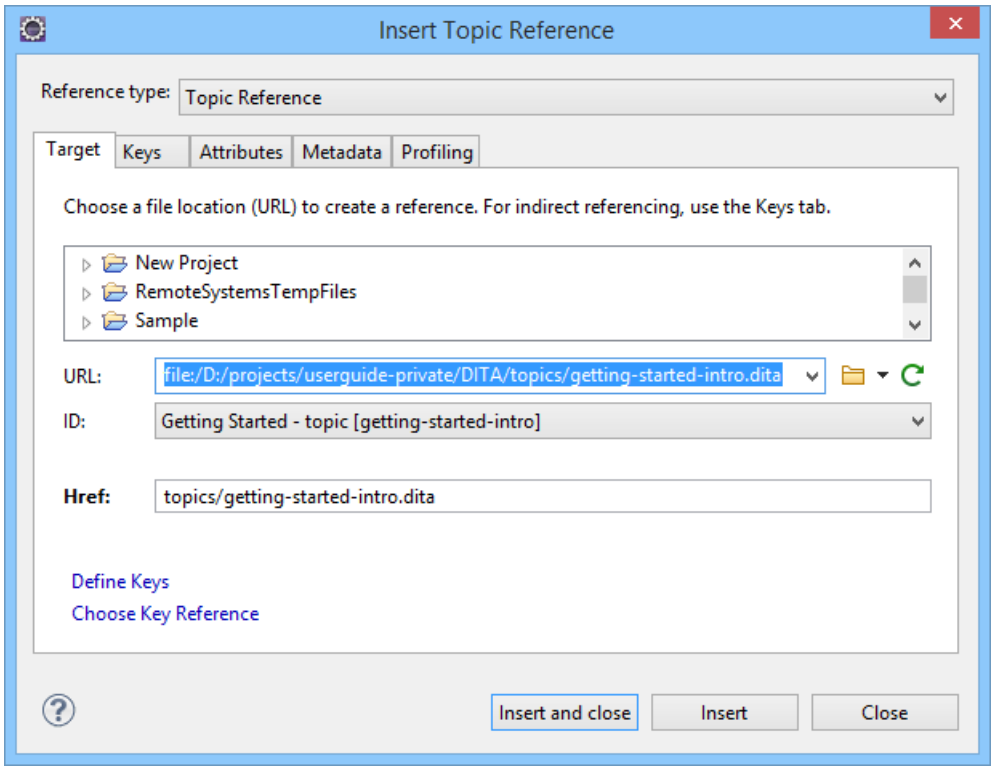


Figure 3: Insert Reference Dialog Box

If you want to see what the resulting map looks like in XML, save your map and then double-click the DITA map in the **DITA Maps Manager** view. The XML version of the map opens in the editor.

As you add topics to your map, you may want to make a topic the child or sibling of another topic. This is usually done at the map level. To create a child topic reference, right-click the parent topic in the **DITA Maps Manager** view and choose **Append child**. To create a sibling topic reference, right-click a topic in the **DITA Maps Manager** view and choose **Insert After**. From either of these submenus you can then choose one of the following options:

- **New** - Opens the ***New*** *file wizard* for creating a new topic.
- **Reference** - Opens the ***Insert Reference*** *dialog box* that allows you to create a reference to an existing topic.
- **Reference to the currently edited file** - Opens the ***Insert Reference*** *dialog box* that helps you to easily create a reference to the file that is currently opened in the editor.

You can also change the order and nesting of topics in the **DITA Maps Manager** view by doing either of the following:

- Select the topic to move while holding down the **Alt** key and use the arrow keys to move it around.
- Use the mouse to drag and drop the topic to the desired location.

The way your parent and child topics are organized in any particular output depends on both the configuration of those topics in the map and the rules of the output transformation that is applied to them. Do not assume that your topics must have the same organization for all output types. The map defines the organization of the topics, not the topics themselves. It is possible to create a variety of different maps, each with different organization and configuration options to produce a variety of different outputs.

Child Maps

If you have a large set of information, such as a long book or extensive help system, a single map can become long and difficult to manage. To make it easier to manage, you can *break up the content into smaller sub-maps*. A sub-map might represent a chapter of a book, a section of a user manual, or a page on a website.

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 the current map, right-click the parent DITA map and choose **Append child** > **Map reference**.

Validating a Map

Just as it is with your individual topics, it is important to *validate your maps*. Oxygen XML Editor plugin provides a validation function for DITA maps that does more than simply validating that the XML is well formed. It also does the following:

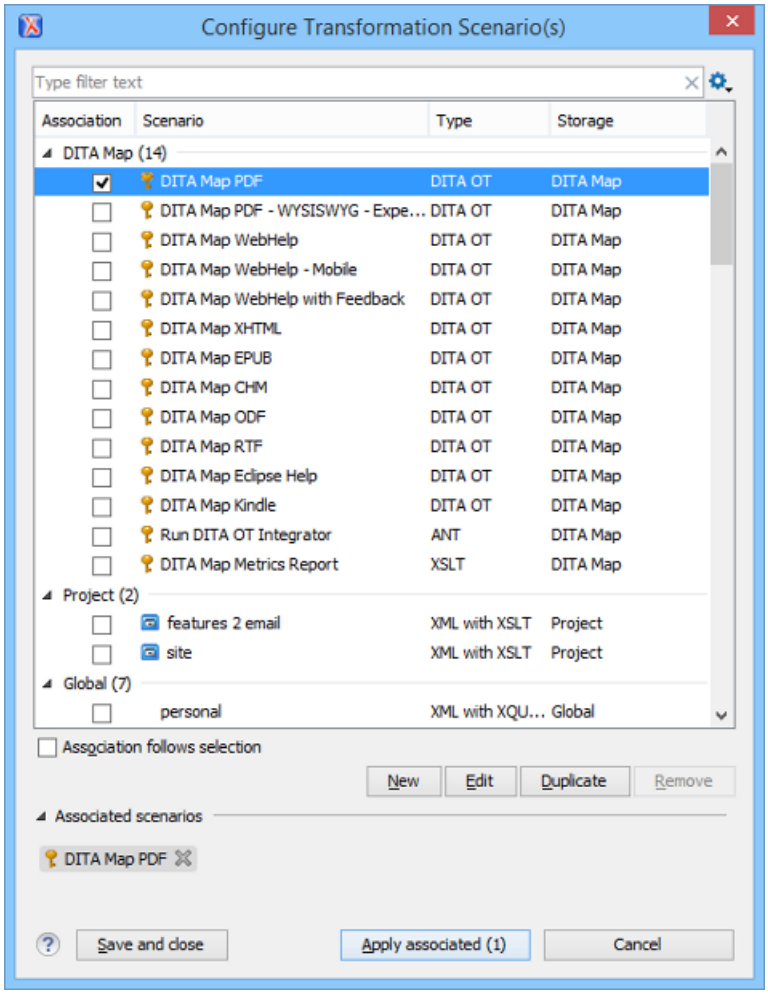
- Validates all of the relationships defined in the maps.
- Validates all of the files that are included in the map.
- Validates all of the links that are expressed in the files.

Validating the map that describes your entire publication 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 previously, in DITA standards you usually do not publish output from an individual topic. Instead, you *create published output* by running a DITA transformation on a map. This collects all the topics that are referenced in the map, organizes them, and produces output in a particular format. By default, Oxygen XML Editor plugin uses the transformations provided by the **DITA Open Toolkit** for publishing to a variety of different output formats (such as PDF, WebHelp or EPUB). Your organization may have created various custom transformations or modified the built-in **DITA Open Toolkit** transformations. In either case, Oxygen XML Editor plugin manages them by using transformation scenarios.

To publish output for a map, select the transformation scenario you want to run and set any of the parameters it requires. To select a transformation, click the  **Configure Transformation Scenario(s)** button in the **DITA Maps Manager** view. This opens the **Configure Transformation Scenario(s)** dialog box.



Choose the transformation scenarios you want to apply and click **Apply associated**. Depending on the configuration of the transformation scenario, when the transformation is finished, your output may automatically be opened in the appropriate application. To change or view the configuration or storage options for a transformation scenario, select the transformation and click **Edit**.

Creating a New Project

Oxygen XML Editor plugin allows you to organize your XML-related files into projects. This helps you manage and organize your files and also allows you to perform batch operations (such as validation and transformation) over multiple files. Use the ***Navigator** view* to manage projects, and the files and folders contained within.

Creating a New Project

To create a new project, select **New** > **XML Project** or **New** > **Sample XML Project** from the contextual menu or **File** menu. This opens a dialog box that allows you to create and customize a new project and adds it to the structure of the project in the ***Navigator** view*.

Adding Items to the Project

To add items to the project, select the desired document type or folder from the **New** menu of the contextual menu, when invoked from the **Navigator** view (or from the **File** menu). You can also create a document from a template by selecting **New** > **New from Templates** from the contextual menu.

Using Linked Folders (Shortcuts)

Another easy way to organize your XML working files is to place them in a directory and then to create a corresponding linked folder in you project. If you add new files to that folder, you can simply use the **Refresh (F5)** action from

the toolbar or contextual menu and the **Navigator** view will display the existing files and subdirectories. If your files are scattered amongst several folders, but represent the same class of files, you might find it useful to combine them in a logical folder.

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

Note: Files may have multiple instances within the folder system, but cannot appear twice within the same folder.

For much more information on managing projects and their content, see the *The Navigator View* on page 59 section.

Getting Help

If you run into specific problems while using Oxygen XML Editor plugin you can take advantage of a variety of support related resources. Those resources include the following:

- *The Oxygen XML Editor plugin Support Section of the Website*
- *The Oxygen XML Editor plugin Forum*
- *The Oxygen XML Editor plugin Video Tutorials*
- *The Common Problems and Solutions Section of the User Manual*
- *The Online Technical Support Form*

The application also includes various specific help-related resources in the **Help** menu.

Chapter 3

Installation

Topics:

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

Installation Options for Oxygen XML Editor plugin

Choosing an installer

You have a choice of installers;

- The Update Site installer
- The Zip archive installer

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

Choosing a license option

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

You can choose from two kinds of license:

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

Upgrading, transferring, and uninstalling.

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

Getting help with installation

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

Install Oxygen XML Editor plugin on Windows

Choosing an Installer

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

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

System Requirements

System requirements for a Windows install:

Operating systems

Windows Vista, Windows 7, Windows 8, Windows 10, Windows Server 2008, Windows Server 2012

CPU

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

Memory

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

Storage

- Minimum - 400 MB free disk space

- Recommended - 1 GB free disk space

Java

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

Eclipse Plugin Installation - The Update Site Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.5. The steps for installing the Eclipse plugin with the Update Site method are as follows:

1. Start Eclipse.
2. Go to **Help** > **Install New Software** > **Available Software**.
3. Click **Add** in the **Available Software** dialog box.
4. Enter `http://www.oxygenxml.com/InstData/Editor/Eclipse/site.xml` into the **Location** field of the **Add Site** dialog box.
5. Click **OK**.
6. Select the **Oxygen XML Editor plugin** checkbox.
7. Click **Next >** and continue with the rest of the installation wizard.
8. Restart Eclipse when prompted.
9. Verify that Oxygen XML Editor plugin is installed correctly by creating a new XML Project. Go to **File** > **New** > **Other** and choose **Oxygen XML Editor plugin** > **XML Project**.
10. When prompted for a license key, enter the license information received in the registration email.

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

Eclipse Plugin Installation - The Zip Archive Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.5. The steps for installing the Eclipse plugin with the Zip Archive method are as follows:

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

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

Install Oxygen XML Editor plugin on Mac OS X

Choosing an Installer

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

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

System Requirements

System requirements for a Mac OS X install:

Operating system

OS X version 10.6 64-bit or later

CPU

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

Memory

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

Storage

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

Java

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

Eclipse Plugin Installation - The Update Site Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.5. The steps for installing the Eclipse plugin with the Update Site method are as follows:

1. Start Eclipse.
2. Go to **Help** > **Install New Software** > **Available Software**.
3. Click **Add** in the **Available Software** dialog box.
4. Enter `http://www.oxygenxml.com/InstData/Editor/Eclipse/site.xml` into the **Location** field of the **Add Site** dialog box.
5. Click **OK**.
6. Select the **Oxygen XML Editor plugin** checkbox.
7. Click **Next >** and continue with the rest of the installation wizard.
8. Restart Eclipse when prompted.
9. Verify that Oxygen XML Editor plugin is installed correctly by creating a new XML Project. Go to **File** > **New** > **Other** and choose **Oxygen XML Editor plugin** > **XML Project**.
10. When prompted for a license key, enter the license information received in the registration email.

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

Eclipse Plugin Installation - The Zip Archive Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.5. The steps for installing the Eclipse plugin with the Zip Archive method are as follows:

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

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

Install Oxygen XML Editor plugin on Linux

Choosing an Installer

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

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

System Requirements

System requirements for a Linux install:

Operating system

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

CPU

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

Memory

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

Storage

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

Java

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

Eclipse Plugin Installation - The Update Site Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.5. The steps for installing the Eclipse plugin with the Update Site method are as follows:

1. Start Eclipse.
2. Go to **Help** > **Install New Software** > **Available Software**.
3. Click **Add** in the **Available Software** dialog box.
4. Enter `http://www.oxygenxml.com/InstData/Editor/Eclipse/site.xml` into the **Location** field of the **Add Site** dialog box.
5. Click **OK**.
6. Select the **Oxygen XML Editor plugin** checkbox.
7. Click **Next >** and continue with the rest of the installation wizard.
8. Restart Eclipse when prompted.
9. Verify that Oxygen XML Editor plugin is installed correctly by creating a new XML Project. Go to **File** > **New** > **Other** and choose **Oxygen XML Editor plugin** > **XML Project**.
10. When prompted for a license key, enter the license information received in the registration email.

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

Eclipse Plugin Installation - The Zip Archive Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.5. The steps for installing the Eclipse plugin with the Zip Archive method are as follows:

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

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

Site-wide Deployment

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

Creating custom default options

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

Creating default project files

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

Shared project files

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

Using floating licenses

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

Obtaining and Registering a License Key for Oxygen XML Editor plugin

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

For demonstration and evaluation purposes, a time limited license is available upon request at *http://www.oxygenxml.com/register.html*. 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 Editor plugin:

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

2. A floating license may be used by any user on any machine. However, the total number of copies of Oxygen XML Editor plugin in use at one time must not be more than the number of floating licenses available. A user who runs two different distributions of Oxygen XML Editor plugin (for example Standalone and Eclipse Plugin) at the same time on the same computer, consumes a single floating license.
3. A special academic license (classroom, department or site license) may be used by students and teachers in academic institutions. These licenses provide a cost effective way of getting access to Oxygen XML Editor plugin for learning purposes.

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

Obtaining a License

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

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

Register a Named-User License

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

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

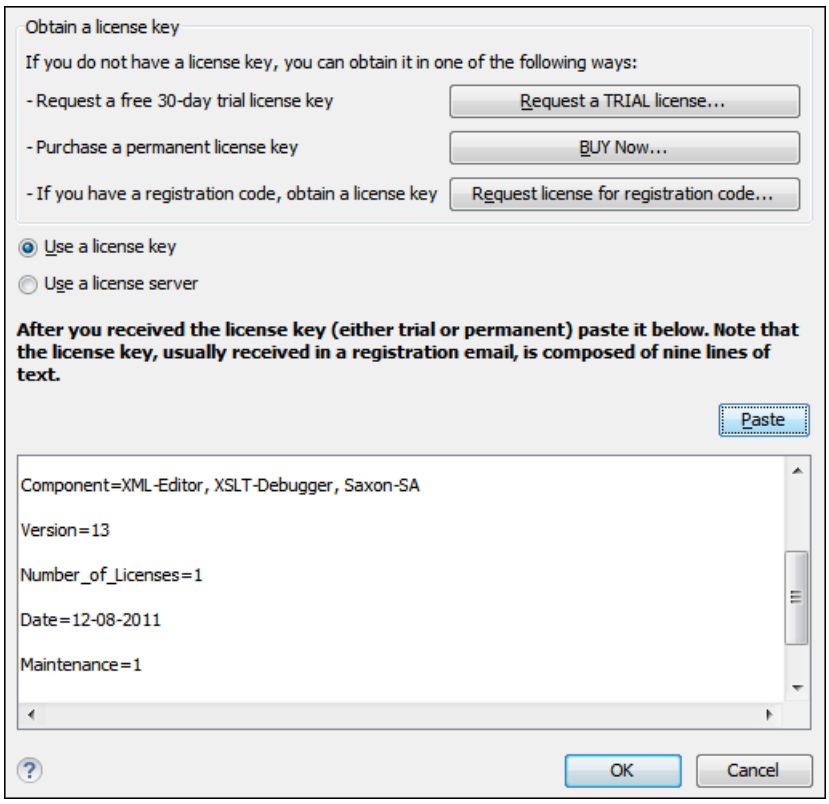


Figure 4: License Registration Dialog Box

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

Registering a Floating License

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

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

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

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

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

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

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

1. Contact your server administrator to get network address and login details for the license server.
2. Start the Eclipse platform.
3. *Open the **Preferences** dialog box* and click the **Register** button.
   The license registration dialog box is displayed.

4. Choose **Use a license server** as licensing method.
5. Select **Standalone server** as server type.
6. In the *Host* field enter the host name or IP address of the license server.
7. In the *Port* field enter the port number used to communicate with the license server.
8. Click the **OK** button.

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

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

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

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

Release a Floating License

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

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

Register a Floating License for Multiple Users

If you are an administrator registering floating licenses for multiple users, you can avoid having to open Oxygen XML Editor plugin on each machine and configuring the registration details by using the following procedure:

1. Reset the registration details:
   a. Select **Register** from the **Help** menu.
   b. Click **OK** without entering any information in this dialog box.
   c. Click **Reset** and restart the application.
2. Register the license using one of the *floating license registration procedures*.
3. Copy the `license.xml` file from the Oxygen XML Editor plugin *preferences directory* to the `lib` sub-folder of the installation folder on each installation to be registered.

Setting Up a Floating License Server for Oxygen XML Editor plugin

Determine if you need to set up a license server

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

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

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

Activating Floating License Keys

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

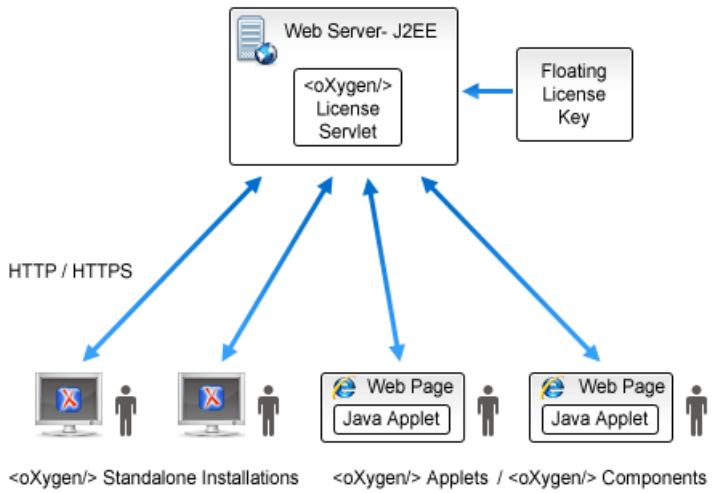
During the activation process, a code that uniquely identifies your license server deployment is sent to the Oxygen XML Editor plugin servers, which in turn will sign the license key.

Split or combine license keys to work with your license servers

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

Setting up a Floating License Server Running as a Java Servlet

Setting up the floating license server as a servlet.



Steps for Installing the Floating License Server as a Servlet

1. Make sure that Apache Tomcat 5.5 or higher is running on the machine you have selected to be the license server. To get it, go to *http://tomcat.apache.org*.
2. Download the **Web ARchive** (**.war**) license servlet from the *Oxygen XML Editor plugin website*.
3. Configure Tomcat to use a security Realm element. Refer to the *Tomcat Documentation* for more information.

4. Edit the `tomcat-users.xml` file from your Tomcat installation and configure one user for each of the following roles: *standard*, *admin*, and *manager*.
5. Go to the Tomcat Web Application Manager page and log-in with the user you configured with the *manager* role. In the **WAR file to deploy** section, choose the WAR file and click the **Deploy** button. The *oXygen License Servlet* is now up and running, but the license keys are not yet registered.
6. Activate the license key. This process involves binding your license key to your license server deployment. Once the process is completed you cannot activate the license with another license server. Follow these steps to activate the license:

   a. Access the license servlet by following the link provided by the Tomcat Web Application Manager page. If prompted for authentication, use the credentials configured for the *admin* or *manager* users.

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

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

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

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

   c. Press **Activate**.

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

7. By default, the license server logs its activity in the `/usr/local/tomcat/logs/oxygenLicenseServlet.log` file. To change the log file location, edit the `log4j.appender.R2.File` property from the `/usr/local/tomcat/webapps/oXygenLicenseServlet/WEB-INF/lib/log4j.properties` configuration file.

Manual License Activation Procedure

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

Upgrading Your Floating License Servlet

The goal of the following procedure is to help you minimize the downtime when you upgrade the Oxygen XML Editor plugin floating license servlet to its latest version.

Follow this procedure:

1. Access the license servlet by following the link provided by the Tomcat Web Application Manager page. If prompted for authentication, use the *admin* or *manager* credentials.
2. Click the **View license key** link and copy the displayed license key to a file for later use.
3. Go to the Tomcat Web Application Manager page, log in with the user you configured with the *manager* role, and *Undeploy* the floating license servlet.
4. Go to Oxygen XML Editor plugin website and *download the license servlet*.
5. Deploy the downloaded license servlet.

6. Access the license servlet by following the link provided by the Tomcat Web Application Manager page. If prompted for authentication, use the credentials configured for the *admin* or *manager* users.
7. Paste the license key into the form and register it.

Replacing a Floating License Key in a Floating License Servlet

The following procedure assumes that your Oxygen XML Editor plugin floating license servlet contains a previously *activated license key* and provides instructions for replacing it with another one. The goal of the procedure is to minimize the license servlet downtime during the activation step of the new license key.

This is useful if, for instance, you want to upgrade your existing license to the latest version or if you receive a new license key *that accommodates a different number of users*.

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

1. Access the license servlet by following the link provided by the Tomcat Web Application Manager page.
2. Click the **Replace license key** link. This will open a page that contains details about the license currently in use.
3. Click the **Yes** button to begin the replacement procedure.

   Note: During the replacement procedure, new instances of Oxygen XML Editor plugin cannot be licensed by the servlet until the process is completed.

4. Paste the new floating license key in the displayed form, then click **Submit**. The browser used in the process needs to have Internet access.

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

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

5. Press **Activate**.

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

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

Getting More Information From the Report Page

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

It displays the following real time information:

- **License load** - A graphical indicator that shows how many licenses are available. When the indicator turns red, there are no more licenses available.
- **Floating license server status** - General information about the license server status, including the following information:
  - server start time
  - license count
  - rejected and acknowledged requests
  - average usage time
  - license refresh and timeout intervals
  - location of the license key
  - server version
- **License key information** - License key data, including the following information:

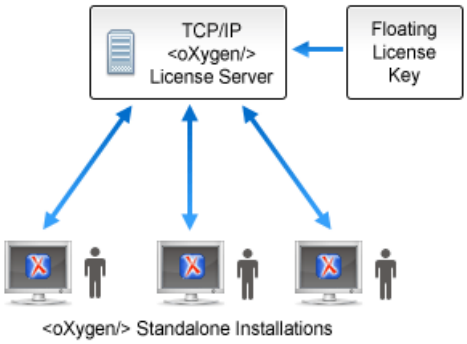
- licensed product
- registration name
- company name
- license category
- number of floating users
- Maintenance Pack validity

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

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

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

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



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

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

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

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

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

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

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

3. Manually add the oXygenLicenseServer.exe file in the Windows Firewall list of exceptions. Go to **Control Panel** > **System and Security** > **Windows Firewall** > **Allow a program or feature through Windows Firewall** > **Allow another program** and browse for oXygenLicenseServer.exe from the Oxygen XML Editor plugin License Server installation folder.
4. Floating licenses require activation prior to use. More details are available either on-screen (if the license server is started in a *command line interface*) or in the `outLicenseServer.log` log file.

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

Upgrading Your Floating License Server

The goal of the following procedure is to help you minimize the downtime generated when you upgrade the Oxygen XML Editor plugin floating license server to its newest version.

Follow this procedure:

1. Go to the Oxygen XML Editor plugin website and download the *latest floating license server*.
2. Run the installation kit.
3. Leave the default **Update the existing installation** option enabled. This will ensure that some options set in the previous version (namely the installation folder, port number, and the floating license key in use) of the license server will be preserved.
4. Follow the on-screen instructions to complete the installation process.

Replacing a Floating License Key in a Floating License Server

The following procedure assumes that your Oxygen XML Editor plugin floating license server contains a previously *activated license key* and provides instructions for replacing the activated license key with another one. The goal of the procedure is to minimize the license servlet downtime during the activation step of the new license key.

This is useful if, for instance, you want to upgrade your existing license to the latest version or if you receive a new license key *that accommodates a different number of users*.

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

1. Stop the service that runs the floating license server.
2. Locate the folder that holds the previous activated license key (by default, it is named `license` and it is located in the installation directory of the license server).
3. Remove the `license.txt` file and try to restart the server. Since the file that stores the license key is missing, *the server will fail to start*.
4. Find the license activation procedure in the on-screen instructions (if the license server is started in a *command line interface*) or in the `outLicenseServer.log` log file.
5. After you copy the activated license key in the `license.txt` file, restart the license server.

Common Problems

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

Windows Service Reports "Incorrect Function" When Started

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

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

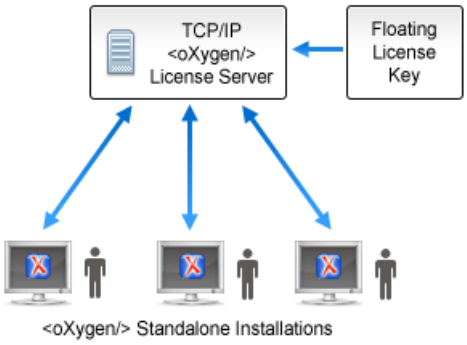
When Started, the Windows Service Reports "Error 1067: The Process Terminated Unexpectedly"

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

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

Setting up a Floating License Server Running as a Standalone Process Using a Platform-independent Distribution

This installation method can be used for running the license server on any platform where a Java virtual machine can run (OS X, Linux/Unix, Windows).



Steps for Installing the Floating License Server as a Standalone Process with a Zip Archive

1. Ensure that a Java runtime version 6 or later is installed on the server machine.
2. Download the license server installation kit for your platform from the *Oxygen XML Editor plugin website*.
3. Unzip the installation kit into a new folder.
4. Start the server using the startup script from a command line console.

   The startup script is called `licenseServer.sh` for OS X and Unix/Linux or `licenseServer.bat` for Windows. The following parameters are accepted:

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

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

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

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

Upgrading Your Floating License Server

The goal of the following procedure is to help you minimize the downtime generated when you upgrade the Oxygen XML Editor plugin floating license server to its newest version.

Follow this procedure:

1. Stop the current license server process.
2. Locate and open the floating server startup script. It should look like:

   ```
   sh licenseServer.sh pathToLicenseDir 54321
   ```

3. Make a note of the path to the license directory (in our example is `pathToLicenseDir`) and the port number (in our example is `54321`).
4. Go to the license directory and copy the license key file (`license.txt`) for later use.

5. Go to the Oxygen XML Editor plugin website and download the *all-platforms floating license server installation kit*.
6. Unzip the archive and overwrite the content of your current floating license server installation.
7. Copy the license key file (`license.txt`) saved in step 4 to `license` directory of the floating license server installation.
8. Edit the floating server startup script and configure with the info you made note of in step 3.
9. Start the floating license server process.

Replacing a Floating License Key in a Floating License Server

The following procedure assumes that your Oxygen XML Editor plugin floating license server contains a previously *activated license key* and provides instructions for replacing the activated license key with another one. The goal of the procedure is to minimize the license servlet downtime during the activation step of the new license key.

This is useful if, for instance, you want to upgrade your existing license to the latest version or if you receive a new license key *that accommodates a different number of users*.

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

1. Stop the process that runs the floating license server.
2. Locate the folder that holds the previous activated license key (by default, it is named `license` and it is located in the installation directory of the license server).
3. Remove the `license.txt` file and try to restart the server. Since the file that stores the license key is missing, the server will fail to start.
4. Find the license activation procedure in the on-screen instructions.
5. After you copy the activated license key in the `license.txt` file, restart the license server.

Transferring or Releasing a License Key

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

1. *Open the **Preferences** dialog box* and click **Register**.
   The license registration dialog box is displayed.
2. The license key field should be empty (this is normal). If it is not empty, delete any text in the field.
3. Make sure the option **Use a license key** is selected.
4. Click **OK**.
   A dialog box is displayed asking if 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 a *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 Editor plugin.

Upgrading Oxygen XML Editor plugin

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

Checking for New Versions of Oxygen XML Editor plugin

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

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

What is Preserved During an Upgrade?

When you install a new version of Oxygen XML Editor plugin, some data is preserved and some is overwritten. If there is a previous version of Oxygen XML Editor plugin already installed on your computer, it can coexist with the new one, which means you do not 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 *document type (framework) files*, XSLT stylesheets, XML catalogs, and templates.
- All global user preferences are preserved in the new version.
- All project preferences will be preserved in their project files.
- Any custom frameworks that were stored outside the installation directory (as configured in ***Document type associations*** > ***Locations***) will be preserved and will be found by the new installation.

If you install in a new directory.

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

Upgrading the Eclipse Plugin

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

Uninstalling Oxygen XML Editor plugin

Uninstalling the Eclipse plugin

Caution:

The following procedure will remove Oxygen XML Editor plugin from your system. It will not remove the Eclipse platform. If you want to uninstall Eclipse, refer to its uninstall instructions.

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

`.com.oxygenxml` of the user home directory on Linux / the subfolder `Library/Preferences/com.oxygenxml` of the user home folder on Mac OS X.

Chapter 4

Perspectives

Topics:

The Oxygen XML Editor plugin interface uses standard interface conventions and components to provide a familiar and intuitive editing environment across all operating systems. There are several perspectives that you can use to work with documents in Oxygen XML Editor plugin. You can change the perspective by selecting the perspective from the **Window** > **Open Perspective** menu.

<oXygen/> XML Perspective

The **<oXygen/> XML** perspective is the most commonly used perspective and it is the default perspective when you start Oxygen XML Editor plugin for the first time. It is the perspective that you will use to edit the content of your XML documents.

To switch the focus to this perspective, select **<oXygen/> XML** from the **Window** > **Open Perspective** menu.

The layout of this perspective is composed of the following components:

Menus

Provides menu driven access to all the features and functions available in Oxygen XML Editor plugin. Most of the menus are common for all types of documents. However, Oxygen XML Editor plugin also includes some context-sensitive and framework-specific menus that are only available for a specific context or type of document.

Toolbars

Provides easy access to common and frequently used functions. Each icon is a button that acts as a shortcut to a related function. Most of the toolbars are common for all types of documents. However, **Author** mode also includes framework-specific toolbars, depending on the type of document that is being edited (for example, if you are editing a DITA document, a *DITA Author Custom Actions* toolbar is available that includes operations that are specific to DITA documents).

Editor Pane

The main editing pane where you spend most of your time reading, editing, applying markup, and validating your documents.

Views

Oxygen XML Editor plugin includes a large variety of views to assist you with editing, viewing, searching, validating, transforming, and organizing your documents. The most commonly used views are displayed by default and you can choose to display others by selecting them from the **Window** > **Show View** menu.

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 top-right corner of the view (▭).

Some of the most helpful views in the **<oXygen/> XML** perspective include the following:

- ***Navigator** view* - Enables the definition of projects and logical management of the documents they contain.
- ***DITA Maps Manager** view* - For DITA frameworks, this view helps you organize, manage, and edit DITA topics and maps.
- ***Outline** view* - It provides an XML tag overview and offers a variety of functions, such as modifications follow-up, document structure change, document tag selection, and elements filtering.
- ***Results view*** - Displays the messages generated as a result of user actions such as *validations*, *transformation scenarios*, *spell checking in multiple files*, search operations, and others. Each message is a link to the location related to the event that triggered the message.
- ***Attributes** view* - Presents all possible attributes of the current element and allows you to edit attribute values. You can also use this view to insert attributes in **Text** mode. **Author** mode also includes an *in-place attribute editor*.
- ***Model** view* - Presents the current edited element structure model and additional documentation as defined in the schema.
- ***Elements view*** - Presents a list of all defined elements that you can insert at the current cursor position according to the document's schema. In ***Author** mode this view* includes tabs that present additional information relative to the cursor location.
- ***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.
- ***XPath Results** view* - Displays the results from running an XPath expression.
- ***Text** view* - Displays the text output that is produced in XSLT transformations.

- ***Browser** view* - Displays HTML output from XSLT transformations.
- **Problems** view - A general Eclipse view that displays system-generated errors, warnings, or information associated with a resource.
- ***Console** view* - Status information generated by the Schema detection, validation, and transformation threads.
- ***WSDL SOAP Analyzer** view* - Provides a tool that helps you test if the messages defined in a Web Service Descriptor (WSDL) are accepted by a Web Services server.

Supported Document Types

You can use the main editing pane in the **Editor** perspective to edit a large variety of types of documents. You can see that a document is associated with Oxygen XML Editor plugin by the special icons displayed in the tabs of the editor title bar. The supported document types include:

-  - XML documents
-  - XSLT stylesheets
-  - XML Schema
-  - DTD (Document Type Definition) schemas
-  - RELAX NG full syntax schemas
-  - RELAX NG compact syntax schemas
-  - NVDL (Namespace-based Validation Dispatching Language) schemas
-  - XSL:FO documents
-  - XQuery documents
-  - WSDL documents
-  - Schematron documents
-  - JavaScript documents
-  - Python documents
-  - CSS documents
-  - LESS documents
-  - XProc scripts
-  - SQL documents
-  - JSON documents
-  - Ant build scripts

XSLT Debugger Perspective

The XSLT Debugger perspective allows you to detect problems in an XSLT transformation by executing the process step by step. To switch the focus to this perspective, select **<oXygen/> XSLT Debugger** from the **Window** > **Open perspective** menu.

The workspace in this perspective is organized as an editing area assisted by special helper views. The editing area contains editor panels that you can *split horizontally or vertically* in a stack of XML editor panels and a stack of XSLT editor panels. The XML files and XSL files can be edited in *Text mode* only.

The layout of this perspective is composed of the following components:

Menus

Provides menu driven access to all the features and functions available in the **XSLT Debugger**.

Toolbars

Contains all actions needed in order to configure and control the debugging process.

XML Source Pane

The editing pane where you can display and edit data or document-oriented XML documents.

XSL Source Pane

The editing pane where you can display and edit XSL stylesheets.

Output View

Displays the transformed output that results from the input of a selected document (XML) and selected stylesheet (XSL) to the transformer. The result of transformation is dynamically written as the transformation is processed. There are three types of views for the output: a text view (with XML syntax highlight), an XHTML view, and one text view for each `xsl:result-document` element used in the stylesheet (if it is a XSLT 2.0 / 3.0 stylesheet).

Debugging Information Views

Presented in two panes, they display various types of information that can be used to understand the transformation process. For each information type there is a corresponding tab. While running a transformation, relevant events are displayed in the various information views. This allows you to obtain a clear view of the transformation progress. See the *Debugging Information Views* topic for a list of all the information views (and links to more details on each view).

Note: You are able to add XPath expression automatically in the **XWatch** view using the **Watch expression** action from the contextual menu. In case you select an expression or a fragment of it and then click **Watch expression** in the contextual menu, the entire selection is presented in the **XWatch** view. Using **Watch expression** without selecting an expression displays the value of the attribute from the cursor position in the **XWatch** view. Variables detected at the cursor position are also displayed. Expressions displayed in the **XWatch** view are *normalized* (unnecessary white spaces are removed from the expression).

To watch our video demonstration about the XSLT debugging capabilities in Oxygen XML Editor plugin, go to *http://www.oxygenxml.com/demo/XSLT_Debugger.html*.

XQuery Debugger Perspective

The XQuery Debugger perspective allows you to detect problems in an XQuery transformation process by executing the process step by step in a controlled environment and inspecting the information provided in the special views. To switch the focus to this perspective, select **<oXygen/> XQuery Debugger** from the **Window** > **Open perspective** menu.

The workspace in this perspective is organized as an editing area assisted by special helper views. The editing area contains editor panels that you can *split horizontally or vertically* in a stack of XML editor panels and a stack of XQuery editor panels. The XML files and XQuery files can be edited in *Text mode* only.

The layout of this perspective is composed of the following components:

Menus

Provides menu driven access to all the features and functions available in the **XQuery Debugger**.

Toolbars

Contains all actions needed in order to configure and control the debugging process.

XML Source Pane

The editing pane where you can display and edit data or document-oriented XML documents.

XQuery Source Pane

The editing pane where you can display and edit XQuery files.

Output View

Displays the transformed output that results from the input of a selected document (XML) and selected XQuery document to the XQuery transformer. The result of transformation is dynamically written as the transformation is processed. There are two types of views for the output: a text view (with XML syntax highlight) and an XHTML view.

Debugging Information Views

Presented in two panes, they display various types of information that can be used to understand the transformation process. For each information type there is a corresponding tab. While running a transformation, relevant events are displayed in the various information views. This allows you to obtain a clear view of the transformation progress. See the *Debugging Information Views* topic for a list of all the information views (and links to more details on each view).

Note: You are able to add XPath expression automatically in the **XWatch** view using the **Watch expression** action from the contextual menu. If you select an expression, or a fragment of it, and then click **Watch expression** in the contextual menu, the entire selection is presented in the **XWatch** view. Expressions displayed in the **XWatch** view are normalized (unnecessary white spaces are removed from the expression).

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

Oxygen XML Editor plugin Database Perspective

The Database perspective 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. To switch the focus to this perspective, select **<oXygen/> Database** from the **Window** > **Open perspective** menu.

This perspective offers support for the following databases:

- 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)

Note: For the databases marked with "Enterprise edition only", the XML capabilities are only available in the Enterprise edition of Oxygen XML Editor plugin. The non-XML capabilities of any database listed here are available in the Enterprise, Academic, and Professional editions of Oxygen XML Editor plugin by registering the database driver as a *Generic JDBC* type driver when defining the data source for accessing the database. For more information, see the *Configuring Relational Database Data Sources* on page 830 or *Configuring Native XML Database Data Sources* on page 850 sections.

The supported non-XML capabilities are as follows:

- 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 supported XML capabilities are as follows:

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

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

Tip: Connections configured on relational data sources can be used to import data to XML or to generate XML schemas.

The layout of this perspective is composed of the following components:

Menus

Provides menu driven access to all the features and functions available in the **XQuery Debugger**.

Toolbars

Contains all actions needed in order to configure and control the debugging process.

Editor Pane

The main editing pane where you spend most of your time reading, editing, applying markup, and validating your documents.

Data Source Explorer View

Provides browsing support for the configured connections.

Table Explorer View

Provides table content editing support for inserting new rows, deleting table rows, editing cell values, exporting to an XML file, and more.

Chapter 5

Editing Modes

Topics:

The main editing area in Oxygen XML Editor plugin includes several editing modes to suit the type of editing that you want to perform. You can easily switch between modes by clicking on the desired mode at the bottom of the main editing pane. Oxygen XML Editor plugin offers the following editing 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 Editor plugin provides the usual functionality of a plain text editor. It also includes a variety of advanced features that are unique to Oxygen XML Editor plugin.

Text Mode Editor

The **Text** mode **Author** editor in Oxygen XML Editor plugin is designed to be a simple, yet powerful, XML editor. It provides support to help you edit, transform, and debug XML-based documents. It also includes a specialized ***Content Completion Assistant***, an *Outline view*, and many other helpful features.

Navigating the Document Content in Text Mode

Using the Keyboard

Oxygen XML Editor plugin allows you to quickly navigate through a document using the **Ctrl Close Bracket (Meta Close Bracket on OS X)** key to go to the next XML node and **Ctrl Open Bracket (Meta Open Bracket on OS X)** to go to the previous one.

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. To position the cursor at the beginning or 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 Breadcrumb Helpers

A *breadcrumb* on the top stripe indicates the path from the document root element to the current element. It can also be used as a helpful tool to navigate to specific elements throughout the structure of the document.

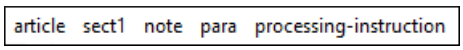


Figure 5: The Breadcrumb in Text Mode

The last element listed in the *breadcrumb* is the element at the current cursor position. Clicking an element from the *breadcrumb* selects the entire element in the editor area. Also, each element provides a contextual menu with access to the following actions:

Append Child

Allows you to select an element (from a drop-down list) that is allowed by the associated schema and inserts it as a child of the current element.

Insert Before

Allows you to select an element (from a drop-down list) that is allowed by the associated schema and inserts it immediately before the current element, as a sibling.

Insert After

Allows you to select an element (from a drop-down list) that is allowed by the associated schema and inserts it immediately after the current element, as a sibling.

Edit Attributes

Opens an editing window that allows you to edit the attributes of the currently selected element.

Toggle Comment

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

Cut

Removes the selected element and copies it to the clipboard.

Copy

Copies the selected element to the clipboard.

Delete

Deletes the currently selected element.

Text Mode Views

There is a large selection of useful views available in the **Window** > **Show View** menu. This section presents some of the most helpful views for editing in **Text** mode.

The Navigator View

The **Navigator** view is designed to assist you with organizing and managing related files grouped in the same XML project. The actions available on the contextual menu and toolbar associated to this panel enable the creation of XML projects and shortcuts to various operations on the project documents.

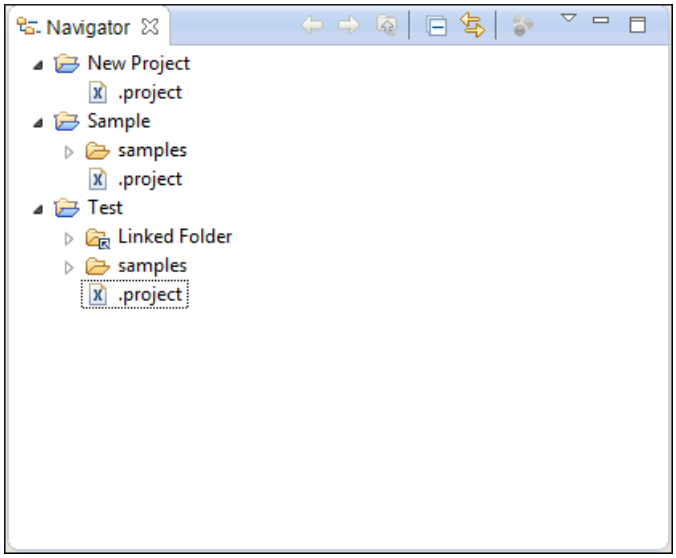


Figure 6: The Navigator View

The following actions are grouped in the upper right corner:

Collapse All

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

Link with Editor

When selected, the project tree highlights the currently edited file, if it is found in the project files.

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

View Menu

Drop-down menu that contains various settings.

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

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

menu or **File** menu. Also, the contextual menu action **Delete** can be used to remove them from the project (the local file system remains unchanged).

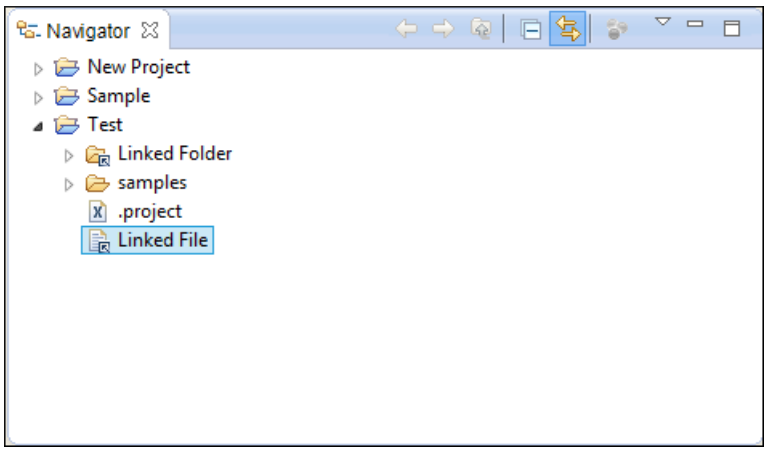


Figure 7: The Navigator View with Examples of the Two Types of Resources

Creating New Projects

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

New > XML Project

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

New > Sample XML Project

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

Creating New Project Items

To create new project items, select the desired document type or folder from the **New** menu of the contextual menu, when invoked from the **Navigator** view (or from the **File** menu). You can also create a document from a template by selecting **New** > **New from Templates** from the contextual menu.

New > File

Opens a **New** file dialog box that helps you create a new file and adds it to the project structure.

New > Folder

Opens a **New Folder** dialog box that allows you to specify a name for a new folder and adds it to the structure of the project.

New > Logical Folder

Available when invoked from the *project root*, this action creates a logical folder in the tree structure (the icon is a magenta folder on Mac OS X -).

New > Logical Folders from Web

Available when invoked from the *project root*, this action 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.

Managing Project Content

Creating/Adding Files and Folders

You can create linked folders (shortcuts) by dragging and dropping folders from the Windows Explorer / Mac OS X Finder to the project tree, or by using the contextual menu from the location in the project tree where you wanted it

added and selecting **New** > **Folder** > **Advanced**. To create a file inside a linked folder, use the contextual menu and select **New** > **File** (you can use the **Advanced** button to link to a file in the local file system).

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

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

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

Removing Files and Folders

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

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

Moving Files and Folders

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

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

Renaming Files and Folders

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

- Invoke the **Rename** action from the contextual menu.
- Press **F2** and type the new name.

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

Locating and Opening Files

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

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

Saving the Project

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

Validate Files

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

Validate > **Check Well-Formedness**

Checks if the selected file or files are well-formed.

Validate **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 **Validate with Schema**

Validates the selected file of files against a specified schema.

Validate **Configure Validation Scenario(s)**

Allows you to configure and run a *validation scenario*.

Validate > **Clear Validation Markers**

Clears all the error markers from the main editor and **Problems** view.

Applying Transformation Scenarios

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

Transform > **Apply Transformation Scenario(s)**

Obtains the output with one of the built-in scenarios.

Transform > **Configure Transformation Scenario(s)**

Opens a dialog box that allows you to configure pre-defined transformation scenarios.

Transform > **Transform with**

Allows you to select a transformation scenario to be applied to the currently selected files.

Refactoring Actions (Available for certain document types (such as XML, XSD, and XSL)

Oxygen XML Editor plugin includes some refactoring operations that help you manage the structure of your documents. The following actions are available from the contextual menu in the **Refactoring** sub-menu:

Refactoring > Rename resource

Allows you to change the name of a resource.

Refactoring > Move resource

Allows you to change the location on disk of a resource.

Refactoring > **XML Refactoring**

Opens *the XML Refactoring tool wizard* that presents refactoring operations to assist you with managing the structure of your XML documents.

Other Contextual Menu Actions

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

Open

Opens the selected file in the editor.

Open with submenu

This submenu offers you choices for opening the selected file in various editors.

Refresh

Refreshes the content and the dependencies between the resources in *the **Master Files** directory*.

XPath in Files

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

Check Spelling in Files

Allows you to *check the spelling of multiple files.*

Format and Indent Files

Opens the ***Format and Indent Files*** *dialog box* that allows you to configure the format and indent (pretty print) action that will be applied on the selected documents.

Properties

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

Outline View

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

By default it is displayed on screen, but if you closed it you can reopen it from **Window** > **Show View** > **Outline**.

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

The Outline view offers the following functionality:

- *XML Document Overview* on page 64
- *Modification Follow-up* on page 64
- *Drag and Drop Actions in Outline View* on page 64
- *Outline Filters* on page 64
- *The Contextual Menu of the Outline View* on page 65
- *Document Tag Selection* on page 66

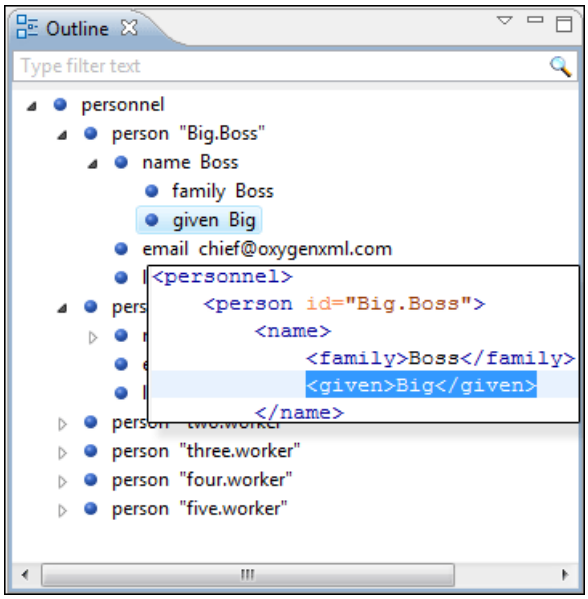


Figure 8: The Outline View

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 you to be aware of the document structure and the way tags are nested.

The **Outline** view allows you to:

- Insert or delete nodes using contextual 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 Single-Click (Command Single-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.

Drag and Drop Actions in Outline View

Entire XML elements can be moved or copied in the edited document using only the mouse in the **Outline** view with 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 a Preferences page.*

Tip: You can select and drag multiple nodes in the **Outline** view when editing in **Author** mode.

Outline Filters

The upper part of the **Outline** view contains a filter box that 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 following actions are available in the **View menu** of the **Outline** view when editing in **Text** mode:

Filter returns exact matches

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

Selection update on cursor move

Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the cursor moves or the changes in the editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

Flat presentation mode of the filtered results

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

Show comments and processing instructions

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

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

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

Configure displayed attributes

Displays the *XML Structured Outline preferences page*.

The Contextual Menu of the Outline View

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

Append Child

Allows you to select an element (from a drop-down list) that is allowed by the associated schema and inserts it as a child of the current element.

Insert Before

Allows you to select an element (from a drop-down list) that is allowed by the associated schema and inserts it immediately before the current element, as a sibling.

Insert After

Allows you to select an element (from a drop-down list) that is allowed by the associated schema and inserts it immediately after the current element, as a sibling.

Edit Attributes

Opens a dialog box that allows you to edit the attributes of the currently selected component.

Toggle Comment

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

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Delete

Deletes the currently selected component.

Expand All

Expands the structure of a component in the **Outline** view.

Collapse All

Collapses the structure of all the component in the **Outline** view.

Document Tag Selection

The Outline view can also be used to search for a specific tag 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 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 the focus in the editor.

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

The Attributes View

The **Attributes** view presents all the attributes of the current element determined by the schema of the document.

You can use the **Attributes** view to insert attributes, edit their values, or add values to existing attributes.

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. If 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.

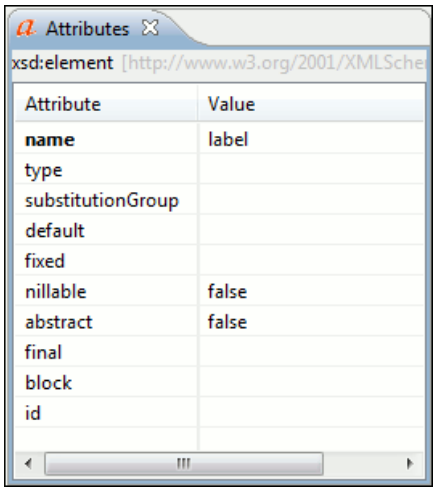


Figure 9: The Attributes View

Contextual Menu Actions in the Attributes View

The following actions are available in the contextual menu of the **Attributes** view when editing in **Text** mode:

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, if the attribute does not appear in the edited document.
- Empty, if the attribute does not appear in the edited document and has no default value set.

Paste

Depending on the content of the clipboard, the following cases are possible:

- If 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.
- If 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.
- If the clipboard only contains text, the value of the selected attribute is modified.

The Model View

The **Model** view presents the structure of the currently selected tag, and its documentation, defined as annotation in the schema of the current document. To open the **Model** view, select it from the **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** menu.

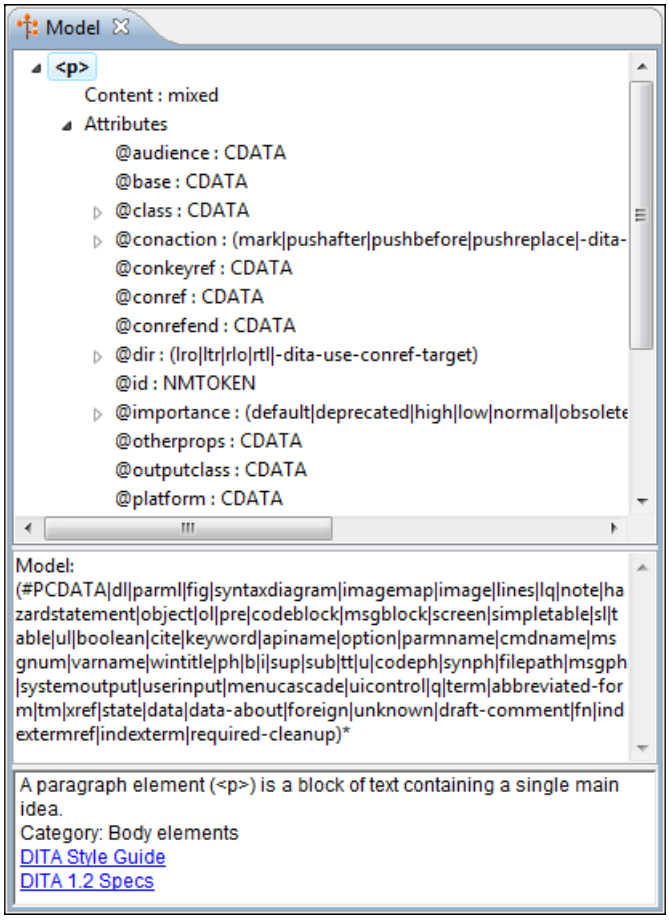


Figure 10: The Model View

The **Model** view is comprised of two sections, an element structure panel and an annotations panel.

Element Structure Panel

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

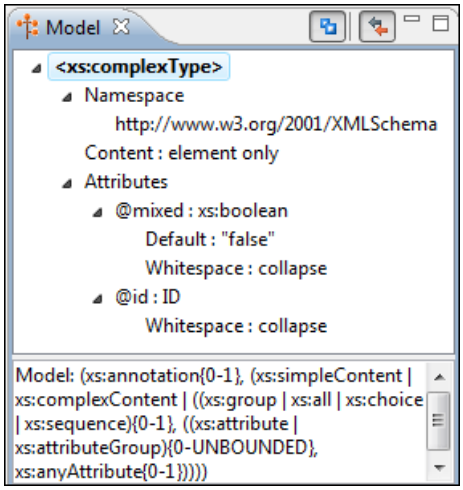


Figure 11: The Element Structure Panel

Annotation Panel

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

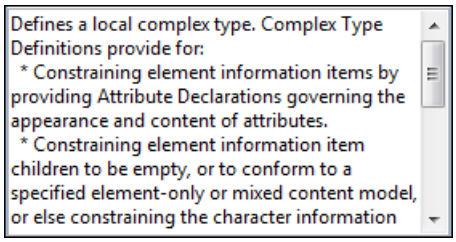


Figure 12: The Annotation panel

The Elements View

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

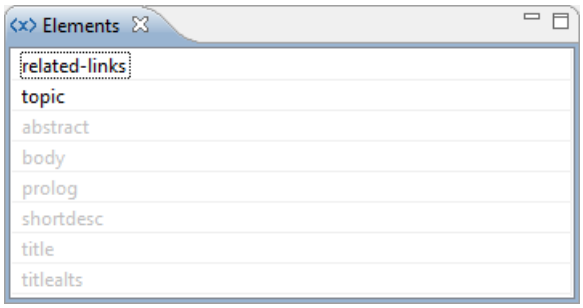


Figure 13: The Elements View

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 by clicking the column headers.

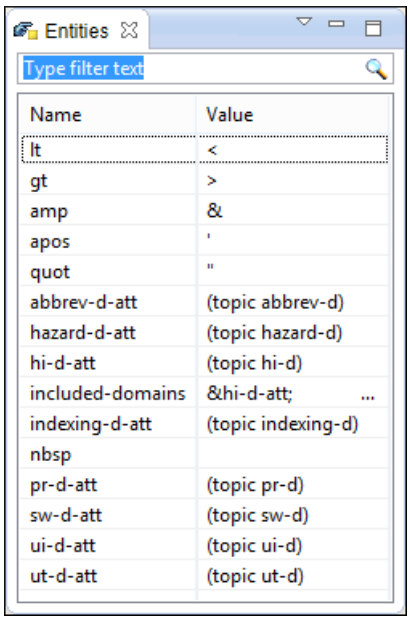


Figure 14: The Entities View

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

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

The Results View

The **Results View** displays the messages generated as a result of user actions such as 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 include the following:

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

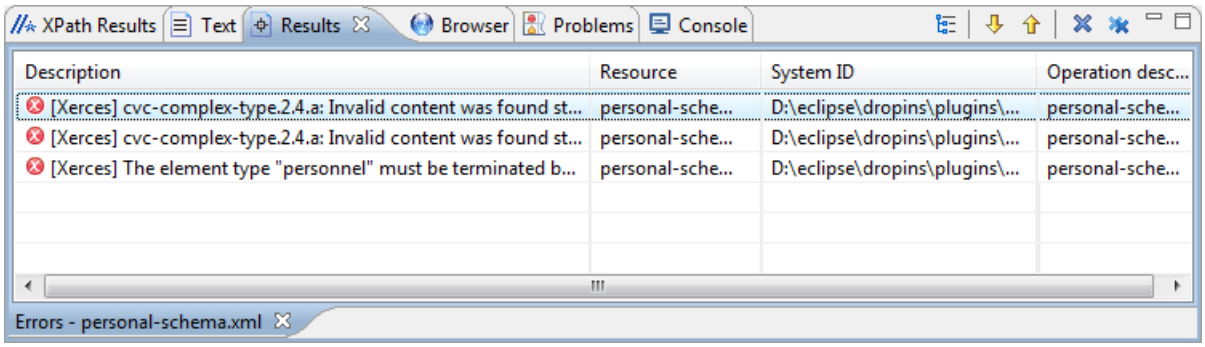


Figure 15: Results View

Results View Toolbar Actions

The view includes a toolbar with the following actions:

Grouping Mode toggle options

You can choose to group the result messages in a **Hierarchical** or **Flat** arrangement.

Next

Navigates to the message below the current selection.

Previous

Navigates to the message above the current selection.

Remove selected

Removes the current selection from the view. This can be helpful if you want to reduce the number of messages or remove those that have already been addressed or not relevant to your task.

Remove all

Removes all messages from the view.

Results View Contextual Menu Actions

The following actions are available when the contextual menu is invoked in the table grid:

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.

Remove

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.
- Error severity (error, warning, info message, etc.)
- Name of validating processor.
- The line and column in the file that triggered the message.

Save Results

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

Save Results as XML

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

Expand All

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

Collapse All

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

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 that 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

(such as XSLT, XHTML, XSL:FO, or XForms) are inserted in the same document and the editor panel can become cluttered. *Marking tags with different colors based on the namespace prefix* allows easier identification of the tags.

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

Figure 16: Example of Coloring XML Tags by Prefix

Presenting Validation Errors in Text Mode

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, but the color will be yellow instead of red. Hovering over a validation error presents a tooltip message with more details about the error and *possible quick fixes* (if available for that error or warning).

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 you to locate them more easily. The ruler contains the following areas:

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

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

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

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

Grid Editing Mode

The **Grid** editing mode in Oxygen XML Editor plugin displays XML documents in a structured spreadsheet-like grid. It allows you to easily edit XML documents consisting of repetitive patterns.

Grid Mode Editor

The Oxygen XML Editor plugin **Grid** editor displays the XML document as a structured grid of nested tables. To activate this mode, select **Grid** at the bottom of the editing area.

If 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 +[plus sign] (Command +[plus sign] on OS X)**, **Ctrl -[minus sign] (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)**.

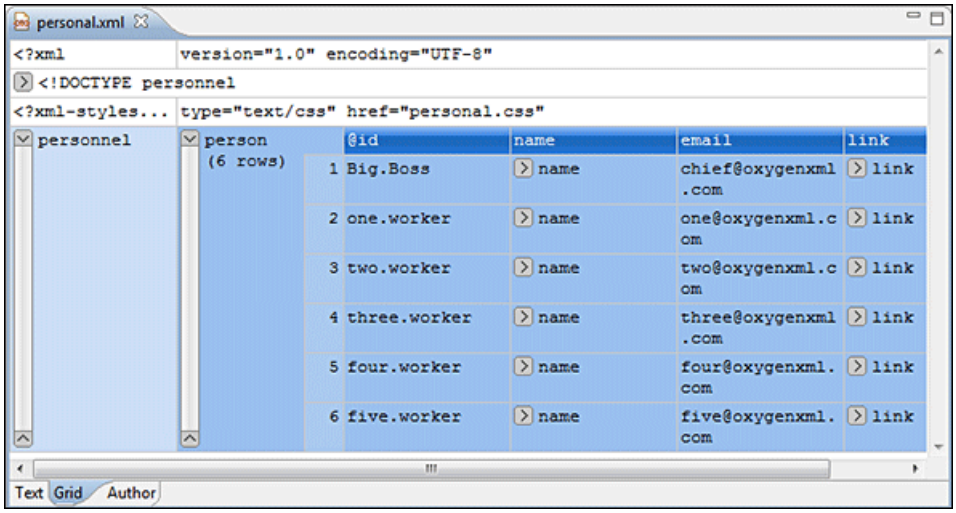


Figure 17: The Grid Editor

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

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

To display the content completion pop-up menu, 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 menu.

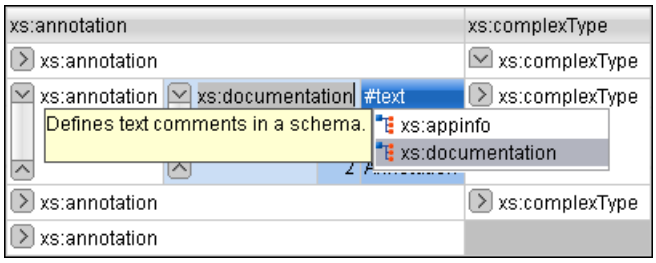


Figure 18: Content Completion in Grid Editor

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.

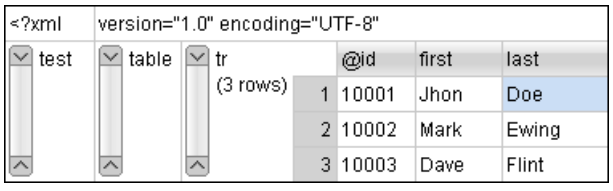


Figure 19: Grid Layout

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

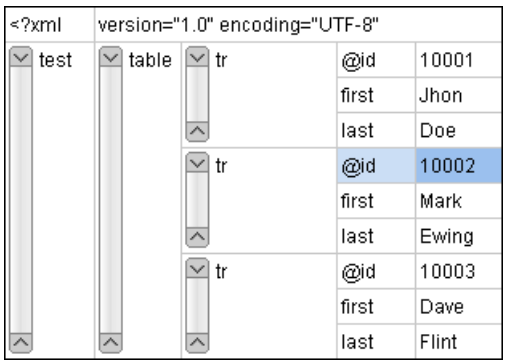


Figure 20: Tree Layout

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

Grid Mode Navigation

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

Table 1: Shortcuts in the Grid Mode

Key	Action
Tab	Moves the cursor to the next editable value in a table row.
Shift Tab	Moves the cursor 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+** and **NumPad-** keys, or from the **Expand/Collapse** submenu of the contextual menu.

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

Expand All

Expands the selection and all its children.

Collapse All

Collapses the selection and all its children.

Expand Children

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

Collapse Children

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

Collapse Others

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

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 **Change Text Orientation**(**Ctrl Shift O (Meta Shift O on OS X)**) action that is available from the **Edit** menu in the **Grid** editing mode. Use this action to switch from the default left to right text orientation to the right to left orientation, and vice versa.

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

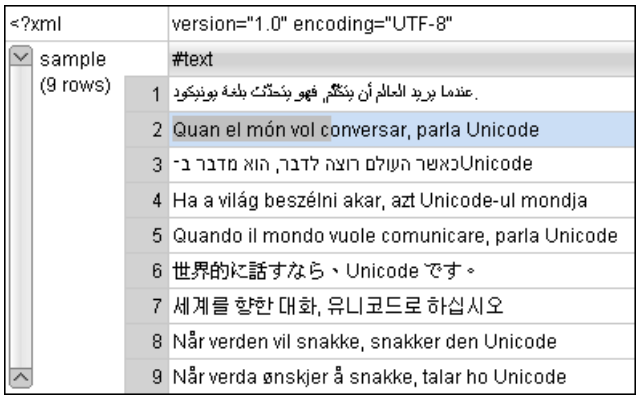


Figure 21: Default left to right text orientation

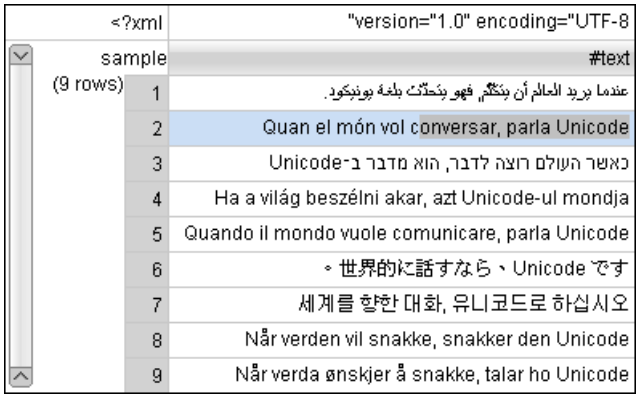


Figure 22: Right to left text orientation

Author Editing Mode

This chapter presents the WYSIWYG-like visual editor, called **Author** mode, that is targeted to content authors.

Author Mode Editor

The **Author** editing mode in Oxygen XML Editor plugin allows you to visually edit XML documents in a visual interface that is similar to a WYSIWYG word processor.

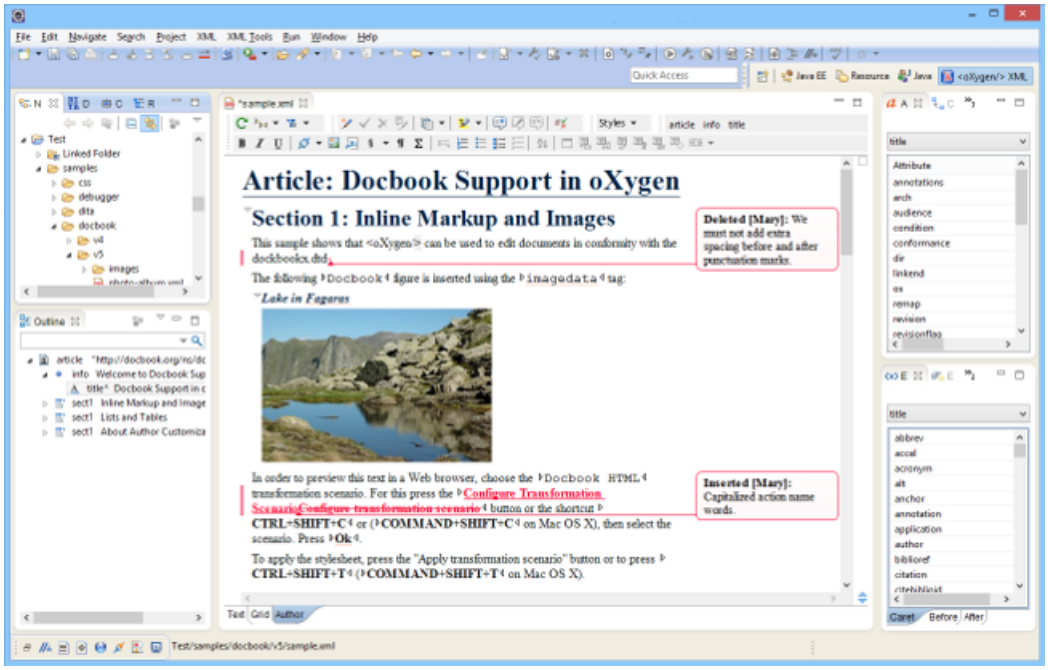


Figure 23: Author Editing Mode

Navigating the Document Content in Author Mode

Using the Keyboard

Oxygen XML Editor plugin allows you to quickly navigate through a document using the **Tab** key to move the cursor to the next XML node and **Shift Tab** to go to the previous one. If you encounter a space preserved element when you navigate through a document and you do not press another key, pressing the **Tab** key will continue the navigation. However, if the cursor is positioned in a space preserved element and you press another key or you position the cursor inside such an element using the mouse, the **Tab** key can be used to arrange the text.

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 Folding Support

When working with a large document, the **folding support** can be used to collapse some element content leaving only those that you need to edit in focus. Foldable elements are marked with a small triangle painted in the upper left corner ( ). If you hover over that arrow, 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 **Folding** sub-menu of the contextual menu:

Toggle Fold (or you can simply click on the / arrow)

Toggles the state of the current fold.

Collapse Other Folds (Ctrl NumPad / (Meta NumPad / on OS X))

Folds all the elements except the current element.

Collapse Child Folds (Ctrl + NumPad- (Meta + NumPad- on OS X))

Folds the child elements that are indented one level inside the current element.

Expand Child Folds (Ctrl NumPad+ (Meta NumPad+ on OS X))

Unfolds all child elements of the currently selected element.

Expand All (Ctrl NumPad * (Meta NumPad * on OS X))

Unfolds all elements in the current document.

Using the Linking Support

When working on multiple documents that reference each other (references, external entities, XInclude, DITA conref, etc), the **linking support** is useful for navigating between the documents. In the predefined customizations that are bundled with Oxygen XML Editor plugin, links are marked with an icon representing a chain link (). When hovering over the icon, the mouse pointer changes its shape to indicate that the link can be accessed and a tooltip presents the destination location. Click the link to open the referenced resource in the editor or system browser. The same effect can be obtained by pressing the **F3** key when the cursor is in a link element.

Note: Depending on the referenced file type, the target link will either be opened in the Oxygen XML Editor plugin or in the default system application. If the target file does not exist, Oxygen XML Editor plugin 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 **Tags display mode** drop-down menu that is available on the toolbar:

Full Tags with Attributes

Displays full tag names with attributes for both block level and in-line level elements.

Full Tags

Displays full tag names without attributes for both block level and in-line level elements.

Block Tags

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

Inline Tags

Displays full tag names 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 and is as close as possible to a word-processor view.

To set a default mode of the tags mode, go to *Author preferences page* and configure the **Tags display mode** options. However, if the document opened in **Author** mode 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 (for example, paragraphs), while the inline level elements are distributed in lines (for example, 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.

Visual Hints for the Cursor Position

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

Here are the common situations that can be encountered:

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

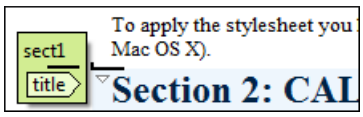


Figure 24: Before first block

- The cursor is positioned between two block elements.

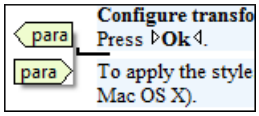


Figure 25: Between two block elements

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

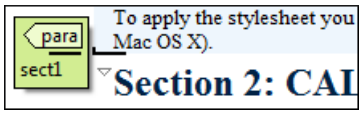


Figure 26: After last block

- The cursor is positioned inside a node.

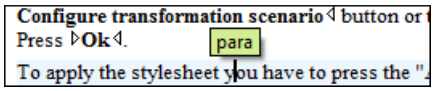


Figure 27: Inside a node

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

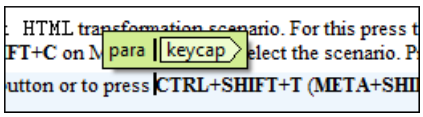


Figure 28: Before an inline element

- The cursor is positioned between two inline elements.

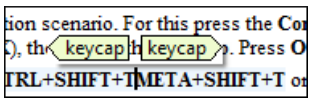


Figure 29: Between two inline elements

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

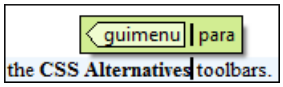


Figure 30: After an inline element

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

To deactivate this feature, *open the **Preferences** dialog box* and go to **Editor / Author** > **Show cursor 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 cursor between certain tags that do not have a visual representation. To counterbalance this, Oxygen XML Editor plugin displays a transparent preview of the Position Information Tooltip, called *Location Tooltip*:

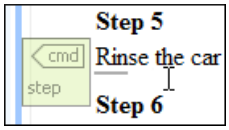


Figure 31: Location Tooltip

Oxygen XML Editor plugin 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 ***Cursor Navigation** preferences page*.

Displaying Referenced Content

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

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

The referenced content is refreshed:

- Automatically, when it is modified and saved from Oxygen XML Editor plugin.
- On demand, by using the *Refresh references action*. Useful when the referenced content is modified outside the Oxygen XML Editor plugin scope.

Presenting Validation Errors in Author Mode

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

Figure 32: Presenting Validation Errors in Author Mode

A fragment with a validation error is marked by underlining the error in red, and validation warnings are underlined in yellow.

Also, the ruler on the right side of the editor panel is designed to display the errors found during the validation process and to help you locate them in the document. The ruler contains the following:

- The top area - A success indicator square will turn green if the validation is successful, red if validation errors are found, or yellow if validation warnings are found.More details about the errors or warnings are displayed in a tool tip when you hover over indicator square. If there are numerous errors, only the first three are presented in the tool tip.
- The middle area - Errors are depicted with red markers, and warnings with yellow markers. If you want to limit the number of markers that are displayed, *open the **Preferences** dialog box* and go to **Editor** > **Document checking** > **Maximum number of validation highlights**.

  Clicking a marker will highlight the corresponding text area in the editor. The error or warning message is also displayed both in a tool tip (when hovering over the marker) and in the message area on the bottom of the editor panel.

  The validation status area at the bottom of the editor panel presents the message of the current validation error. Clicking the **Document checking options** button opens the ***Document checking user preferences*** page
- The bottom area - Two navigation arrows ( ) allow you to skip to the next or previous error. The same actions can be triggered from **Document** > **Automatic validation** > **Next error** (**Ctrl Period (Meta Period on OS X)**) and **Document** > **Automatic validation** > **Previous error** (**Ctrl Comma (Meta Comma on OS X)**).

Status messages from every validation action are logged in the ***Console** view* (the **Enable oXygen consoles** option must be enabled in **Preferences** > **View**).

Whitespace Handling in Author Mode

When you edit a document in **Author** mode, Oxygen XML Editor plugin must serialize the resulting document as XML. Oxygen XML Editor plugin serializes the document when you save it or switch to another editing mode. When the document is serialized, Oxygen XML Editor plugin *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 Editor plugin 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 Editor plugin 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 Editor plugin 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 Editor plugin 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 Editor plugin recognizes that the current element is a preserve-space element.

Author Mode Views

The content author is supported by special views that 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.

There is a large selection of useful views available in the **Window** > **Show View** menu. This section presents some of the most helpful views for editing in **Author** mode.

The Navigator View

The **Navigator** view is designed to assist you with organizing and managing related files grouped in the same XML project. The actions available on the contextual menu and toolbar associated to this panel enable the creation of XML projects and shortcuts to various operations on the project documents.

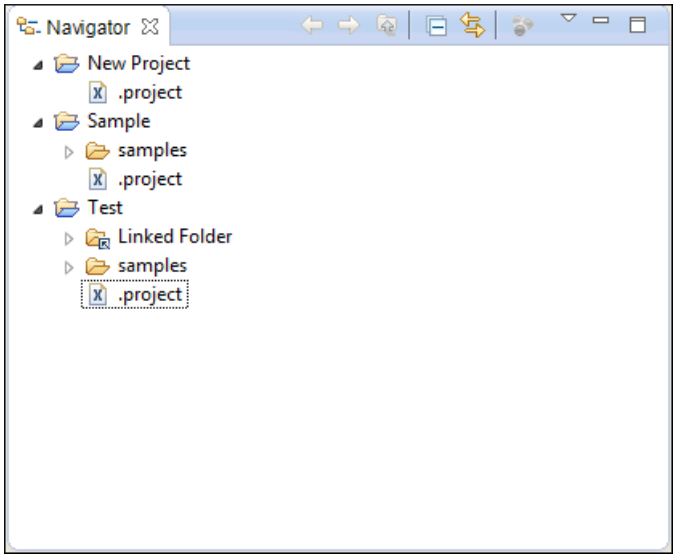


Figure 33: The Navigator View

The following actions are grouped in the upper right corner:

Collapse All

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

Link with Editor

When selected, the project tree highlights the currently edited file, if it is found in the project files.

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

View Menu

Drop-down menu that contains various settings.

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

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

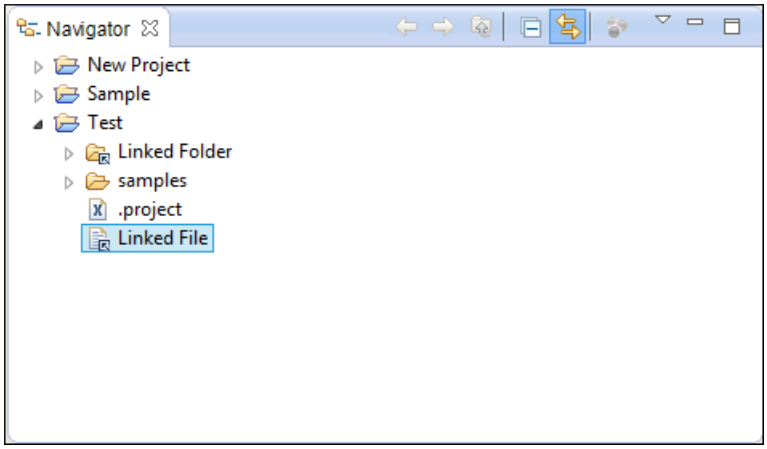


Figure 34: The Navigator View with Examples of the Two Types of Resources

Creating New Projects

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

New > XML Project

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

New > Sample XML Project

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

Creating New Project Items

To create new project items, select the desired document type or folder from the **New** menu of the contextual menu, when invoked from the **Navigator** view (or from the **File** menu). You can also create a document from a template by selecting **New** > **New from Templates** from the contextual menu.

New > **File**

Opens a **New** file dialog box that helps you create a new file and adds it to the project structure.

New > **Folder**

Opens a **New Folder** dialog box that allows you to specify a name for a new folder and adds it to the structure of the project.

New > **Logical Folder**

Available when invoked from the *project root*, this action creates a logical folder in the tree structure (the icon is a magenta folder on Mac OS X -).

New > Logical Folders from Web

Available when invoked from the *project root*, this action 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.

Managing Project Content

Creating/Adding Files and Folders

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

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

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

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

Removing Files and Folders

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

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

Moving Files and Folders

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

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

Renaming Files and Folders

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

- Invoke the **Rename** action from the contextual menu.
- Press **F2** and type the new name.

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

Locating and Opening Files

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

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

Saving the Project

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

Validate Files

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

Validate > **Check Well-Formedness**

Checks if the selected file or files are well-formed.

Validate **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.

ValidateValidate with Schema

Validates the selected file of files against a specified schema.

Validate **Configure Validation Scenario(s)**

Allows you to configure and run a *validation scenario*.

Validate > **Clear Validation Markers**

Clears all the error markers from the main editor and **Problems** view.

Applying Transformation Scenarios

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

Transform > **Apply Transformation Scenario(s)**

Obtains the output with one of the built-in scenarios.

Transform > **Configure Transformation Scenario(s)**

Opens a dialog box that allows you to configure pre-defined transformation scenarios.

Transform > **Transform with**

Allows you to select a transformation scenario to be applied to the currently selected files.

Refactoring Actions (Available for certain document types (such as XML, XSD, and XSL)

Oxygen XML Editor plugin includes some refactoring operations that help you manage the structure of your documents. The following actions are available from the contextual menu in the **Refactoring** sub-menu:

Refactoring > Rename resource

Allows you to change the name of a resource.

Refactoring > Move resource

Allows you to change the location on disk of a resource.

Refactoring > XML Refactoring

Opens *the XML Refactoring tool wizard* that presents refactoring operations to assist you with managing the structure of your XML documents.

Other Contextual Menu Actions

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

Open

Opens the selected file in the editor.

Open with submenu

This submenu offers you choices for opening the selected file in various editors.

Refresh

Refreshes the content and the dependencies between the resources in *the **Master Files** directory*.

XPath in Files

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

Check Spelling in Files

Allows you to *check the spelling of multiple files.*

Format and Indent Files

Opens the ***Format and Indent Files** dialog box* that allows you to configure the format and indent (pretty print) action that will be applied on the selected documents.

Properties

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

Outline View

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

By default it is displayed on screen, but if you closed it you can reopen it from **Window** > **Show View** > **Outline**.

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

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

- *XML Document Overview* on page 64
- *Modification Follow-up* on page 64
- *Drag and Drop Actions in Outline View* on page 64
- *Outline Filters* on page 87
- *The Contextual Menu of the Outline Tree* on page 87
- *Document Tag Selection* on page 66

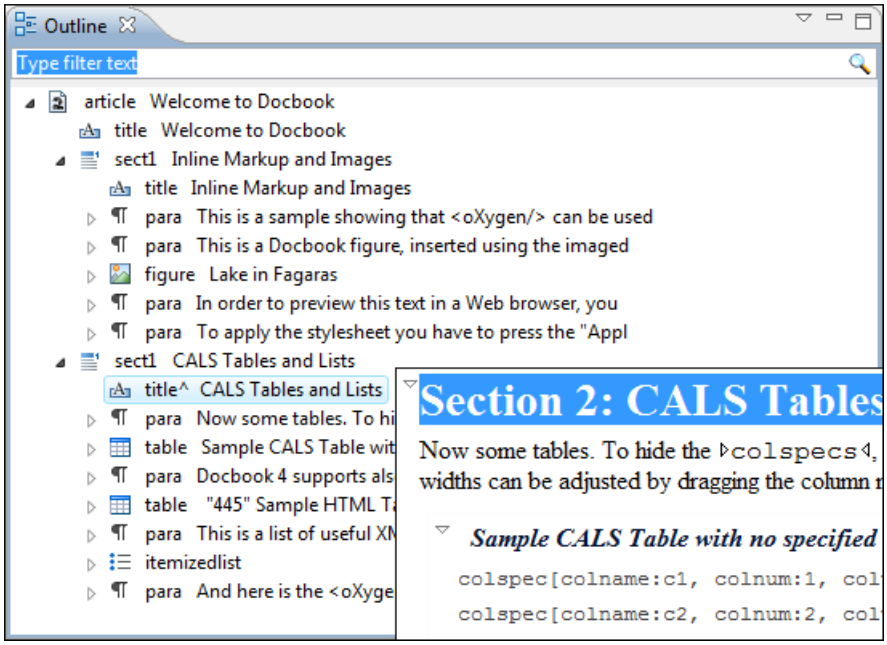


Figure 35: The Outline View

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 you to be aware of the document structure and the way tags are nested.

The **Outline** view allows you to:

- Insert or delete nodes using contextual 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 Single-Click (Command Single-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.

Drag and Drop Actions in Outline View

Entire XML elements can be moved or copied in the edited document using only the mouse in the **Outline** view with 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 a Preferences page.*

Tip: You can select and drag multiple nodes in the **Outline** view when editing in **Author** mode.

Outline Filters

The upper part of the **Outline** view contains a filter box that 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 following actions are available in the **View menu** of the **Outline** view when editing in **Author** mode:

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 Contextual Menu of the Outline Tree

The contextual menu of the **Outline** view in **Author** mode contains the following actions:

Edit Attributes

Allows you to edit the attributes of a selected node. You can find more details about this action in the *Attributes View in Author Mode* on page 88 topic.

Edit Profiling Attributes

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

Append Child

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

Insert Before

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

Insert After

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

Cut, Copy, Paste, Delete

Executes the typical editing actions on the currently selected elements. The **Cut** and **Copy** operations preserve the styles of the copied content. The **Paste before**, **Paste after**, and **Paste as Child** actions allow you to insert a well-formed element before, after, or as a child of the currently selected element.

Toggle Comment

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

Rename Element

Invokes a **Rename** dialog box that allows you to rename the currently selected element, siblings with the same name, or all elements with the same name.

Expand All

Expands the structure tree of the currently selected element.

Collapse All

Collapses all of the structure tree of the currently selected node.

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

Document Tag Selection

The Outline view can also be used to search for a specific tag 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 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 the focus in the editor.

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

Attributes View in Author Mode

The **Attributes** view presents all the attributes of the current element determined by the schema of the document.

You can use this view to edit or add attribute values. The attributes of an element are editable if any 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 in a deleted *Track Changes* marker.
- The element is part of a content fragment that is referenced in **Author** mode from another document.

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. If 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.

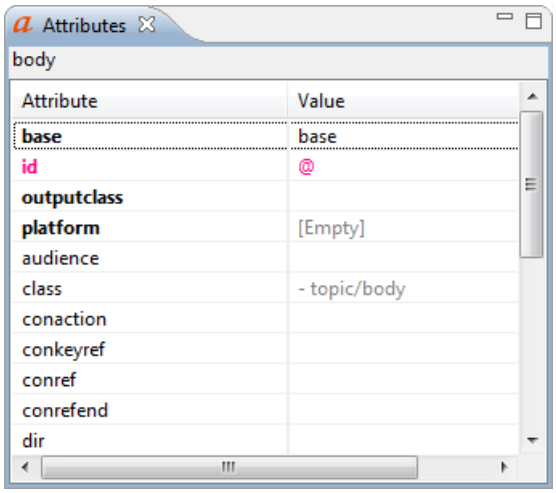


Figure 36: The Attributes View

A drop-down list located in the upper part of the view allows you to select the current element or its ancestors.

Contextual Menu Actions in the Attributes View

The following actions are available in the contextual menu of the **Attributes** view when editing in **Author** mode:

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, if the attribute does not appear in the edited document.
- Empty, if the attribute does not appear in the edited document and has no default value set.

Paste

Depending on the content of the clipboard, the following cases are possible:

- If 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.
- If 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.
- If the clipboard only contains text, the value of the selected attribute is modified.

In-place Attributes Editor

Oxygen XML Editor plugin includes an in-place attributes editor in **Author** mode. To edit the attributes of an XML element in-place, do one of the following:

- Select an element or place the cursor 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**, or **Inline Tags**.

This opens an in-place attributes editor that contains the same content as the **Attributes** view. By default, this editor presents the **Name** and **Value** fields, with the list of all the possible attributes collapsed.

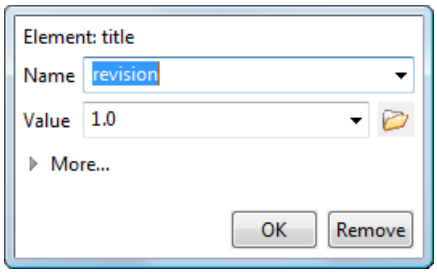


Figure 37: In-place Attributes Editor

Name Combo Box

Use this combo box to select an attribute. The drop-down list displays the list of possible attributes allowed by the schema of the document, as in the **Attributes** view.

Value Combo Box

Use this combo box to add, edit, or select the value of an attribute. If the selected attribute has predefined values in the schema, the drop-down list displays those possible values.

If you click **More** while in the collapsed version, it is expanded to the full version of the in-place attribute editor.

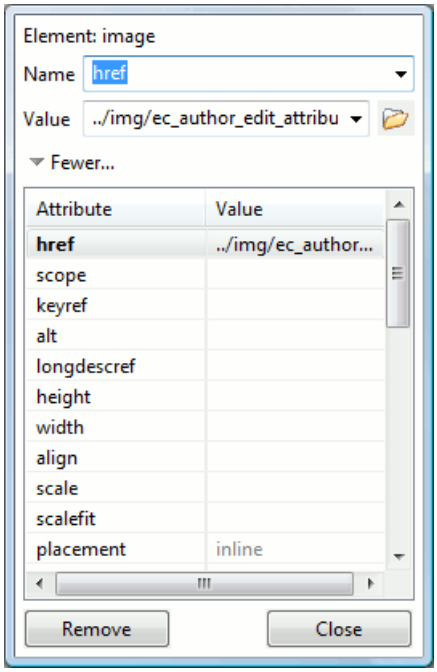


Figure 38: In-place Attributes Editor (Full Version)

The full version includes a table grid, similar to the **Atributes** view, that presents all the attributes for the selected element.

The Model View

The **Model** view presents the structure of the currently selected tag, and its documentation, defined as annotation in the schema of the current document. To open the **Model** view, select it from the **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** menu.

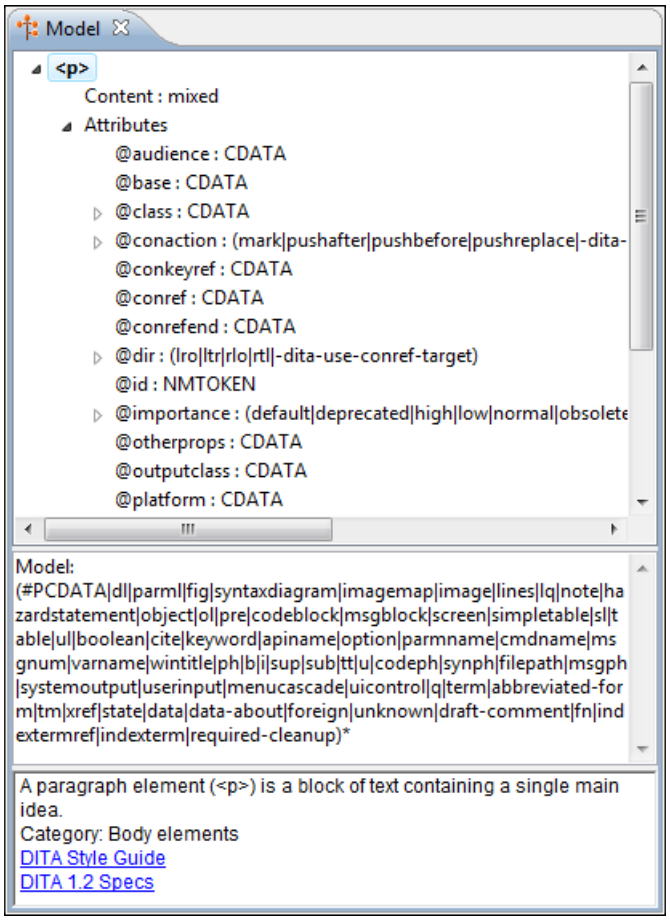


Figure 39: The Model View

The **Model** view is comprised of two sections, an element structure panel and an annotations panel.

Element Structure Panel

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

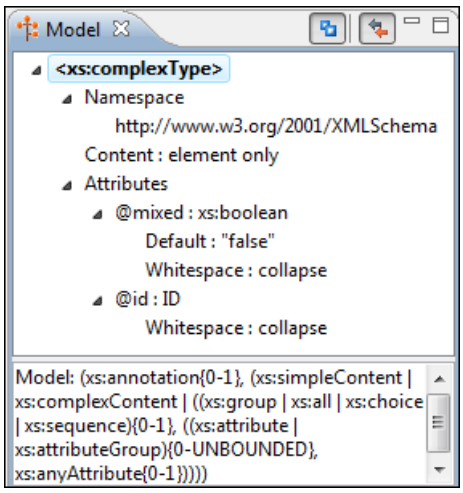


Figure 40: The Element Structure Panel

Annotation Panel

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

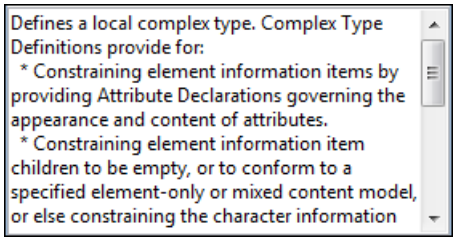


Figure 41: The Annotation panel

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 disabled and rendered in gray. 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.

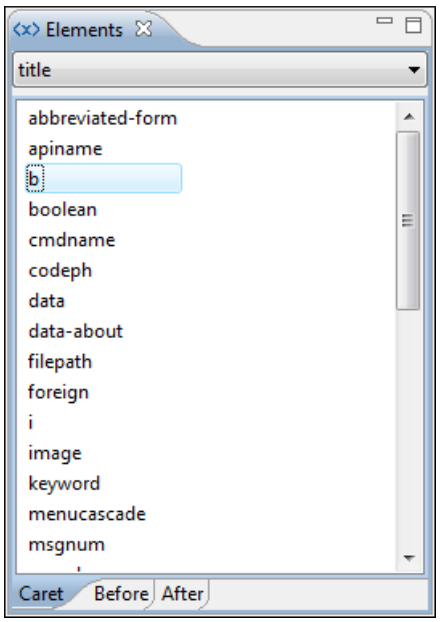


Figure 42: The Elements View

Three tabs present information relative to the cursor location:

- **Cursor** - Displays a list of all the elements allowed at the current cursor location. Double-clicking any of the listed elements inserts that element at the cursor position.
- **Before** - Displays 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 cursor position.
- **After** - Displays 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 cursor 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.

The Entities View

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

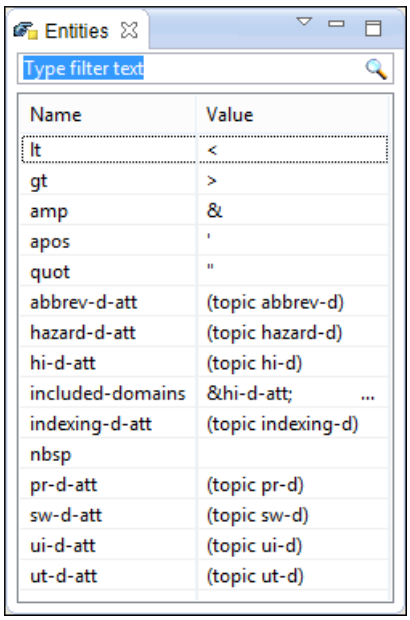


Figure 43: The Entities View

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

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

The Results View

The **Results View** displays the messages generated as a result of user actions such as 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 include the following:

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

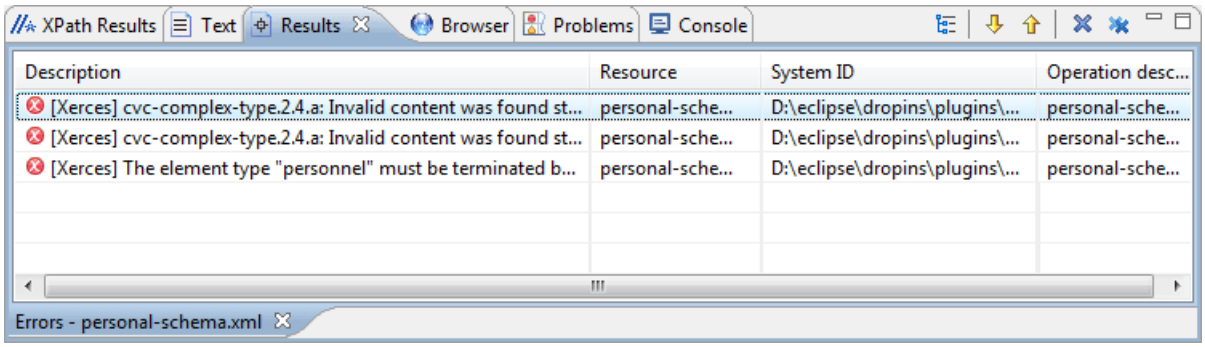


Figure 44: Results View

Results View Toolbar Actions

The view includes a toolbar with the following actions:

Grouping Mode toggle options

You can choose to group the result messages in a **Hierarchical** or **Flat** arrangement.

Next

Navigates to the message below the current selection.

Previous

Navigates to the message above the current selection.

Remove selected

Removes the current selection from the view. This can be helpful if you want to reduce the number of messages or remove those that have already been addressed or not relevant to your task.

Remove all

Removes all messages from the view.

Results View Contextual Menu Actions

The following actions are available when the contextual menu is invoked in the table grid:

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.

Remove

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.
- Error severity (error, warning, info message, etc.)
- Name of validating processor.
- The line and column in the file that triggered the message.

Save Results

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

Save Results as XML

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

Expand All

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

Collapse All

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

CSS Inspector View

The purpose of the **CSS Inspector** view is to display information about the styles applied to the currently selected element. You can use this view to examine the structure and layout of the CSS rules that match the element. The matching rules displayed in this view include a link to the line in the CSS file that defines the styles. With this tool you can see

how the CSS rules were applied and the properties defined, and use the link to open the associated CSS for editing purposes.

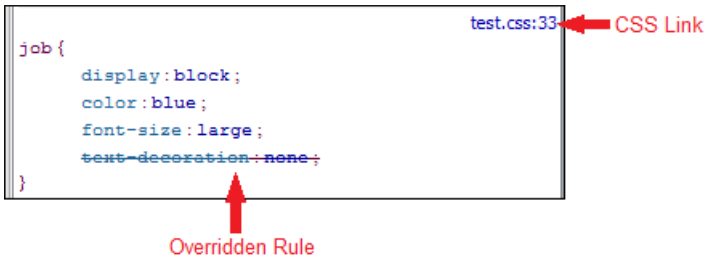
Figure 45: CSS Inspector View

Displaying the CSS Inspector View

You can open this view by selecting the **Inspect Styles** action from the contextual menu in **Author** mode, or selecting the **CSS Inspector** view in the **Window** > **Show View** menu. This action makes the view visible and also initializes it for the currently selected element.

Displaying Rules

All rules that apply to the current element are displayed in sections, which are listed in order of importance (from most specific to least specific). Rules that are overridden by other rules are crossed out. If you click the link in the top-right corner of a rule Oxygen XML Editor plugin opens the associated CSS file at the line number where the properties of the rule are defined.



The **CSS Inspector** view contains five tabs:

- **Element** - displays the CSS rules matching the currently selected element in the **Author** page (ordered from most-specific to least-specific)
- **:before** - displays the rules matching the `:before` pseudo-element
- **:after** - displays the rules matching the `:after` pseudo-element

- **Computed** - displays all the styling properties that apply to the current element, as a result of all the CSS rules matching the element
- **Path** - displays the path for the current element, and its attributes, allowing you to quickly see the attributes on all parent elements, and allows you to copy fragments from this view and paste it into the associated CSS to easily create new rules

The information displayed in each of the five tabs is updated when you click different elements in the **Author** editing view. The first three tabs include the link to the associated CSS source, while the other two tabs simply display the style properties that match the current element.

Each of the tabbed panes include a contextual menu with the following actions:

- **Copy** - copies the current selection
- **Select all** - selects all information listed in the pane

Also, a **Show empty rules** action is available from a drop-down menu in the toolbar of the view. This action forces the view to show all the matching rules, even if they do not declare any CSS properties. By default, the empty rules are not displayed.

Bidirectional Text Support in Author Mode

Oxygen XML Editor plugin 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 Editor plugin 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 Editor plugin Supports the following CSS properties:

Table 2: 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 Editor plugin 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 3: 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 **Symbols** toolbar action. 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 Editor plugin offers the support for bi-directional text in all the side views (**Outline** view, **Attributes** view and so on) and text fields.

Design Editing Mode

This section presents the **Design** mode that allows you to edit XML Schemas in a visual schema diagram editor.

XML Schema Diagram Editor (Design Mode)

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

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

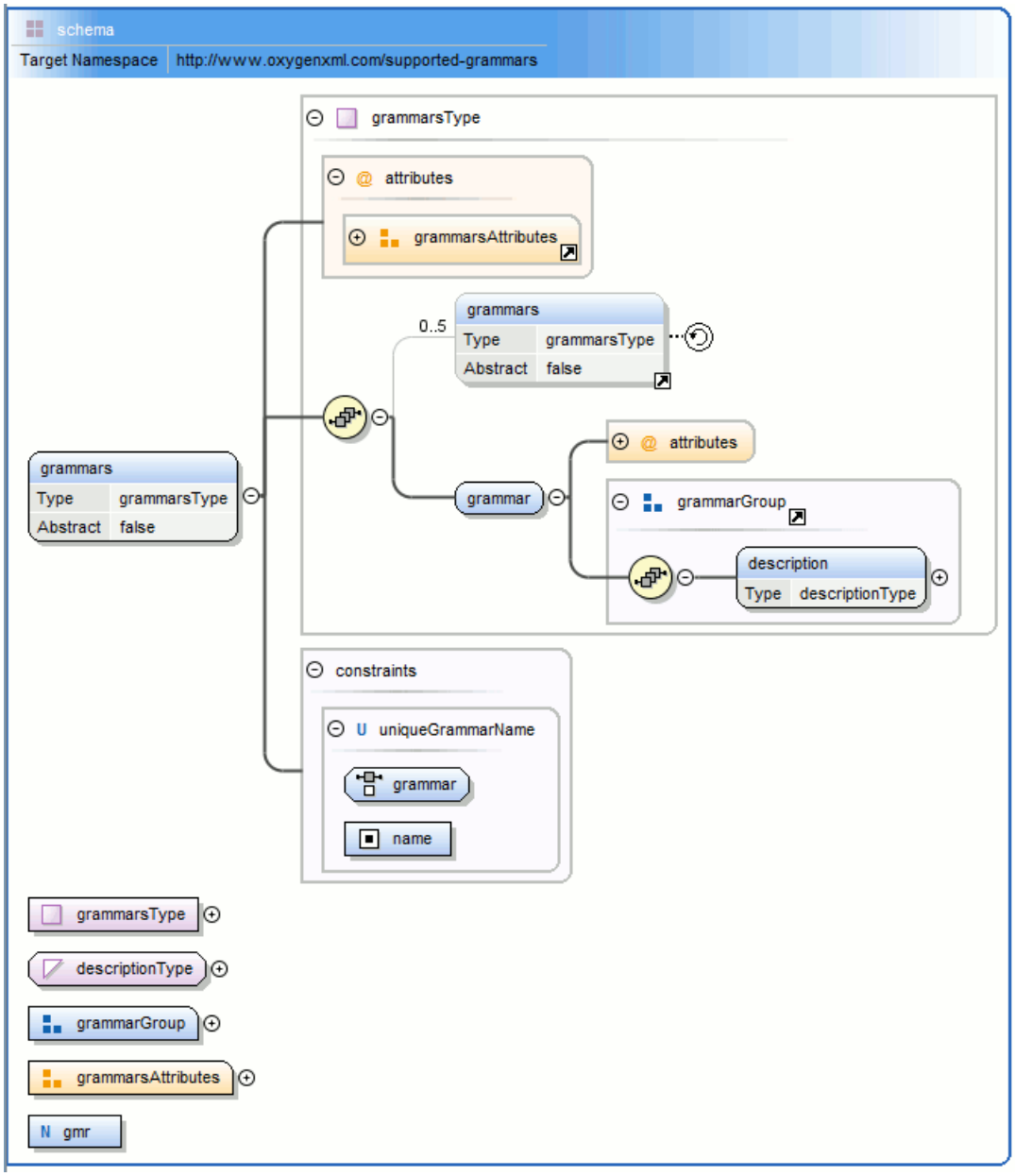


Figure 46: XML Schema Diagram

To watch our video demonstration about the basic aspects of designing an XML Schema using the new Schema Editor, go to *http://oxygenxml.com/demo/XML_Schema_Editing.html*.

Navigation in the Schema Diagram

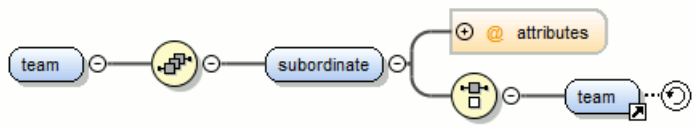
The following editing and navigation features work for all types of schema components:

- Move/reference components in the diagram using drag-and-drop actions.
- Select consecutive components on the diagram (components from the same level) using the *Shift* key. You can also make discontinuous selections in the schema diagram using the **Ctrl (Meta on Mac OS)** key. To deselect one of the components, use **Ctrl Single-Click (Command Single-Click on OS X)**.
- Use the arrow keys to navigate the diagram vertically and horizontally.
- Use *Home*/*End* keys to jumpt to the first/last component from the same level. Use **Ctrl Home (Command Home on OS X)** key combination to go to the diagram root and **Ctrl End (Command End on OS X)** to go to the last child of the selected component.

- You can easily go back to a previously visited component while moving from left to right. The path will be preserved only if you use the left arrow key or right arrow key. For example, if the current selection is on the second attribute from an attribute group and you press the left arrow key to jump to the attribute group, when you press the right arrow key, then the selection will be moved to the second attribute.
- Go back and forward between components viewed or edited in the diagram by selecting them in the **Outline** view:
  - **Back** (go to previous schema component).
  - **Forward** (go to next schema component).
  - **Go to Last Modification** (go to last modified schema component).
- Copy, reference, or move global components, attributes, and identity constraints to a different position and from one schema to another using the **Cut/Copy** and **Paste/Paste as Reference** actions.
- Go to the definition of an element or attribute with the **Show Definition** action.
- You can expand and see the contents of the imports/includes/redefines in the diagram. In order to edit components from other schemas the schema for each component will be opened as a separate file in Oxygen XML Editor plugin.

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

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



XML Schema Outline View

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

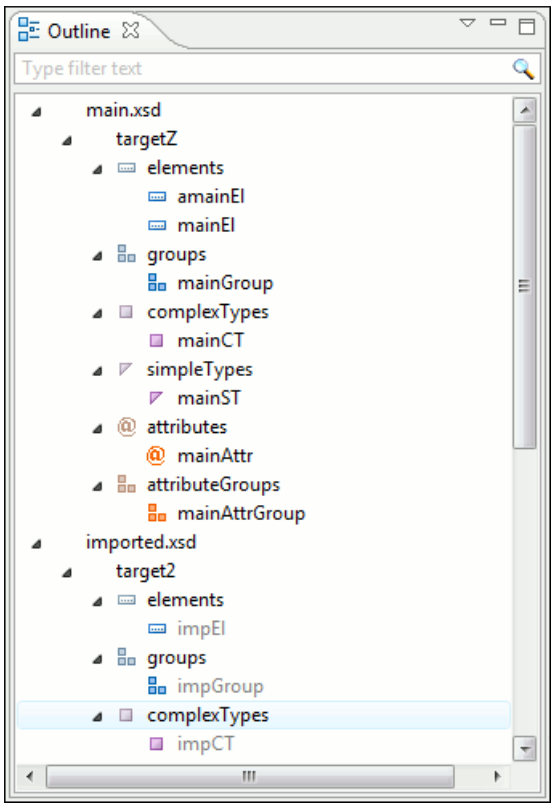


Figure 47: The Outline View for XML Schema

The **Outline** view provides the following options in the **View Menu** on the **Outline** view action bar:

Filter returns exact matches

The text filter of the **Outline** view returns only exact matches;

Selection update on cursor move

Allows a synchronization between **Outline** view and schema diagram. The selected view from the diagram is also selected in the **Outline** view.

Sort

Allows you to sort alphabetically the schema components.

Show all components

Displays all components that were collected starting from the main files. Components that are not referable from the current file are marked with an orange underline. To reference them, add an import directive with the `componentNS` namespace.

Show referable components

Displays all components (collected starting from the main files) that can be referenced from the current file. This option is set by default.

Show only local components

Displays the components defined in the current file only.

Group by location/namespace/type

These three operations allow you to group the components by location, namespace, or type. When grouping by namespace, the main schema target namespace is the first presented in the **Outline** view.

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

Remove (Delete)

Removes the selected item from the diagram.

 **Search References**

Searches all references of the item found at current cursor position in the defined scope, if any.

Search References in

Searches all references of the item found at current cursor position in the specified scope.

 **Component Dependencies**

Allows you to see the dependencies for the current selected component.

Resource Hierarchy

Allows you to see the hierarchy for the current selected resource.

Resource Dependencies

Allows you to see the dependencies for the current selected resource.

 **Rename Component in**

Renames the selected component.

 **Generate Sample XML Files**

Generate XML files using the current opened schema. The selected component is the XML document root.

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

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

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

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

The content of the **Outline** view and the editing area are synchronized. When you select a component in the **Outline** view, its definition is highlighted in the editing area.

The Attributes View

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

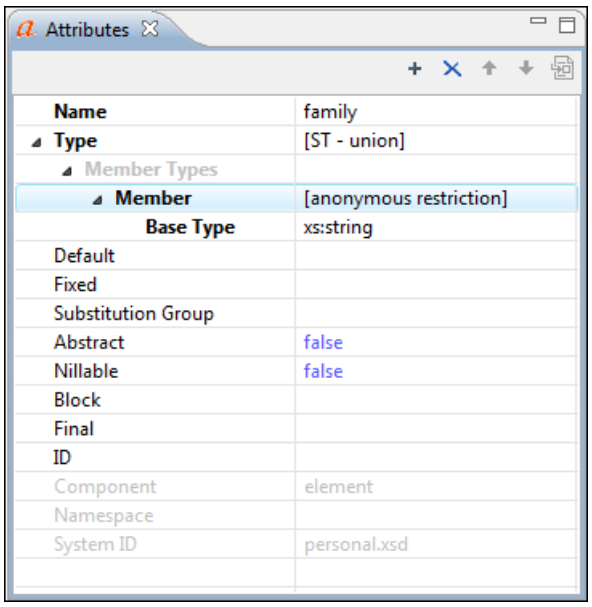


Figure 48: The Attributes View

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

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

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

For imports, includes and redefines, the properties are not edited directly in the **Attributes** view. A dialog box will open that allows you to specify properties for them.

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

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

Add

Allows you to add a new member type to an union's member types category.

Remove

Allows you to remove the value of a property.

Move Up

Allows you to move up the current member to an union's member types category.

Move Down

Allows you to move down the current member to an union's member types category.

Copy

Copy the attribute value.

Show DefinitionCtrl (Meta on MAC OS) + Click

Shows the definition for the selected type.

Show Facets

Allows you to edit the facets for a simple type.

The Facets View

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

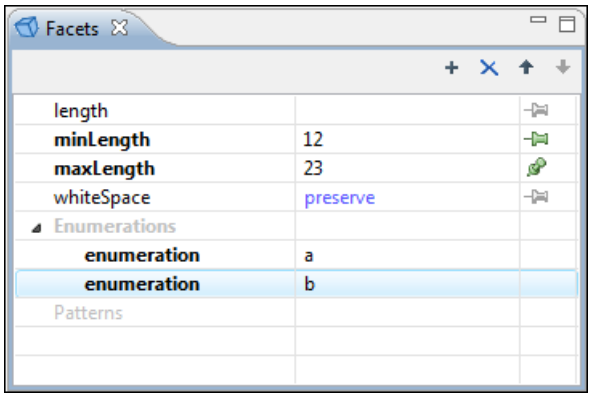


Figure 49: The Facets View

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

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

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

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

The **Facets** view provides the following actions in its toolbar and contextual menu:

Add

Allows you to add a new enumeration or a new pattern.

Remove

Allows you to remove the value of a facet.

Edit Annotations

Allows you to edit an annotation for the selected facet.

Move Up

Allows you to move up the current enumeration/pattern in **Enumerations/Patterns** category.

Move Down

Allows you to move down the current enumeration/pattern in **Enumerations/Patterns** category.

Copy

Copy the attribute value.

Open in Regular Expressions Builder

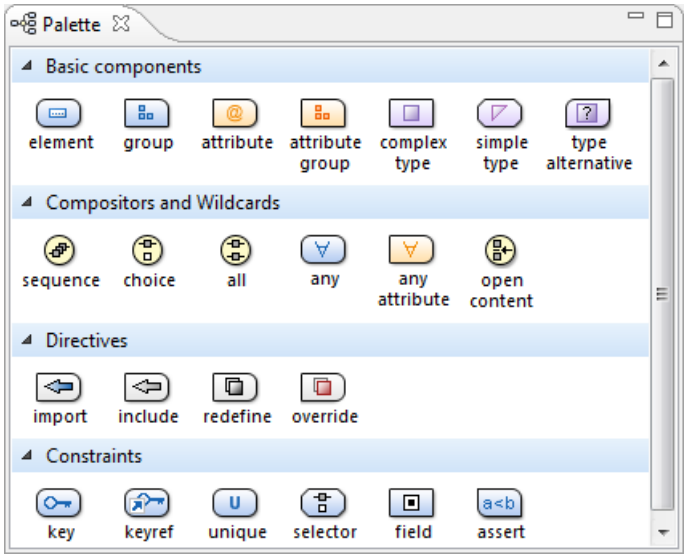
Rather than editing regular expressions manually, this action allows you to open the pattern in the *XML Schema Regular Expressions Builder* that guides you through the process of testing and constructing the pattern..

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

The Palette View

The **Palette** view is designed to offer quick access to XML Schema components and to improve the usability of the XML Schema diagram builder. You can use the **Palette** to drag and drop components in the **Design** mode. The components offered in the **Palette** view depend on the XML schema version set in the ***XML Schema*** *preferences page*.

Figure 50: Palette View



Components are organized functionally into 4 collapsible categories:

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

Note: The *type alternative*, *open content*, *override*, and *assert* components are available for XML Schema 1.1.

To add a component to the edited schema:

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

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

To watch our video demonstration about the Schema palette and developing XML Schemas, go to *http://oxygenxml.com/demo/Schema_Palette.html* and *http://oxygenxml.com/demo/Developing_XML_Schemas.html* respectively.

Chapter 6

Editing Documents

Topics:

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

Working with Unicode

Unicode provides a unique number for every character, independent of the platform and language. Unicode is an internationally recognized standard, adopted by industry leaders. The Unicode is required by modern standards (such as XML, Java, 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 multiple tiered applications and websites offers significant cost savings over the use of legacy character sets.

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

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

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

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

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

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

Inserting Symbols

You can insert symbols by using the **Character Map** dialog box that can be opened with the **Edit** > Ω **Insert from Character Map** action.

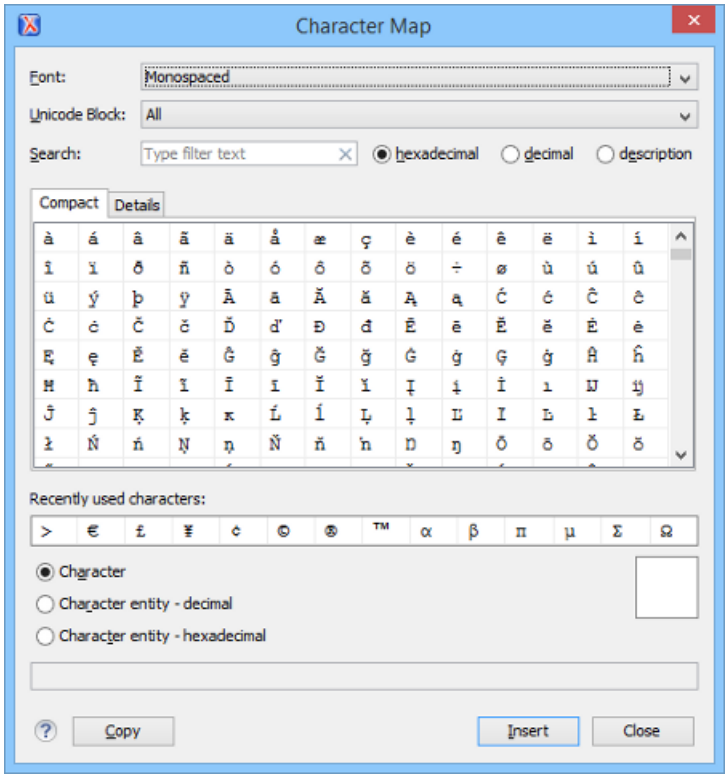


Figure 51: The Character Map Dialog Box

The **Character Map** dialog box allows you to visualize all characters that are available in a particular font, pick the character you need, and insert it in the document you are editing. It includes the following fields and sections:

Font

Use this drop-down list to choose the font for which you want to display characters.

Unicode Block

Use this drop-down list to only see a certain range of characters. 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.

Search

Use this filter to search for a character by one of the following attributes:

- **hexadecimal**
- **decimal**
- **description**

Note: Selecting the **description** option opens the **Details** tab. If you enter a character description in the **Search** field, the **description** option is selected automatically.

Character Table Section

The characters that are available to be inserted are listed in two tabs:

- **Compact** - Matrix-like table that displays a visual representation of the characters.
- **Details** - Displays the available characters in a tabular format, presenting their decimal and hexadecimal value along with their description.

Recently Used Characters Section

Displays the symbols that you have used recently and you can also select one from there to insert it in the current document.

Character Mode Section

The next section of the dialog box allows you to select how you want the character to appear in the **Text** editing mode. You can choose between the following:

- **Character**
- **Character entity - decimal**
- **Character entity - hexadecimal**

You can see the character or code that will be inserted in **Text** mode next to the selections in this section and a box on the right side of the dialog box allows you to see the character that will be inserted in **Author** mode. You can also see the name and range name of a character either at the bottom of the dialog box, or in a tooltip when hovering the cursor over the character.

Press the **Insert** button to insert the selected character in the current editor at cursor 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.

Note: The **Character Map** dialog box is not available in *the **Grid** editor*.

Unicode Fallback Font Support

Oxygen XML Editor plugin provides fonts for most common Unicode ranges. However, if you use special symbols or characters that are not included in the default fonts, they will be rendered as small rectangles. A *fallback* font is a reserve typeface that contains symbols for as many Unicode characters as possible. When a display system encounters a character that is not part of the range of any of the available fonts, Oxygen XML Editor plugin will try to find that symbol in a *fallback* font.

Example of a Scenario Where a Fallback Font is Needed

Suppose that you need to insert the Wheelchair symbol (♿ - `U+267F`) into your content in a Windows operating system. By default, Oxygen XML Editor plugin does not render this symbol correctly since it is not included in any of the default fonts. It is included in *Segoe UI Symbol*, but this font is not part of the default fonts that come with Oxygen XML Editor plugin. To allow Oxygen XML Editor plugin to recognize and render the symbol correctly, you can add *Segoe UI Symbol* as a *fallback* font.

Add a Fallback Font in Windows (7 or Later)

To add a fallback font to the Oxygen XML Editor plugin installation, use the following procedure:

1. Start Windows Explorer and browse to the `[OXYGEN_INSTALLATION_DIR]/jre/lib/fonts` directory.
2. Create a directory called `fallback` (if it is not already there).
3. Copy a font file (True Type Font - TTF) that includes the special characters into this directory.

   Tip: You could, for example, copy the *Segoe UI Symbol Regular* font from `C:\Windows\Fonts`.

4. Restart Oxygen XML Editor plugin for the changes to take full effect.

Result: Whenever Oxygen XML Editor plugin finds a character that cannot be rendered using its standard fonts, it will look for the glyph in the fonts stored in the `fallback` folder.

Alternate Solution for Other Platforms

For Mac OS X or other platforms, you could use the following approach:

1. Use a font editor (such as *FontForge*) to combine multiple true type fonts into a single custom font.
2. Install the font file into the dedicated font folder of your operating system.
3. In Oxygen XML Editor plugin, *open the **Preferences** dialog box* and go to **Appearance** > **Fonts**.
4. Click the **Choose** button in the **Editor** option and select your custom font from the drop down list in the subsequent dialog box.
5. Restart Oxygen XML Editor plugin for the font changes to take full effect.

Creating and Working with Documents

Oxygen XML Editor plugin includes various features, actions, and wizards to assist you with creating new files and working with existing files. This section explains many of these features, including information on creating new documents, opening, saving, and closing existing files, searching documents, viewing file properties, and more.

Creating New Documents

Oxygen XML Editor plugin includes a handy **New Document** wizard that allows you to customize and create new files from a large list of document types and predefined new file templates. You can also create your own templates and share them with others.

Oxygen XML Editor plugin New Document Wizard

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

The Oxygen XML Editor plugin installs a series of Eclipse wizards for easy creation of documents. If you use these wizards, Oxygen XML Editor plugin automatically completes the following details:

- The system ID, or schema location of a new XML document.
- The minimal markup of a DocBook article, or the namespace declarations of a Relax NG schema.

1. To create a document, either select **File** > **New** > **Other** > **Oxygen XML Editor plugin** or click the  **New** button on the toolbar.
   The **New** file wizard is displayed.
2. Select a document type. You can also select ***New from Templates*** to create a document based upon predefined templates or custom templates.
3. Click the **Next** button.
   For example if XML was selected the **Create an XML Document** wizard is started.

   The **Create an XML Document** dialog box enables definition of an XML Document Prolog using the system identifier of an XML Schema, DTD, Relax NG (full or compact syntax) schema, or NVDL (Namespace-based Validation Dispatching Language) schema. As not all XML documents are required to have a Prolog, you can choose to skip this step by clicking **OK**. If the prolog is required, complete the fields as described in the next step.
4. Type a name for the new document and press the **Next** button.
5. If you select **Customize**, Oxygen XML Editor plugin opens the following dialog box. You can customize various options, depending on the document type you select.

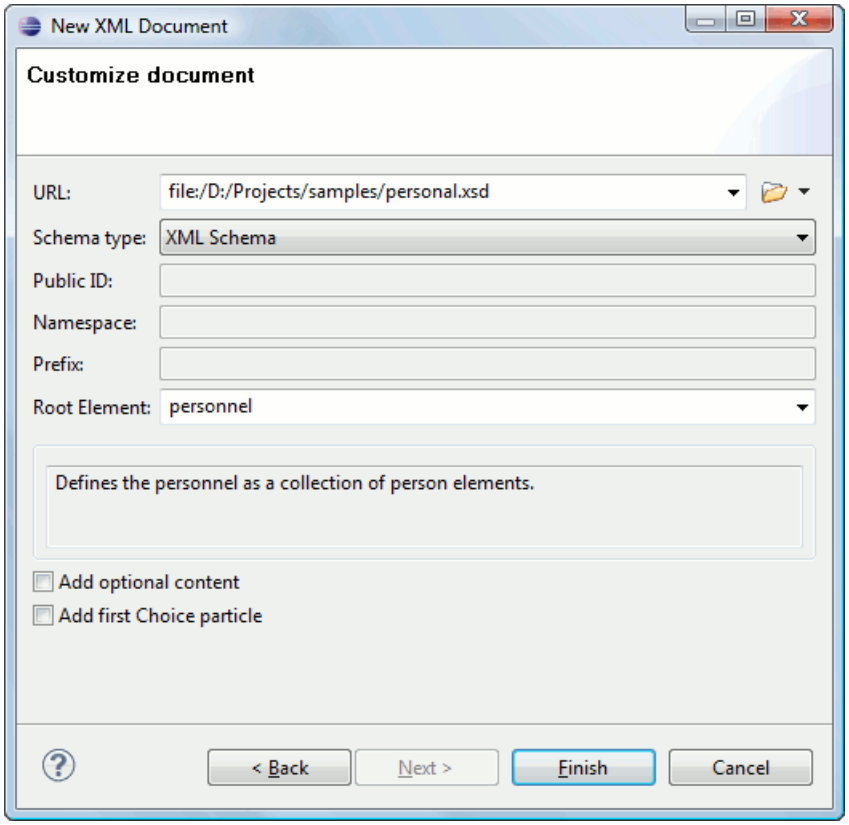


Figure 52: New XML Document Dialog Box

- **URL** - Specifies the path to the schema file. When you select a file, Oxygen XML Editor plugin analyzes its content and tries to fill the rest of the dialog box.
- **Schema type** - Allows you to select the schema type. The following options are available: XML Schema, DTD, RelaxNG XML syntax, RelaxNG compact syntax, and NVDL.
- **Public ID** - Specifies the PUBLIC identifier declared in the document prolog.
- **Namespace** - Specifies the document namespace.
- **Prefix** - Specifies the prefix for the namespace of the document root.
- **Root Element** - Populated with elements defined in the specified schema, enables selection of the element used as document root.
- **Description** - 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 Editor plugin generates the first element of an `xs:choice` schema element in the skeleton XML document. Oxygen XML Editor plugin creates this document in a new editor panel when you click **Finish**.

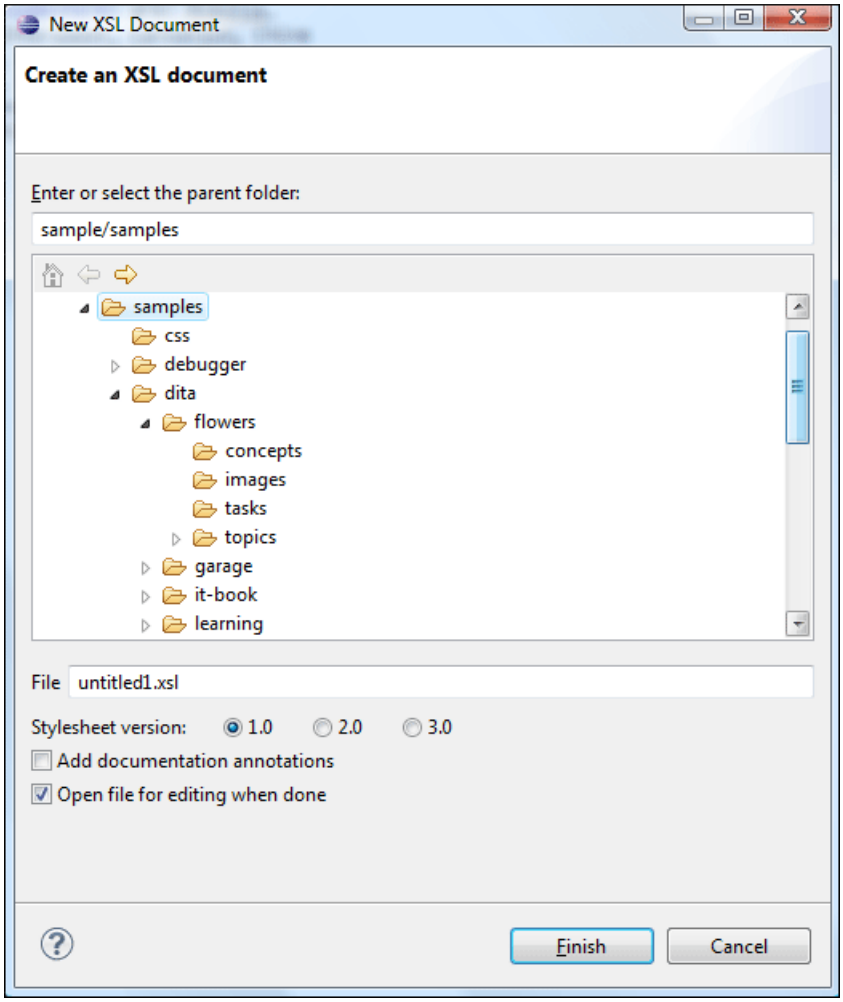


Figure 53: New XSL Document Dialog Box

- **Stylesheet version** - Allows you to select the Stylesheet version number. You can select from: 1.0, 2.0, and 3.0.
- **Add documentation annotations** - Adds annotation for XSL components.
- **Open file for editing when done** - Specifies whether or not you want to open the newly created file in the editor.

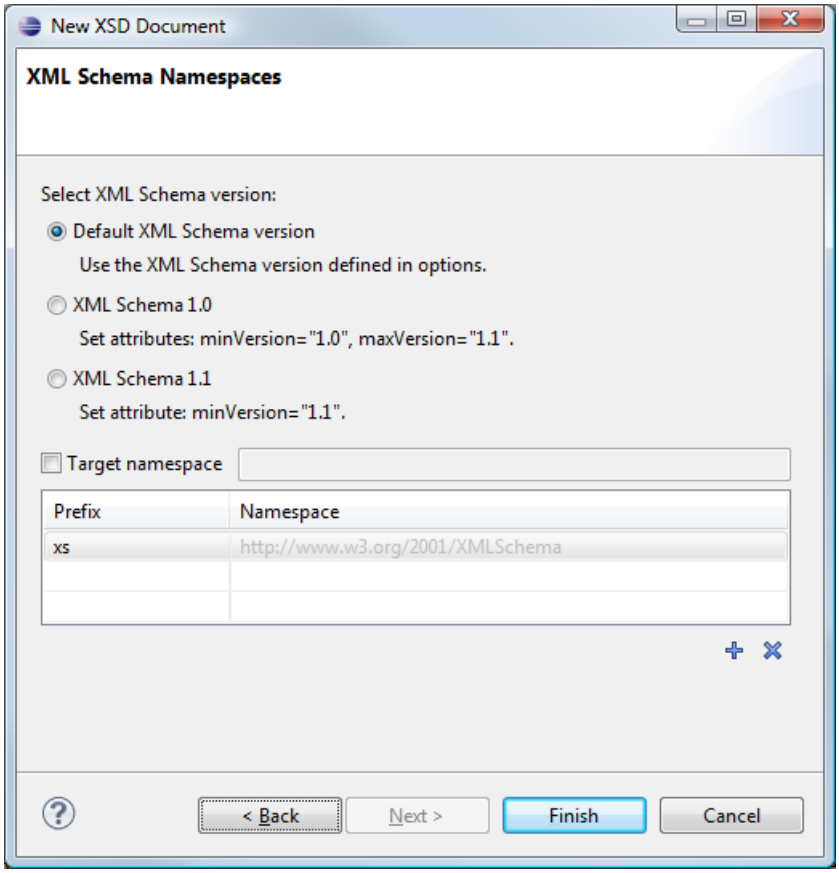


Figure 54: New XML Schema Document (XSD) Dialog Box

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

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

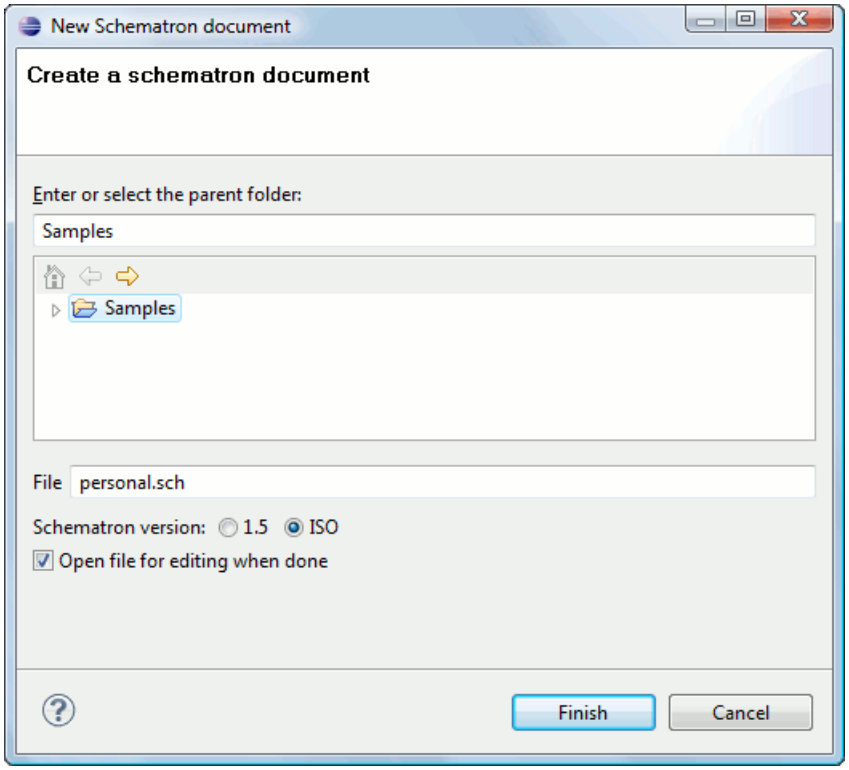


Figure 55: New Schematron Document Dialog Box

- **Schematron version** - Specifies the Schematron version. Possible options: 1.5 and ISO.
- **Open file for editing when done** - Specifies whether or not you want to open the file for editing purposes.

Creating New Documents Based on Templates

*The **New Document** wizard* allows you to select predefined templates or custom templates that were created in previous sessions or by other users.

The list of templates presented in the dialog box includes:

- **User defined templates** - You can add your custom templates by creating template files in a directory and then add that directory to the list of template directories that Oxygen XML Editor plugin uses in the *Document Templates Preferences* page.

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

- **Global templates** - Contains a list of predefined templates.
- **Framework templates** - Contains the list of templates defined in the ***Document Type** configuration dialog box* (**Templates** tab).

To create a new document from a file template, follow these steps

1. Open the **New from Templates** wizard by doing one of the following:
   a) Go to **File** > **New** > **Other** > **Oxygen XML Editor plugin** > **New From Templates**.
   b) Click the **New** button on the toolbar and select **New from Templates**.
2. Select a document type and press **Next**.
3. Type a name for the new document and press **Next**.
4. Press the **Finish** button.

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

Saving Documents

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

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

Opening and Saving Remote Documents via FTP/SFTP

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

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

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

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

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

The Open Using FTP/SFTP/WebDAV Dialog Box

To access the **Open using FTP/SFTP/WebDAV** dialog box, go to **File** > **Open URL** menu, then choose the **Browse for remote file** option from the drop-down action list.

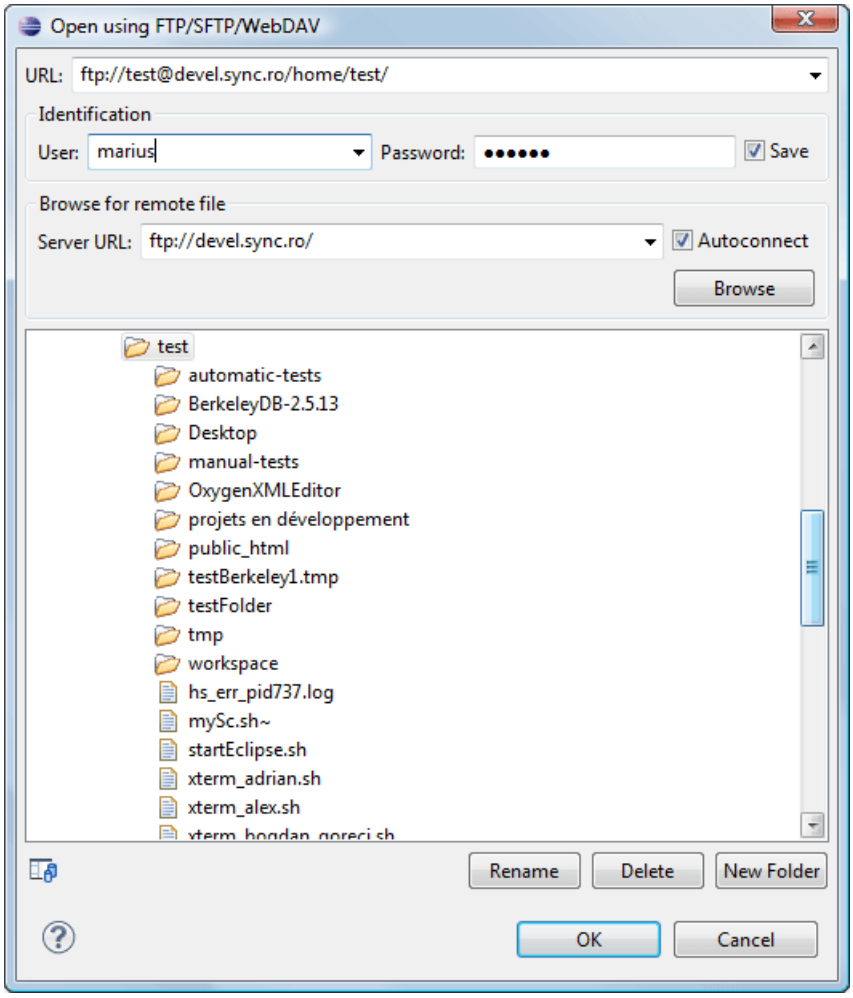


Figure 56: Open URL Dialog Box

The displayed dialog box is composed of several parts:

- The editable **URL** combo box, in which you specify the URL to be opened or saved.

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

  This combo box also displays the current selection when you change selection by browsing the tree of folders and files on the server.

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

  Note: Your password is well protected. If the options file is used on another machine by a user with a different user name the password, it will become unreadable since the encryption is user-name dependent. This is also true if you add URLs to your project that include a user and password.

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

  Tip: When accessing a FTP server, you only need to specify the protocol and the host (such as `ftp://server.com` or if using a nonstandard port `ftp://server.com:7800/`).

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

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

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

Changing File Permissions on a Remote FTP Server

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

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

WebDAV over HTTPS

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

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

Troubleshooting HTTPS

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

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

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

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

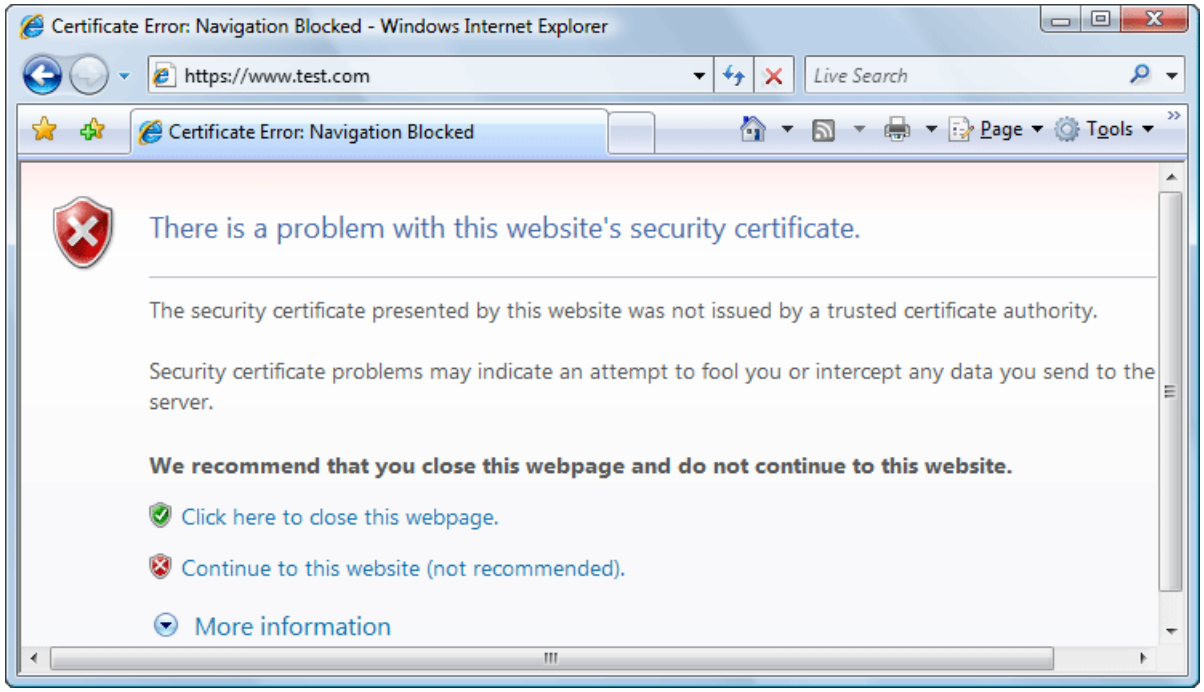


Figure 57: Security alert - untrusted certificate

b) Go to menu **Tools** > **Internet Options**.
Internet Options dialog box is opened.
c) Select **Security** tab.
d) Select **Trusted sites** icon.
e) Press **Sites** button.
This will open **Trusted sites** dialog box.
f) Add repository URL to **Websites** list.
g) Close the **Trusted sites** and **Internet Options** dialog boxes.
h) Try again to connect to the same repository URL in Internet Explorer.
The same error page as above will be displayed.
i) Select **Continue to this website** option.
A clickable area with a red icon and text **Certificate Error** is added to Internet Explorer address bar.
j) Click the **Certificate Error** area.
A dialog box containing a **View certificates** link is displayed.
k) Click the **View certificates** link.
Certificate dialog box is displayed.
l) Select **Details** tab of **Certificate** dialog box.
m) Press **Copy to File** button.
Certificate Export Wizard is started.
n) Follow indications of wizard for DER encoded binary X.509 certificate. Save certificate to local file `server.cer`.

2. Import the local file into the JRE running Oxygen XML Editor plugin.

a) Open a text-mode console with administrative rights.
b) Go to the `lib/security` directory of the JRE running Oxygen XML Editor plugin. You find the home directory of the JRE in the *java.home* property that is displayed in the **About** dialog box (**Installation Details** > **Configuration**). On Mac OS X systems, the `lib/security` directory is usually located in `/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home` directory.
c) Run the following command:

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

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

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

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

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

3. Restart Oxygen XML Editor plugin.

HTTP Authentication Schemes

Oxygen XML Editor plugin supports the following HTTP authentication schemes:

- **Basic** - The *basic* authentication scheme defined in the *RFC2617 specifications*.
- **Digest** - The *digest* authentication scheme defined in the *RFC2617 specifications*.
- **NTLM** - The *NTLM* scheme is a proprietary Microsoft Windows Authentication protocol (considered to be the most secure among currently supported authentication schemes).

  Note: For NTLM authentication, the user name must be preceded by the name of the domain it belongs to, as in the following example:

  ```
  domain\username
  ```

- ***Kerberos*** - An authentication protocol that works on the basis of *tickets* to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner.

Single Sign-on

Oxygen XML Editor plugin implements the *Single sign-on* property (meaning that you can log on once and gain access to multiple services without being prompted to log on for each of them), based on the *Kerberos* protocol and relies on a *ticket-granting ticket (TGT)* that Oxygen XML Editor plugin obtains from the operating system.

To turn on the *Kerberos*-based authentication, you need to add the following system property in the `eclipse.ini` configuration file (on a separate line after the `-vmargs` parameter):

```
-Djavax.security.auth.useSubjectCredsOnly=false
```

Closing Documents

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

- Click **Close(Ctrl F4 (Command F4 on OS X))** in the contextual menu of an open tab (or from the **File** menu) to close it.
- Click **Close Other Files** in the contextual menu of an open tab (or from the **File** menu) to close all the open tabs except the selected one.
- Click **Close All(Ctrl Shift F4 (Meta Shift F4 on OS X))** in the contextual menu of an open tab (or from the **File** menu) to close all open tabs.

The Contextual Menu of the Current Editor Tab

The contextual menu is available when you right-click the current editor tab label. It includes 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

The **Editor Properties** view, you can quickly access information about the currently edited document. The information includes:

- Character encoding.
- Full path on the file system.
- Schema used for content completion and document validation.
- Document type name and path.
- Associated transformation scenario.
- Read-only state of a file.
- Bidirectional text (left to right and right to left) state.
- Total number of characters in the document.
- Line width.
- Indent with tabs state.
- Indent size.

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

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

Using Projects to Group Documents

This section explains how to create and work with Projects.

Creating a New Project

Oxygen XML Editor plugin allows you to organize your XML-related files into projects. This helps you manage and organize your files and also allows you to perform batch operations (such as validation and transformation) over multiple files. Use the ***Navigator*** *view* to manage projects, and the files and folders contained within.

Creating a New Project

To create a new project, select **New** > **XML Project** or **New** > **Sample XML Project** from the contextual menu or **File** menu. This opens a dialog box that allows you to create and customize a new project and adds it to the structure of the project in the ***Navigator*** *view*.

Adding Items to the Project

To add items to the project, select the desired document type or folder from the **New** menu of the contextual menu, when invoked from the **Navigator** view (or from the **File** menu). You can also create a document from a template by selecting **New** > **New from Templates** from the contextual menu.

Using Linked Folders (Shortcuts)

Another easy way to organize your XML working files is to place them in a directory and then to create a corresponding linked folder in you project. If you add new files to that folder, you can simply use the **Refresh (F5)** action from the toolbar or contextual menu and the **Navigator** view will display the existing files and subdirectories. If your files are scattered amongst several folders, but represent the same class of files, you might find it useful to combine them in a logical folder.

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

Note: Files may have multiple instances within the folder system, but cannot appear twice within the same folder.

For much more information on managing projects and their content, see the *The Navigator View* on page 59 section.

The Navigator View

The **Navigator** view is designed to assist you with organizing and managing related files grouped in the same XML project. The actions available on the contextual menu and toolbar associated to this panel enable the creation of XML projects and shortcuts to various operations on the project documents.

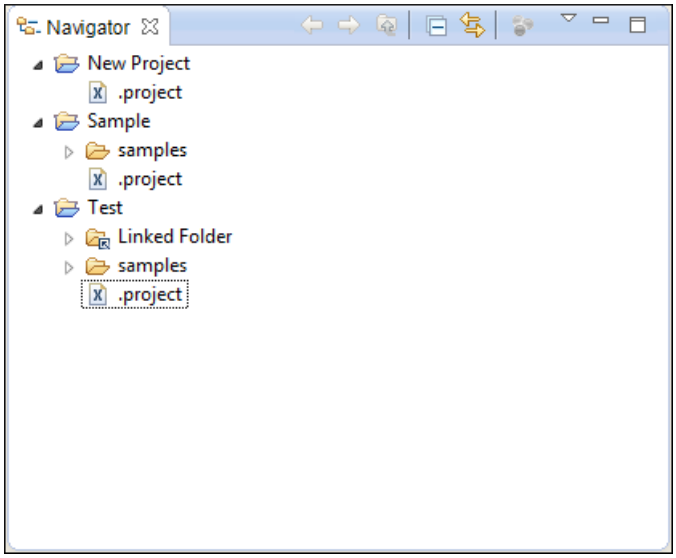


Figure 58: The Navigator View

The following actions are grouped in the upper right corner:

Collapse All

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

Link with Editor

When selected, the project tree highlights the currently edited file, if it is found in the project files.

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

View Menu

Drop-down menu that contains various settings.

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

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

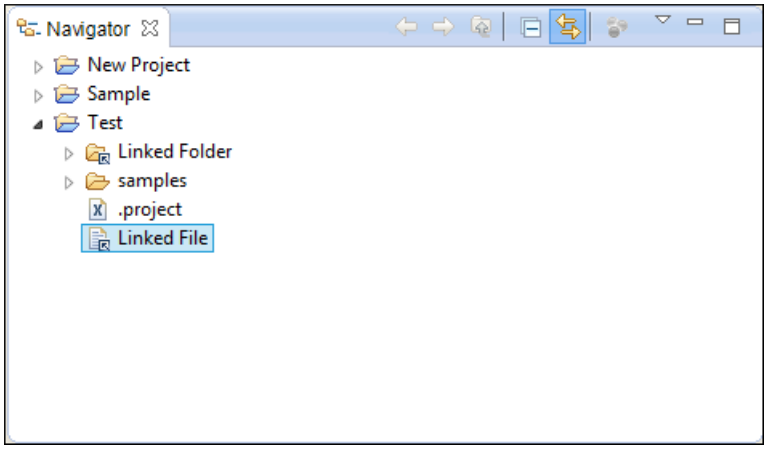


Figure 59: The Navigator View with Examples of the Two Types of Resources

Creating New Projects

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

New > XML Project

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

New > Sample XML Project

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

Creating New Project Items

To create new project items, select the desired document type or folder from the **New** menu of the contextual menu, when invoked from the **Navigator** view (or from the **File** menu). You can also create a document from a template by selecting **New** > **New from Templates** from the contextual menu.

New > **File**

Opens a **New** file dialog box that helps you create a new file and adds it to the project structure.

New > **Folder**

Opens a **New Folder** dialog box that allows you to specify a name for a new folder and adds it to the structure of the project.

New > **Logical Folder**

Available when invoked from the *project root*, this action creates a logical folder in the tree structure (the icon is a magenta folder on Mac OS X -).

New > Logical Folders from Web

Available when invoked from the *project root*, this action 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.

Managing Project Content

Creating/Adding Files and Folders

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

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

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

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

Removing Files and Folders

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

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

Moving Files and Folders

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

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

Renaming Files and Folders

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

- Invoke the **Rename** action from the contextual menu.
- Press **F2** and type the new name.

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

Locating and Opening Files

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

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

Saving the Project

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

Validate Files

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

Validate > **Check Well-Formedness**

Checks if the selected file or files are well-formed.

Validate **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.

ValidateValidate with Schema

Validates the selected file of files against a specified schema.

Validate **Configure Validation Scenario(s)**

Allows you to configure and run a *validation scenario*.

Validate > **Clear Validation Markers**

Clears all the error markers from the main editor and **Problems** view.

Applying Transformation Scenarios

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

Transform > **Apply Transformation Scenario(s)**

Obtains the output with one of the built-in scenarios.

Transform > **Configure Transformation Scenario(s)**

Opens a dialog box that allows you to configure pre-defined transformation scenarios.

Transform > **Transform with**

Allows you to select a transformation scenario to be applied to the currently selected files.

Refactoring Actions (Available for certain document types (such as XML, XSD, and XSL)

Oxygen XML Editor plugin includes some refactoring operations that help you manage the structure of your documents. The following actions are available from the contextual menu in the **Refactoring** sub-menu:

Refactoring > Rename resource

Allows you to change the name of a resource.

Refactoring > Move resource

Allows you to change the location on disk of a resource.

Refactoring > XML Refactoring

Opens *the XML Refactoring tool wizard* that presents refactoring operations to assist you with managing the structure of your XML documents.

Other Contextual Menu Actions

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

Open

Opens the selected file in the editor.

Open with submenu

This submenu offers you choices for opening the selected file in various editors.

Refresh

Refreshes the content and the dependencies between the resources in *the **Master Files** directory*.

XPath in Files

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

Check Spelling in Files

Allows you to *check the spelling of multiple files.*

Format and Indent Files

Opens the ***Format and Indent Files** dialog box* that allows you to configure the format and indent (pretty print) action that will be applied on the selected documents.

Properties

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

Moving/Renaming Resources in the Navigator View

The **Navigator** view allows you to move or rename files in the current project.

Moving Resources

To move a file or directory in the **Navigator** view, drag and drop it to the new location in the tree structure or use the **Move** action from the contextual menu (you can also use the **Copy** and **Paste** actions from the contextual menu or **Edit** menu to copy resources to a new location).

You can also move certain types of files (such as XML, XML Schema, Relax NG, WSDL, and XSLT) by using the **Refactoring** > **Move resource** action from the contextual menu. This action opens the **Move resource** dialog box that includes the following options:

- **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, based upon the selected scope. You can *select or configure the scope* by using the button.

Renaming Resources

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.

You can also rename certain types of files (such as XML, XML Schema, Relax NG, WSDL, and XSLT) by using the **Refactoring** > **Rename resource** action from the contextual menu. This action opens the **Rename resource** dialog box that includes the following options:

- **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. You can *select or configure the scope* by using the button.

Problems Updating References of Moved/Renamed Resources

In some case the references of a moved or a renamed resource can not be updated. For example, when a resource is resolved through an XML catalog or when the path to the moved or renamed resource contains entities. For these cases, Oxygen XML Editor plugin displays a warning dialog box.

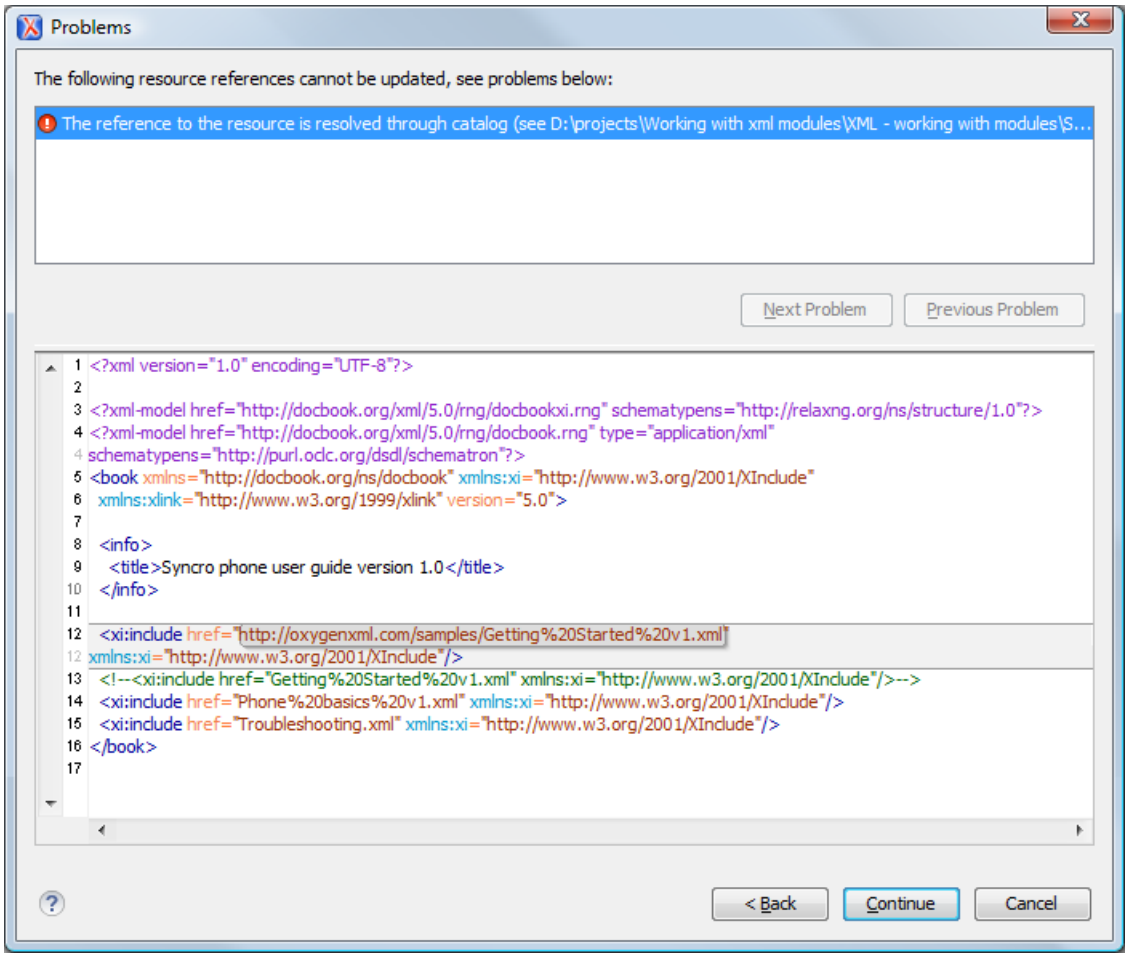


Figure 60: Problems Dialog Box

Defining Master Files at Project Level

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

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

Introduction to Master Files at Project Level

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

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

Master Files Benefits

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

- When the module is validated, Oxygen XML Editor plugin automatically identifies the *master files* that include the module and validates all of them.
- The ***Content Completion Assistant*** presents all the components that are collected, from the *master files* to the modules they include.
- The **Outline** view displays all the components that are defined in the *master files* hierarchy.
- The *master files* that are defined for the current module determines the *scope of the search and refactoring actions*. Oxygen XML Editor plugin performs the search and refactoring actions in the context that the *master files* determine, thus improving the speed of execution.

Enabling the Master Files Support

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

Detecting Master Files

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

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

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

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

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

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

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

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

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

Adding/Removing a Master File

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

- Right-click a file from your project and select **Add to Master Files** from the contextual menu.
- Drag and drop files into the `Master Files` directory.
- From the contextual menu of the **Resource Hierarchy Dependencies** view, use the **Add to Master Files** action.

You can view the *master files* for the currently edited resource in the **Editor Properties** view.

Editing XML Documents

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

Editing XML Documents in Text Mode

This section includes features and actions for editing XML documents in the **Text** mode of Oxygen XML Editor plugin.

Contextual Menu Actions in Text Mode

When editing XML documents in **Text** mode, Oxygen XML Editor plugin provides the following actions in the contextual menu (many of them also appear in the submenus of the **Document** menu):

Cut, Copy, Paste

Executes the typical editing actions on the currently selected content.

Copy XPath

Copies the XPath expression of the current element or attribute from the current editor to the clipboard.

Toggle Comment (Ctrl Shift Comma (Meta Shift Comma on OS X))

Comments the current selection of the current editor. If the selection already contains a comment the action removes the comment from around the selection. If there is no selection in the current editor and the cursor is not positioned inside a comment the current line is commented. If the cursor is positioned inside a comment then the commented text is uncommented.

Go to submenu

This submenu includes the following actions:

Go to Matching Tag (Ctrl Shift G)

Moves the cursor to the end tag that matches the start tag, or vice versa.

Go after Next Tag (Ctrl Close Bracket (Meta Close Bracket on OS X))

Moves the cursor to the end of the next tag.

Go after Previous Tag (Ctrl Open Bracket (Meta Open Bracket on OS X))

Moves the cursor to the end of the previous tag.

Select submenu

This submenu allows you to select the following:

Element

Selects the entire element at the current cursor position.

Content

Selects the entire content of the element at the current cursor 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.

Attributes

Selects all the attributes of the element at the current cursor position.

Parent

Selects the parent of the element at the current cursor position.

Source submenu

This submenu includes the following actions:

 **Shift Right**

Shifts the currently selected block to the right.

 **Shift Left**

Shifts the currently selected block to the left.

 **Indent selection (Ctrl I (Meta I on OS X))**

Corrects the indentation of the selected block of lines if it does not follow the current *indenting preferences*.

 **Escape Selection**

Escapes a range of characters by replacing them with the corresponding character entities.

 **Unescape Selection**

Replaces the character entities with the corresponding characters.

 **Format and Indent Element (Ctrl Shift I (Meta Shift I on OS X))**

Pretty prints the element that surrounds the current cursor position.

Convert Hexadecimal Sequence to Character (Ctrl + Shift + H (Meta + Shift + H on OS X))

Converts a sequence of hexadecimal characters to the corresponding Unicode character. A valid hexadecimal sequence can be composed of 2 to 4 hexadecimal characters, preceded or not by the `0x` or `0X` prefix. Examples of valid sequences: `0x0045`, `0X0125`, `1253`, `265`, `43`.

To convert a hexadecimal sequence to a character, do one of the following:

- place the cursor at the right side of a valid hexadecimal sequence, then press **Ctrl + Shift + H (Meta + Shift + H on OS X)** on your keyboard
- select a valid hexadecimal sequence, then press **Ctrl + Shift + H (Meta + Shift + H on OS X)** on your keyboard

Join and Normalize Lines

For the current selection, this action joins the lines by replacing the *line separator* with a single space character. It also normalizes the whitespaces by replacing a sequence of such characters with a single space.

 **Insert XInclude**

Displays a dialog box that allows you to browse and select the content to be included and automatically generates the corresponding XInclude instruction.

Note: In the **Author** mode, this dialog box presents a preview of the inserted document as an author page in the **preview** tab and as a text page in the **Source** tab. In the **Text** mode, the **Source** tab is presented.

 **Import entities list**

Displays a dialog box that allows you to select a list of files as sources for external DTD entities. The internal subset of the DOCTYPE declaration of your document will be updated with the chosen entities. For instance, choosing the files `chapter1.xml` and `chapter2.xml` inserts the following section in the DOCTYPE:

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

Canonicalize

Opens the **Canonicalize** dialog box that allows you to select a canonicalization algorithm to standardize the format of the document.

Sign

Opens the **Sign** dialog box that allows you to configure a digital signature for the document.

Verify Signature

Allows you to specify the location of a file to verify its digital signature.

Manage Highlighted Content submenu

This submenu is available from the contextual menu when it is invoked from a highlight after you perform a search operation or apply an XPath expression that highlights more than one result. The following options are available in this submenu:

Modify All

Allows you to modify (in-place) all the occurrences of the selected content. A thin rectangle replaces the highlights and allows you to start editing. If matches with different letter cases are found, a dialog box is displayed that allows you select whether you want to modify only matches with the same letter case or all matches.

Surround All

Surround the highlighted content with a specific tag. This option opens the **Tag** dialog box. The **Specify the tag** drop-down menu presents all the available elements that you can choose from.

Remove All

Removes all the highlighted content.

Modify All Matches

Use this option to modify (in-place) all the occurrences of the selected content (or the contiguous fragment in which the cursor is located). When you use this option, a thin rectangle replaces the highlights and allows you to start editing. If matches with different letter cases are found, a dialog box is displayed that allows you select whether you want to modify only matches with the same letter case or all matches.

Show Definition (Ctrl Shift Enter)

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. If the current attribute is a “type” belonging to the “*http://www.w3.org/2001/XMLSchema-instance*” namespace, the cursor is moved in the XML schema to the definition of the type referenced in the value of the attribute.

Refactoring submenu

This submenu includes the following actions:

Rename Element

The element from the cursor position, and any elements with the same name, can be renamed according with the options from the **Rename** dialog box.

Rename Prefix (Alt Shift P)

The prefix of the element from the cursor position, and any elements with the same prefix, can be renamed according with the options from the **Rename** dialog box.

- If you select the **Rename current element prefix** option, the application will recursively traverse the current element and all its children.

  Note: For example, to change the `xmlns:p1="ns1"` association in the current element to `xmlns:p5="ns1"`, if the `xmlns:p1="ns1"` association is applied on the parent element, then Oxygen XML Editor plugin will introduce `xmlns:p5="ns1"` as a new declaration in the current element and will change the prefix from `p1` to `p5`. If `p5` is already associated with another namespace in the current element, then the conflict will be displayed in a dialog box. By pressing **OK**, the prefix is modified from `p1` to `p5` without inserting a new declaration.

- If you select the **Rename current prefix in all document** option, the application will apply the change on the entire document.
- To also apply the action inside attribute values, check the **Rename also attribute values that start with the same prefix** checkbox.

Surround with submenu

Presents a drop-down menu that allows you to choose a tag to surround a selected portion of content.

Surround with Tags (Alt + Shift + E)

Allows you to choose a tag that encloses a selected portion of content. If there is no selection, the start and end tags are inserted at the cursor position.

- If the **Cursor position between tags** option is enabled in the *Content Completion Preferences page*, the cursor is placed between the start and end tag.
- If the **Cursor position between tags** option is disabled in the *Content Completion Preferences page*, the cursor is placed at the end of the start tag, in an insert-attribute position.

Surround with (Alt + Shift + /)

Surround the selected content with the last tag used.

Delete element tags (Alt + Shift + ,)

Deletes the start and end tag of the current element.

Split element

Split the element from the cursor position into two identical elements. The cursor must be inside the element.

Join elements (Alt Shift F (Meta Alt F on OS X))

Joins the left and right elements relative to the current cursor position. The elements must have the same name, attributes, and attributes values.

Attributes submenu

Contains predefined XML refactoring operations that pertain to attributes. Oxygen XML Editor plugin considers the editing context to get the names and namespaces of the element or attribute for which the contextual menu was invoked, and uses this information to preconfigure some of the parameter values for the selected refactoring operation.

Add/Change attribute

Allows you to change the value of an attribute or insert a new one.

Delete attribute

Allows you to remove one or more attributes.

Rename attribute

Allows you to rename an attribute.

Replace in attribute value

Allows you to search for a text fragment inside an attribute value and change the fragment to a new value.

Elements submenu

Contains predefined XML refactoring operations that pertain to elements. Oxygen XML Editor plugin considers the editing context to get the names and namespaces of the element or attribute for which the contextual menu was invoked, and uses this information to preconfigure some of the parameter values for the selected refactoring operation.

Delete element

Allows you to delete elements.

Delete element content

Allows you to delete the content of elements.

Insert element

Allows you to insert new elements.

Rename element

Allows you to rename elements.

Unwrap element

Allows you to remove the surrounding tags of elements, while keeping the content unchanged.

Wrap element

Allows you to surround elements with element tags.

Wrap element content

Allows you to surround the content of elements with element tags.

Fragments submenu

Contains predefined XML refactoring operations that pertain to XML fragments. Oxygen XML Editor plugin considers the editing context to get the names and namespaces of the element or attribute for which the contextual menu was invoked, and uses this information to preconfigure some of the parameter values for the selected refactoring operation.

Insert XML fragment

Allows you to insert an XML fragment.

Replace element content with XML fragment

Allows you to replace the content of elements with an XML fragment.

Replace element with XML fragment

Allows you to replace elements with an XML fragment.

Manage IDs submenu

This submenu is available for XML documents that have an associated DTD, XML Schema, or Relax NG schema. It includes the following actions:

 **Rename in**

Renames the ID and all its occurrences. Selecting this action opens the **Rename XML ID** dialog box. This dialog box 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 box, 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.

 **Search References**

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

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 Declarations**

Searches for the declaration of the ID reference. By default, the scope of this action is the current project. If you configure a scope using the ***Select the scope for the Search and Refactor operations*** dialog box, 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 Occurrences in file

Searches for the declaration an references of the ID in the current document.

Quick Assist (Ctrl 1 (Meta 1 on OS X))

Available when the cursor is inside an ID or IDREF, this action opens the Quick Assist window that allows you to select some search and refactoring actions for the selected ID or IDREF.

Open File at Cursor

Opens the file at the cursor position in a new panel. If the file path represents a directory path, it will be opened in system file browser. If the file at the specified location does not exist, an error dialog box is displayed and it includes a **Create new file** button that 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 referenced location and name and is opened in a new editor panel.

Resource Hierarchy

Opens the **Resource Hierarchy/Dependencies** view that allows you to see the resource hierarchy for an XML document.

Resource Dependencies

Opens the **Resource Hierarchy/Dependencies** view that allows you to see the resource dependencies for an XML document.

Smart Editing

Oxygen XML Editor plugin includes *smart editing* features to help you edit XML documents in **Text** mode. The following smart editing features are included:

- *Closing tag auto-expansion* - This feature helps save some keystrokes by automatically inserting a closing tag when you insert a complete start tag and the cursor is automatically placed in between the start and end tags. For instance, after entering a start `<tag>`, the corresponding closing `</tag>` is automatically inserted and the cursor is placed between the two (`<tag>|</tag>`.
- *Auto-rename matching tag* - When you edit the name of a start tag, Oxygen XML Editor plugin will mirror-edit the name of the matching end tag. This feature can be controlled from the *Content Completion option page*.
- *Auto-breaking the edited line* - The *Hard line wrap option* automatically breaks the edited line 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 you press **Enter**.
- *Smart Enter* - The *Smart Enter option* inserts an empty line between the start and end tags. If you press **Enter** between a start and 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 certain text, depending on the position of the click in the document:
  - If the click position is on a start tag or end tag, then the element name is selected.
  - If the click position is after a start tag or before an end tag, then the entire content of the element without the start and end tags is selected.
  - If the click position is before a start tag or after an end tag, then the entire tag is selected, including the start and end tags, and the content in between.
  - If the click position is immediately before an attribute, then the entire attribute and its value is selected.
  - If the click position is immediately after the opening quote or immediately before the closing quote of an attribute value, then the entire attribute value is selected.
  - Otherwise, a double-click selects contiguous text.
- *Triple-click* - A triple-click selects the entire current line of text.

Highlight ID 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 jump to the occurrence that it corresponds to. The occurrences are also highlighted in the editing area.

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

Quick Assist Support for IDs and IDREFS

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

The following actions are available:

Rename in

Renames the ID and all its occurrences. Selecting this action opens the **Rename XML ID** dialog box. This dialog box 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 box, 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. If you configure a scope using the ***Select the scope for the Search and Refactor operations*** dialog box, 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. If you configure a scope using the ***Select the scope for the Search and Refactor operations*** dialog box, this scope will be used instead.

Change scope

Opens the ***Select the scope for the Search and Refactor operations*** dialog box.

Rename in File

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

Search Occurrences

Searches for the declaration an references of the ID located at the cursor position in the current document.

Formatting and Indenting XML Documents

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

- **Document** > **Source** > **Format and Indent** - Formats and indents the whole document.
- **Document** > **Source** > **Indent Selection** - Indents the current selection (but does not add line breaks). This action is also available in the **Source** submenu of the contextual menu.
- **Document** > **Source** > **Format and Indent Element** - Formats and indents the current element (the inmost nested element which currently contains the cursor) and its child-elements. This action is also available in the **Source** submenu of the contextual menu.

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

Significant and insignificant whitespace in XML

XML documents are text files that describe complex documents. Some of the white space (spaces, tabs, line feeds, etc.) in the XML document belongs to the document it describes (such as the space between words in a paragraph) and some

of it belongs to the XML document (such as a line break between two XML elements). Whitespace belonging to the XML file is called *insignificant whitespace*. The meaning of the XML would be the same if the insignificant whitespace were removed. Whitespace belonging to the document being described is called *significant whitespace*.

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

```
<p>
NO Freeman shall be taken or imprisoned, or be disseised of his Freehold, or Liberties, or
free Customs, or be outlawed, or exiled, or any other wise destroyed; nor will We not pass
upon him, nor condemn him, but by lawful judgment of his Peers, or by the <xref
href="http://en.wikipedia.org/wiki/Law_of_the_land" format="html" scope="external">Law of the land</xref>.
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*.

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 Editor plugin determines when whitespace is significant

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

Ignore space

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

Normalize space

In the normalize space category, a single whitespace character between character strings is considered significant and all other spaces are considered insignificant. This generally applies to elements that contain text content only. This content can be normalized by removing insignificant whitespace. 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 Editor plugin consults several pieces of information to assign an element to one of these categories. An element is always assigned to the most restrictive category (from Ignore to Preserve) that it is assigned to by any of the sources Oxygen XML Editor plugin consults. For instance, if the element is named on the **Default elements** list (as described below) but it has an `xml:space="preserve"` attribute in the source file, it will be assigned to the preserve space category. If an element has the `xml:space="default"` attribute in the source, but is listed on the **Mixed content** elements list, it will be assigned to the mixed content category.

To assign elements to these categories, Oxygen XML Editor plugin 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 Editor plugin can use information from the schema to promote the element to the appropriate category. For example:

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

Schema aware formatting can be turned on and off.

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

Preserve space elements list

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

Default space elements list

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

Mixed content elements list

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

Element content

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

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

Text node content

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

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 Editor plugin formats and indents XML

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

- *Format Preferences* on page 939
- *XML Formatting Preferences* on page 940
- *Whitespaces Preferences* on page 942

When Oxygen XML Editor plugin formats and indents XML

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

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

Setting an Indent Size to Zero

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

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

There are two cases to consider.

Maintaining zero indent in documents with zero indent

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

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

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

Enforcing zero indent for all documents

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

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

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

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

Format and Indent (Pretty Print) Multiple Files

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

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

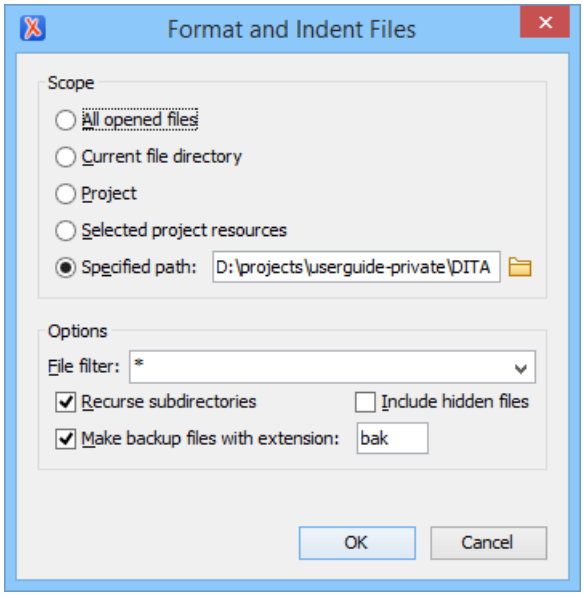


Figure 61: The Format and Indent Files Dialog Box

The **Scope** section allows you choose from the following scopes:

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

The **Options** section includes the following options:

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

Managing Highlighted Content

While working with XML documents you often have frequent changes to the structure and content. You are often faced with a situation where you need to make a slight change in multiple places in the same document. Oxygen XML Editor plugin includes a feature, **Manage Highlighted Content**, that is designed to help you achieve this.

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, and the following options are available in its submenu:

- **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 allows you to start editing. If matches with different letter cases are found, a dialog box is displayed that allows you select whether you want to modify only matches with the same letter case or all matches.

  Note: If you select a very large number of highlights that you want to modify using this feature, a dialog box informs you that you may experience performance issues. You have the option to either use the **Find/Replace** operation, or continue the operation.

- **Surround All** - Use this option to surround the highlighted content with a specific tag. This option opens the **Tag** dialog box. The **Specify the tag** drop-down menu presents all the available elements that you can choose from.

- **Remove All** - Removes all the highlighted content.

If you right-click content in another part of the document, other than a highlight, you have the option to select the following option:

- **Modify All Matches** - Use this option to modify (in-place) all the occurrences of the selected content (or the contiguous fragment in which the cursor is located). When you use this option, a thin rectangle replaces the highlights and allows you to start editing. If matches with different letter cases are found, a dialog box is displayed that allows you select whether you want to modify only matches with the same letter case or all matches.

Editing XML Documents in Grid Mode

This section includes features and actions for editing XML documents in the **Grid** mode of Oxygen XML Editor plugin.

Editing Actions in Grid Mode

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

Sorting a Table Column

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

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

Inserting a Row in a Table

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

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

Inserting a Column in a Table

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

Clearing the Content of a Column

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

Adding Nodes

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

- **Insert before** - Offers a list of valid nodes, depending on the context, and inserts your selection before the currently selected node, as a sibling.
- **Insert after** - Offers a list of valid nodes, depending on the context, and inserts your selection after the currently selected node, as a sibling.
- **Append child** - Offers a list of valid nodes, depending on the context, and appends your selection as a child of the currently selected node.

Duplicating Nodes

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

Refresh Layout

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

Start and Stop Editing a Cell Value

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

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

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

Drag and Drop in the Grid Editor

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

You can do the following with drag and drop:

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

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

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

Copy and Paste in the Grid Editor

The selection in the **Grid** mode is a bit complex compared to the selection in a text component. It consists of a current selected cell and additional selected cells. These additional cells are either selected with the cursor, or implied by the currently selected cell. To be more specific, consider that 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 Single-Click (Command Single-Click on OS X)**. Pasting these nodes relative to the current selected cell may be done in two ways: just below (after) as a brother, which is the default behavior, or as the last child of the selected cell.

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

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

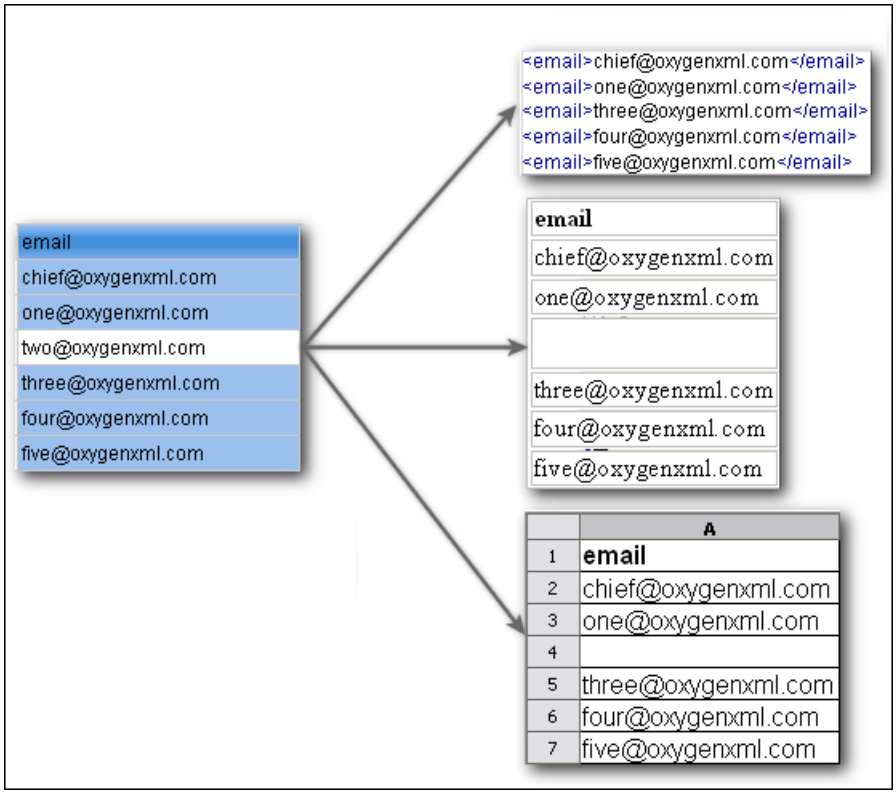


Figure 62: Copying from Grid to Other Editors

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.

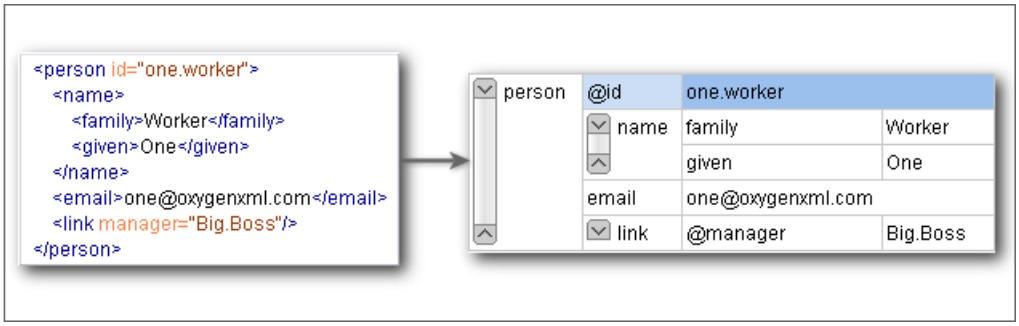


Figure 63: Copying XML Data into Grid

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.

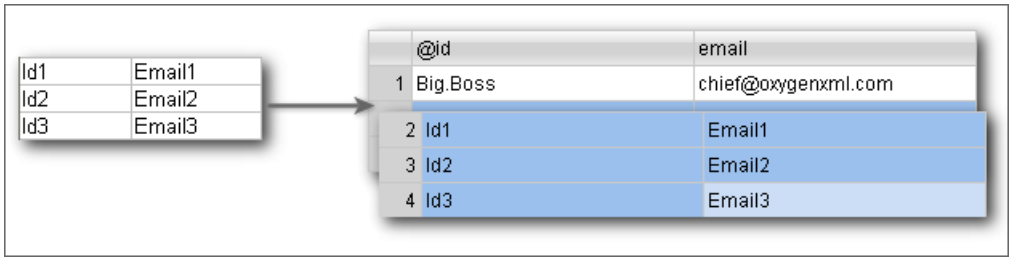


Figure 64: Copying Tab Separated Values into Grid

Editing XML Documents in Author Mode

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

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-style editor in which the XML markup is not visible.

This type of tagless editor is available in Oxygen XML Editor plugin as the **Author** mode. To enter this 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.

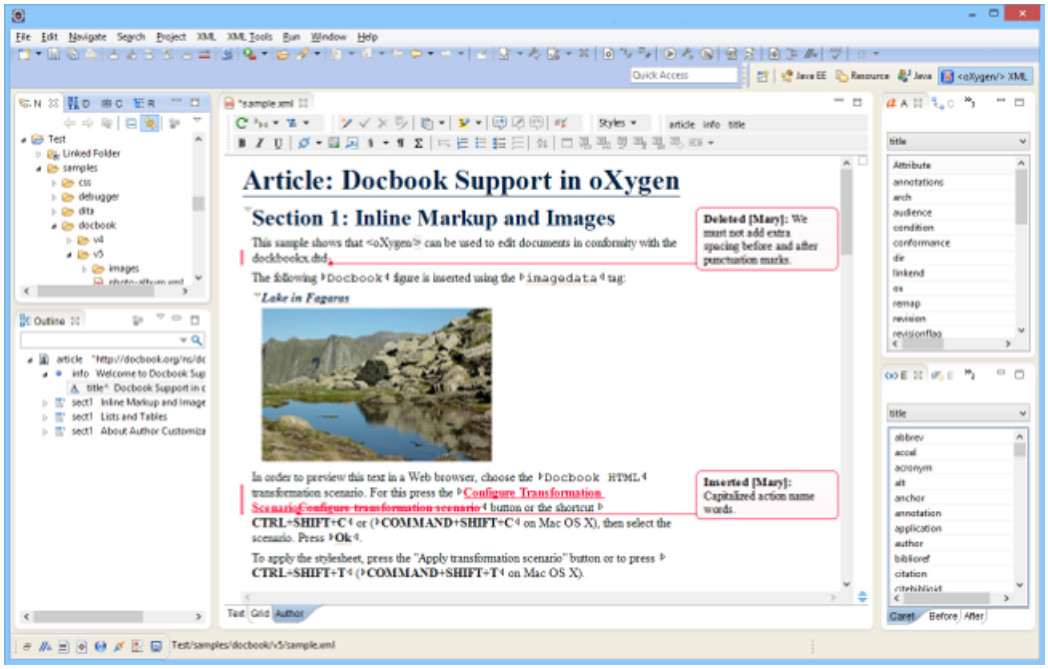


Figure 65: Author Editing Mode

Associating a Stylesheet 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, such as namespaces and custom extensions, of the CSS specification are also supported. Oxygen XML Editor plugin also supports stylesheets coded with the LESS dynamic stylesheet language.

There are several methods for associating a stylesheet (CSS or LESS) with an XML document:

1. Insert the `xml-stylesheet` processing instruction with the `type` attribute at the beginning of the XML document. If you do not want to alter your XML documents, *you should create a new document type (framework)*.

   CSS example:

   ```
   <?xml-stylesheet type="text/css" href="test.css"?>
   ```

   LESS example:

   ```
   <?xml-stylesheet type="text/css" href="test.less"?>
   ```

   Note: XHTML documents need a `link` element, with the `href` and `type` attributes in the `head` child element, as specified in the *W3C CSS specification*. XHTML example:

   ```
   <link href="/style/screen.css" rel="stylesheet" type="text/css"/>
   ```

   Tip: You can also insert the `xml-stylesheet` processing instruction by using the **Associate XSLT/CSS Stylesheet** action that is available on the toolbar or in the **XML** menu.

2. Configure a *Document Type Association* by adding a new CSS or LESS file in the settings. To do so, *open the **Preferences** dialog box* and go to **Document Type Association**. Edit the appropriate framework, open the **Author** tab, then the **CSS** tab. Press the **New** button to add a new CSS or LESS file.

   Note: The Document Type Associations are read-only, so you need to extend an existing one.

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

If a document has no CSS association or the referenced 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.

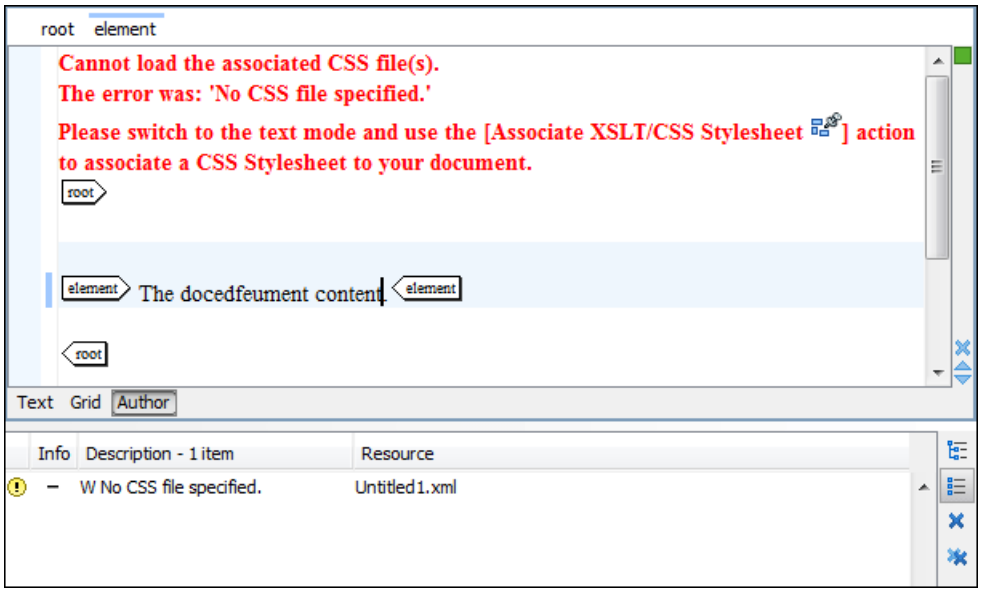


Figure 66: Document with no CSS association default rendering

Selecting and Combining Multiple CSS Styles

Oxygen XML Editor plugin provides a **Styles** drop-down menu on the **Author Styles** toolbar that allows you to select one *main* (*non-alternate*) CSS style and multiple *alternate* CSS styles. An option in the preferences can be enabled to allow the *alternate* styles to behave like layers and be combined with the *main* CSS style. This makes it easy to change the look of the document.

You can select a *main* CSS stylesheet that styles the whole document and then apply *alternate* styles, as layers, to specific parts of the document. In the subsequent figure, a DITA document has the **Century** style selected for the *main* CSS and the *alternate* styles **Full width**, **Show table column specification**, **Hints**, and **Inline actions** are combined for additive styling to specific parts of the document.

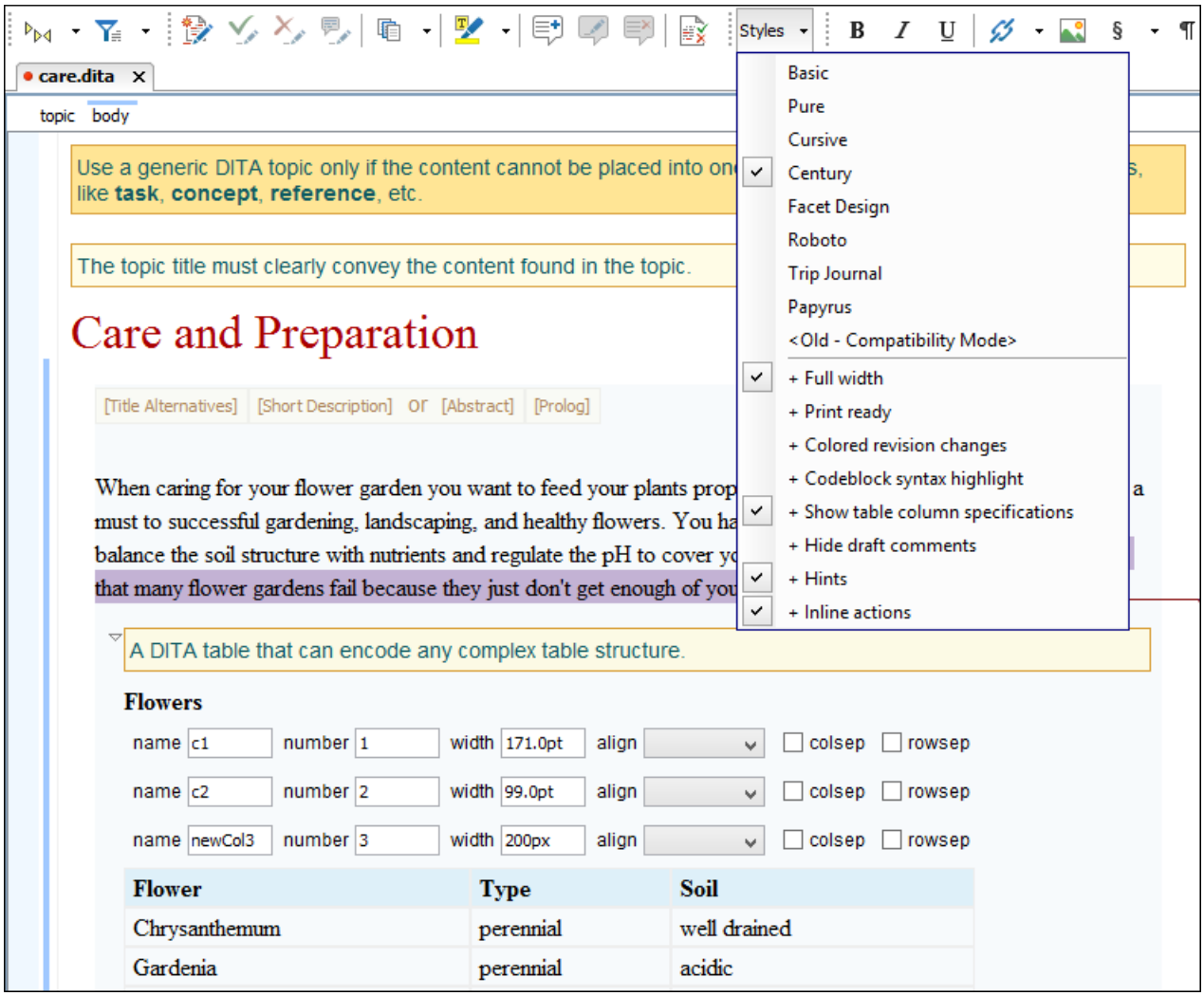


Figure 67: Styles Drop-down Menu in a DITA Document

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 for the content authors. A *content author* does not need to have advanced knowledge about XML tags, operations such as validation of XML documents, or applying an XPath expression to an XML document. The *content author* just uses the framework set-up by the developer in the application and starts editing the content of XML documents without editing the XML tags directly.

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

The framework details that are created and customized by the developer include:

- The CSS stylesheet that 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 such as DocBook, DITA, TEI, and XHTML.

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

Contextual Menu Actions in Author Mode

Oxygen XML Editor plugin includes powerful support for editing XML documents through actions included in the contextual menu. When editing XML documents in **Author** mode, the contextual menu includes *general* actions that are available for all of the recognized document types and *document type-specific* actions that are configured for each document type.

General Contextual Menu Actions in Author Mode

The *general* actions that are available in the contextual menu (some of them are also available in the submenus of the **Document** menu) for all document types include the following:

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

Available when the contextual menu is invoked on an error where *Oxygen XML Editor plugin 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 a default system application associated with the current image type.

Track Changes Actions

Available when the **Track Changes** feature is enabled and the contextual menu is invoked on a change. The following options are available:

Accept Change(s)

Accepts the tracked change located at the cursor position. If you select a part of a *delete* or *insert* change, 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 tracked change located at the cursor position. If you select a part of a *delete* or *insert* change, 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

Opens a dialog box that allows you to add a comment to an existing tracked change. The comment appears in a callout box and a tooltip (when hovering over the change).

Author Callout Actions

Available when the contextual menu is invoked on a callout. If enabled in the *Callouts preferences page*, the callouts are displayed in **Author** mode for comments, tracked insertion changes, or tracked deletion changes.

The following actions are available in the contextual menu of an insertion or deletion callout:

Accept Change(s)

Accepts the tracked change located at the cursor position. If you select a part of a *delete* or *insert* change, 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 tracked change located at the cursor position. If you select a part of a *delete* or *insert* change, 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

Opens a dialog box that allows you to add a comment to an existing tracked change. The comment appears in a callout box and a tooltip (when hovering over the change).

Edit Reference

If the fragment that contains a callout is a reference, use this option to go to the reference and edit the callout.

Callouts Options

Select this option to open a *preferences page* where you can configure the callout options.

The following actions are available in the contextual menu of a comment callout:

Edit comment

Opens the **Edit Comment** dialog box that allows you to edit the selected comment.

Remove comment

Removes a selected comment.

Callouts Options

Select this option to open a *preferences page* where you can configure the callout options.

Edit Attributes

Displays an *in-place attributes editor* that allows you to manage the attributes of an element.

Edit Profiling Attributes

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

Cut

Deletes the currently selected content and copies it to the clipboard.

Copy

Copies the currently selected content to the clipboard.

Paste

Pastes the content of the clipboard at the current cursor location.

Paste special submenu

This submenu includes special paste actions that are specific to each framework, as well as the following general paste actions:

Paste As XML

Pastes clipboard content that is considered to be XML.

Paste As Text

Pastes clipboard content, ignoring any structure or styling markup.

Insert submenu

This submenu includes insert actions that are specific to each framework, along with the following general action:

Insert Entity

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

- #<*decimal value*> - e. g. #65
- &#<*decimal value*>; - e. g. &#65

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

Select submenu

This submenu allows you to select the following:

Element

Selects the entire element at the current cursor position.

Content

Selects the entire content of the element at the current cursor 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.

Parent

Selects the parent of the element at the current cursor position.

Text submenu

This submenu contains the following actions:

To Lower Case

Converts the selected content to lower case characters.

To Upper Case

Converts the selected content to upper case characters.

Capitalize Sentences

Converts to upper case the first character of every selected sentence.

Capitalize Words

Converts to upper case the first character of every selected word. 0034

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 *change tracking* is ignored when counting words.

Convert Hexadecimal Sequence to Character (Ctrl + Shift + H (Meta + Shift + H on OS X))

Converts a sequence of hexadecimal characters to the corresponding Unicode character. A valid hexadecimal sequence can be composed of 2 to 4 hexadecimal characters, preceded or not by the `0x` or `0X` prefix. Examples of valid sequences: `0x0045`, `0X0125`, `1253`, `265`, `43`.

To convert a hexadecimal sequence to a character, do one of the following:

- place the cursor at the right side of a valid hexadecimal sequence, then press **Ctrl + Shift + H (Meta + Shift + H on OS X)** on your keyboard
- select a valid hexadecimal sequence, then press **Ctrl + Shift + H (Meta + Shift + H on OS X)** on your keyboard

Refactoring submenu

Contains a series of actions designed to alter the XML structure of the document:

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 position of the cursor. Thus, if the cursor 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 cursor position is between the two adjacent elements. Also, joining two elements can be done by pressing the **Delete** or **Backspace** keys and the cursor 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.

Rename Element

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

Delete Element Tags

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

Remove All Markup

Removes all the XML markup inside the selected block of content and keeps only the text content.

Remove Text

Removes the text content of the selected block of content.

Attributes submenu

Contains predefined XML refactoring operations that pertain to attributes. Oxygen XML Editor plugin considers the editing context to get the names and namespaces of the element or attribute for which the contextual menu was invoked, and uses this information to preconfigure some of the parameter values for the selected refactoring operation.

Add/Change attribute

Allows you to change the value of an attribute or insert a new one.

Delete attribute

Allows you to remove one or more attributes.

Rename attribute

Allows you to rename an attribute.

Replace in attribute value

Allows you to search for a text fragment inside an attribute value and change the fragment to a new value.

Elements submenu

Contains predefined XML refactoring operations that pertain to elements. Oxygen XML Editor plugin considers the editing context to get the names and namespaces of the element or attribute for which the contextual menu was invoked, and uses this information to preconfigure some of the parameter values for the selected refactoring operation.

Delete element

Allows you to delete elements.

Delete element content

Allows you to delete the content of elements.

Insert element

Allows you to insert new elements.

Rename element

Allows you to rename elements.

Unwrap element

Allows you to remove the surrounding tags of elements, while keeping the content unchanged.

Wrap element

Allows you to surround elements with element tags.

Wrap element content

Allows you to surround the content of elements with element tags.

Fragments submenu

Contains predefined XML refactoring operations that pertain to XML fragments. Oxygen XML Editor plugin considers the editing context to get the names and namespaces of the element or attribute for which the contextual menu was invoked, and uses this information to preconfigure some of the parameter values for the selected refactoring operation.

Insert XML fragment

Allows you to insert an XML fragment.

Replace element content with XML fragment

Allows you to replace the content of elements with an XML fragment.

Replace element with XML fragment

Allows you to replace elements with an XML fragment.

Review submenu

This submenu includes the following actions:

 **Track Changes**

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

 **Accept Change(s)**

Accepts the tracked change located at the cursor position. If you select a part of a *delete* or *insert* change, 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 tracked change located at the cursor position. If you select a part of a *delete* or *insert* change, 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**

Opens a dialog box that allows you to add a comment to an existing tracked change. The comment appears in a callout box and a tooltip (when hovering over the change).

 **Highlight**

Enables the highlighting tool that allows you to mark text in your document.

Colors

Allows you to select the color for highlighting text.

Stop highlighting

Use this action to disable the highlighting tool.

Remove highlight(s)

Use this action to remove highlighting from the document.

 **Add Comment**

Inserts a comment at the cursor position. The comment appears in a callout box and a tooltip (when hovering over the change).

Remove comment

Removes a selected comment.

Manage Reviews

Opens the ***Review*** *view*.

Manage IDs submenu

This submenu is available for XML documents that have an associated DTD, XML Schema, or Relax NG schema. It includes the following actions:

Rename in

Renames the ID and all its occurrences. Selecting this action opens the **Rename XML ID** dialog box. This dialog box lets you insert the new ID value and choose the scope of the rename operation.

Search References

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

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 Occurrences in file

Searches for the occurrences of the ID in the current document.

Folding submenu

This submenu includes the following actions:

Toggle Fold

Toggles the state of the current fold.

Collapse Other Folds (Ctrl NumPad / (Meta NumPad / on OS X))

Folds all the elements except the current element.

Collapse Child Folds (Ctrl + NumPad- (Meta + NumPad- on OS X))

Folds the elements indented with one level inside the current element.

Expand Child Folds (Ctrl NumPad+ (Meta NumPad+ on OS X))

Unfolds all child elements of the currently selected element.

Expand All (Ctrl NumPad * (Meta NumPad * on OS X))

Unfolds all elements in the current document.

Inspect Styles

Opens the ***CSS Inspector*** *view* that allows you to examine the CSS rules that match the currently selected element.

Options

Opens the *Author mode options page*.

Document Type-Specific Contextual Menu Actions in Author Mode

Other *document type-specific* actions are available in the contextual menu of **Author** mode for the following document types (click the links to see the default actions that are available for each specific document types):

- ***DocBook4*** *Author Actions*
- ***DocBook5*** *Author Actions*
- ***DITA*** *Author Actions*
- ***DITA Map*** *Author Actions*
- ***XHTML*** *Author Actions*
- ***TEI ODD*** *Author Actions*

- *TEI P4 Author Actions*
- *TEI P5 Author Actions*
- *JATS Author Actions*

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 displayed:

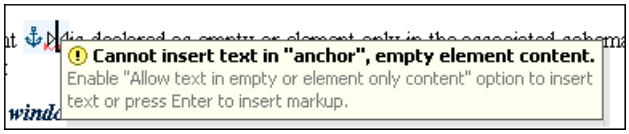


Figure 68: Editing in empty element warning

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 of the drag and drop sources are from the **Author** mode editor, a well-formed XML fragment is transferred. The section is balanced before dropping it by adding matching tags when needed.
- When the drag source is from the **Author** mode editor 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** mode editor is inserted without changes.

The font size of the current visual editor can be increased and decreased on the fly with the same actions as in the **Text** mode:

- **Ctrl NumPad+[plus sign] (Command NumPad+[plus sign] on OS X)** or **Ctrl NumPad-[minus sign] (Command NumPad-[minus sign] on OS X)** or **Ctrl Scroll Forward (Command Scroll Forward on OS X)** - Increases font size.
- **Ctrl NumPad-[minus sign] (Command NumPad-[minus sign] on OS X)** or **Ctrl -[minus sign] (Command - on OS X)** or **Ctrl Scroll Backwards (Command Scroll Backwards on OS X)** - Decreases font size.
- **Ctrl NumPad0 (Command NumPad0 on OS X)** or **Ctrl 0 (Command 0 on OS X)** - Restores font size to *the size specified in **Preferences***.

Removing the Text Content of the Current Element

You can remove the text content of the current element and keep only the markup by highlighting the appropriate block of content and use the **Remove Text** action that is available in the **Refactoring** submenu of the contextual menu. This is useful when the markup of an element must be preserved, for example a table structure but the text content must be replaced.

Editing the XML Markup

One of the most useful features in **Author** mode 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.

The content completion window offers the following types of actions:

- Inserting allowed elements for the current context according to the associated schema.
- 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.

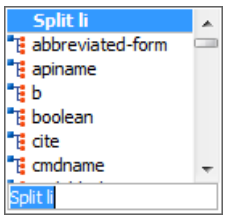


Figure 69: Content Completion Window

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 cursor 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 cursor position. Thus, if the cursor is positioned at the beginning or at the end of the element, the newly created sibling will be empty.

If the cursor is positioned inside a space preserved 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 cursor 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 cursor is located before the end position of the first element and **(Delete)** key is pressed.
- The cursor is located after the end position of the first element and **(Backspace)** key is pressed.
- The cursor is located before the start position of the second element and **(Delete)** key is pressed.
- The cursor 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 cursor is located before the start position of the element and **(Delete)** key is pressed.
- The cursor is located after the start position of the element and **(Backspace)** key is pressed.
- The cursor is located before the end position of the element and **(Delete)** key is pressed.
- The cursor 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 by highlighting the appropriate block of content and use the **Remove All Markup** action that is available in the **Refactoring** submenu of the contextual menu.

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 a marker representing the start or end tag of an element, the entire element is selected and numerous context specific actions are available in the various menus and *the contextual menu*.

Code Templates

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

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

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

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

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, JATS, and XHTML documents. Oxygen XML Editor plugin keeps the original text styling (such as bold, italics, etc.) and formatting (such as lists, tables, paragraphs, etc.), and helps you make the resulting document valid.

The following document types include support for *Smart Paste*:

- *DITA*
- *DocBook 4*
- *DocBook 5*
- *TEI 4*
- *TEI 5*
- *XHTML*
- *JATS*

The styles and general layout of the pasted content are transformed to the equivalent XML markup of the target document type.

You can disable the Smart Paste feature 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 Editor plugin 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*.

Reviewing Documents

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 **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*.

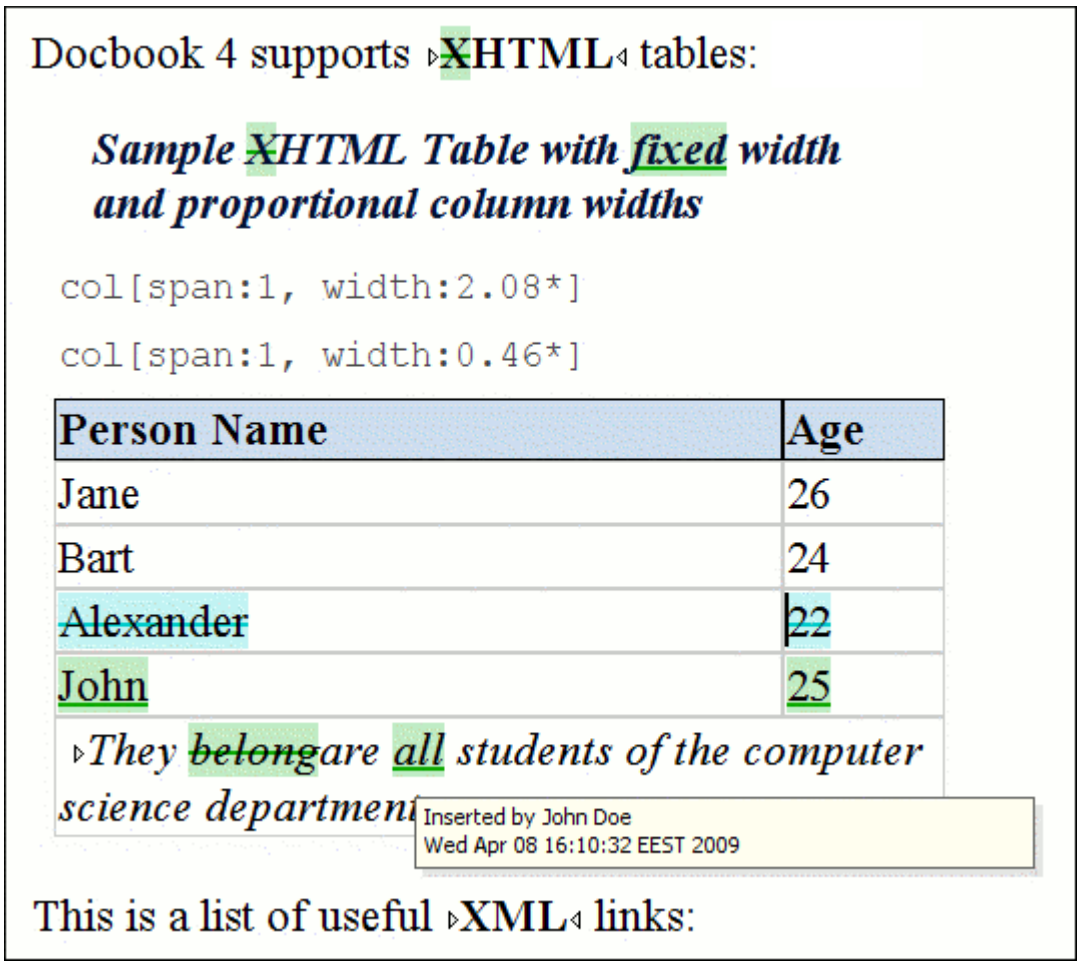


Figure 70: Change Tracking in Author Mode

When hovering over 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
- Inserting and deleting element tags
- Editing attributes
- Performing a **Split** operation
- Performing a **Surround with** operation

If the selection in the **Author** view contains tracked 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 instruction is an example of how an *insert* change is stored in a document:

```
<?oxy_insert_start author="John Doe"
timestamp="20090408T164459+0300"?>all<?oxy_insert_end?>
```

The following processing instruction is an example of how an *delete* change is stored in a document:

```
<?oxy_delete author="John Doe" timestamp="20090508T164459+0300"
content="belong"?>
```

Note: Tracked changes are also shown in the **Outline** view. Deleted content is rendered with a strike through.

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**, **Remove Comment**, and **Manage reviews**. They are available on the **Review** toolbar and in the **Review** submenu of the contextual menu in **Author** mode.

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 Editor plugin presents the tracked changes in DITA conrefs and XInclude fragments.

Managing Changes

You can review the changes that you or other authors have made and then accept or reject them using the **Track Changes** toolbar buttons  , or 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 tracked change located at the cursor position. If you select a part of a *delete* or *insert* change, 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 tracked change located at the cursor position. If you select a part of a *delete* or *insert* change, 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

Opens a dialog box that allows you to add a comment to an existing tracked change. The comment appears in a callout box and a tooltip (when hovering over the change).

Highlight

Enables or disables the ***Highlight** tool*. Use the **Highlight** drop-down menu to select a new color.

Add Comment

Inserts a comment at the cursor position. The comment appears in a callout box and a tooltip (when hovering over the change).

Edit comment

Opens the **Edit Comment** dialog box that allows you to edit the selected comment.

Remove comment

Removes a selected comment.

Manage Reviews

Opens the ***Review** view*.

Track Changes Visualization Modes Drop-Down Menu

This drop-down menu includes specialized actions that 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 the drop-down menu in the **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 behavior 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:

- *Inserted content*
- *Surrounded content*
- *Deleted characters*
- *Deleted content*
- *Copied content*
- *Pasted content*
- *Attribute changes*

Keeping Track of Inserted Content

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

- Making an insertion in a **Delete** change results in the change being split in two and the content is inserted without being marked as change.

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

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

- Making an insertion in a **Delete** change results in the change being split in two and the current inserted content appears marked as an INSERT.
- Making an insertion in an **Insert** change results in the following:
  - 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.

Keeping Track of Surrounded Content

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

- Making a surround in a **Delete** change results in nothing happening.
- Making a surround in an **Insert** change results in the following:
  - 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 results in the operation being marked as a surround change.

Keeping Track of Deleted Characters

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

- Deleting content in an existing **Delete** change results in nothing happening.
- Deleting content in an existing **Insert** change results in the content being deleted without being marked as a deletion and the INSERT change shrinks accordingly.
- Deleting in regular content results in a regular deletion.

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

- Deleting content in an existing **Delete** change results in the following:
  - 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 results in the following:
  - 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 results in the content being marked as **Delete** change by the current author.

Keeping Track of Deleted Content

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

- If the selection contains an entire **Delete** change, the change disappears and the content is deleted.
- If the selection intersects with a **Delete** change (starts or ends in one), it results in nothing happening.
- If the selection contains an entire **Insert** change, the change disappears and the content is deleted.
- If the selection intersects with an **Insert** change (starts or ends in one), the **Insert** change is shrunk and the content is deleted.

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

- If 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.
- If 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.
- If the selection contains an entire **Insert** change, the following is possible:
  - 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.

Keeping Track of Copied Content

When **Track Changes** is disabled and you copy content, the following is 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 is 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).

Keeping Track of Pasted Content

When **Track Changes** is disabled and you paste content, the following is 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 is 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.

Keeping Track of Attribute Changes

The **Track Changes** feature is able to keep 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 is 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 is 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`
Attribute Change	`<?oxy_attributes?>`	_	`id`, `type`, `oldValue`, `author`, `timestamp`

If 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.

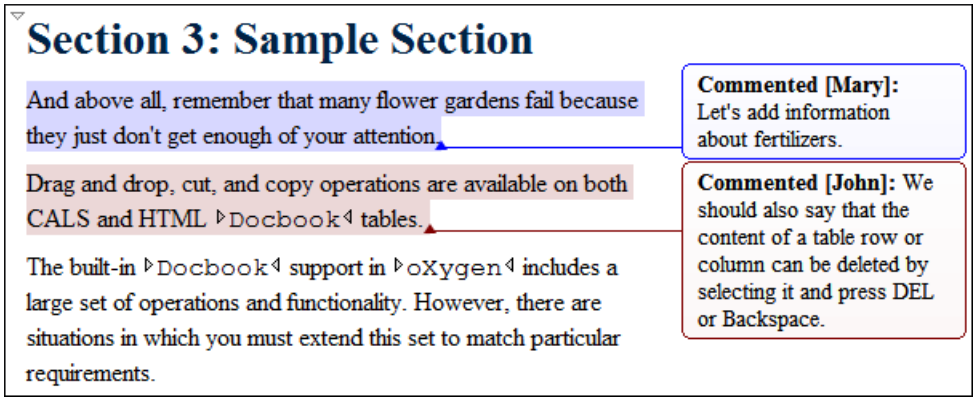


Figure 71: Comments in Author Mode

You can manage comments using the following actions that are available on the toolbar, in the **Review** submenu of the contextual menu:

Add Comment

Allows you to insert a comment at the cursor position or on a specific selection of content.

Edit Comment

Allows you to change an existing content.

Remove Comment(s)

Removes the comment at the cursor position or all comments found in the selected content.

Managing Highlights

Use the **Highlight** tool to mark fragments in your document using various colors. This is especially useful when you want to mark sections that needs additional editing or to draw the attention of others to particular content.

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** submenu. You can also choose the highlight **Colors** or choose to **Stop highlighting** from the same menus.

This tool allows you to do the following:

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

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

Note: Oxygen XML Editor plugin 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 a Document

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 (Meta A on OS X)** if 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)** (or select the action from the **Edit** > **Review** menu).
   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 Editor plugin 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 Editor plugin 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 box* 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 Editor plugin 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 Editor plugin 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 box* and go to **Editor** > **Edit Modes** > **Author** > **Review** > **Callouts**. Enable the following options:

- **Comments** - Oxygen XML Editor plugin displays comment callouts when you insert a comment. You can use two types of comments in Oxygen XML Editor plugin:
  - *Author Review Comments* - Comments that you associate with specific fragments of text.
  - *Callout 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 box*, go to **Editor** > **Edit Modes** > **Author** > **Review** > **Callouts**, and enable **Show review time**.

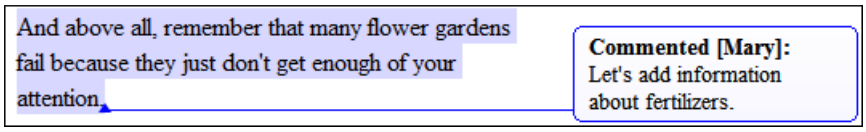


Figure 72: Comment Callouts

- **Track Changes deletions** - Oxygen XML Editor plugin 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 box*, go to **Editor** > **Edit Modes** > **Author** > **Review** > **Callouts**, and enable **Show review time** and **Show deleted content in callout**.

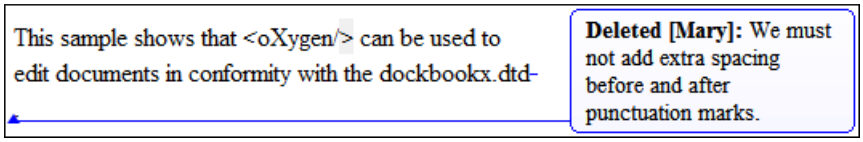


Figure 73: Deletion Callouts

- **Track Changes insertions** - Oxygen XML Editor plugin 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 box*, go to **Editor** > **Edit Modes** > **Author** > **Review** > **Callouts**, and enable **Show review time** and **Show inserted content in callout**.

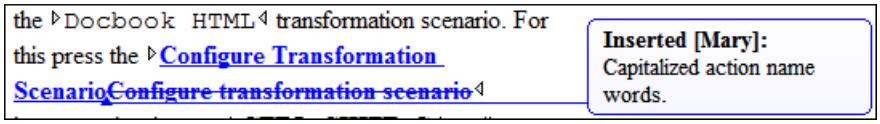


Figure 74: Insertion Callouts

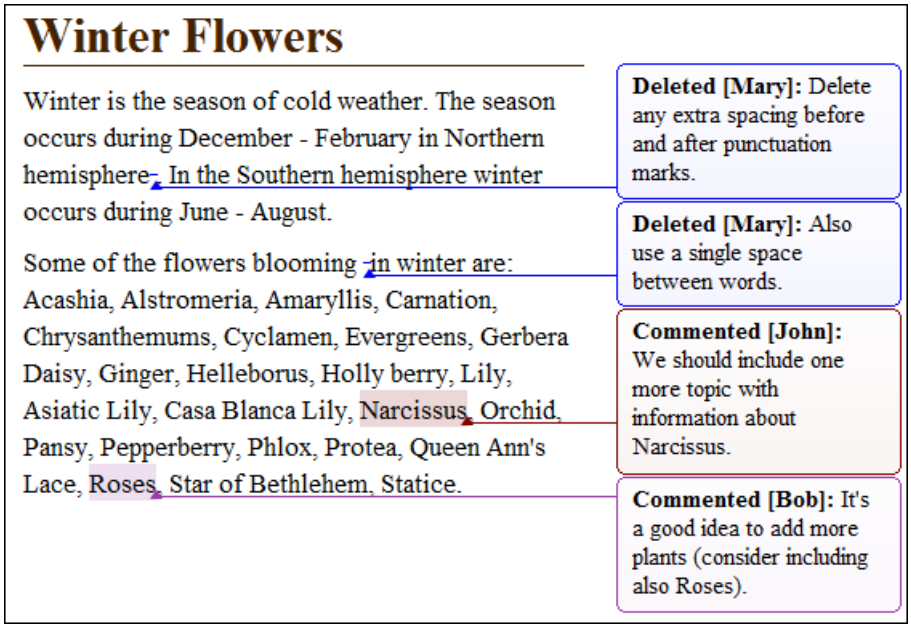


Figure 75: Multiple Authors Callouts

Note: Oxygen XML Editor plugin displays callouts only if **View All Changes/Comments** or **View Only Changes/Comments by** is selected. Oxygen XML Editor plugin 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 Editor plugin applies transparency to these callouts. When the cursor 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.

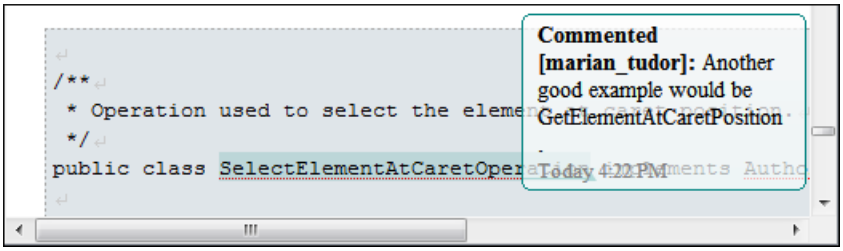


Figure 76: Transparent Callout

Note: Oxygen XML Editor plugin 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 **Review** toolbar.

- **Edit Reference** - If the fragment that contains a callout is a reference, use this option to go to the reference and edit the callout.
- **Callouts Options** - Select this option to open a *preferences page* where you can configure the callout options.

The following options are available in the contextual menu of a comment callout:

- **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** - If the fragment that contains a callout is a reference, use this option to go to the reference and edit the callout.
- **Callouts Options** - Select this option to open a *preferences page* where you can configure the callout options.

When you print a document from Oxygen XML Editor plugin, 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.

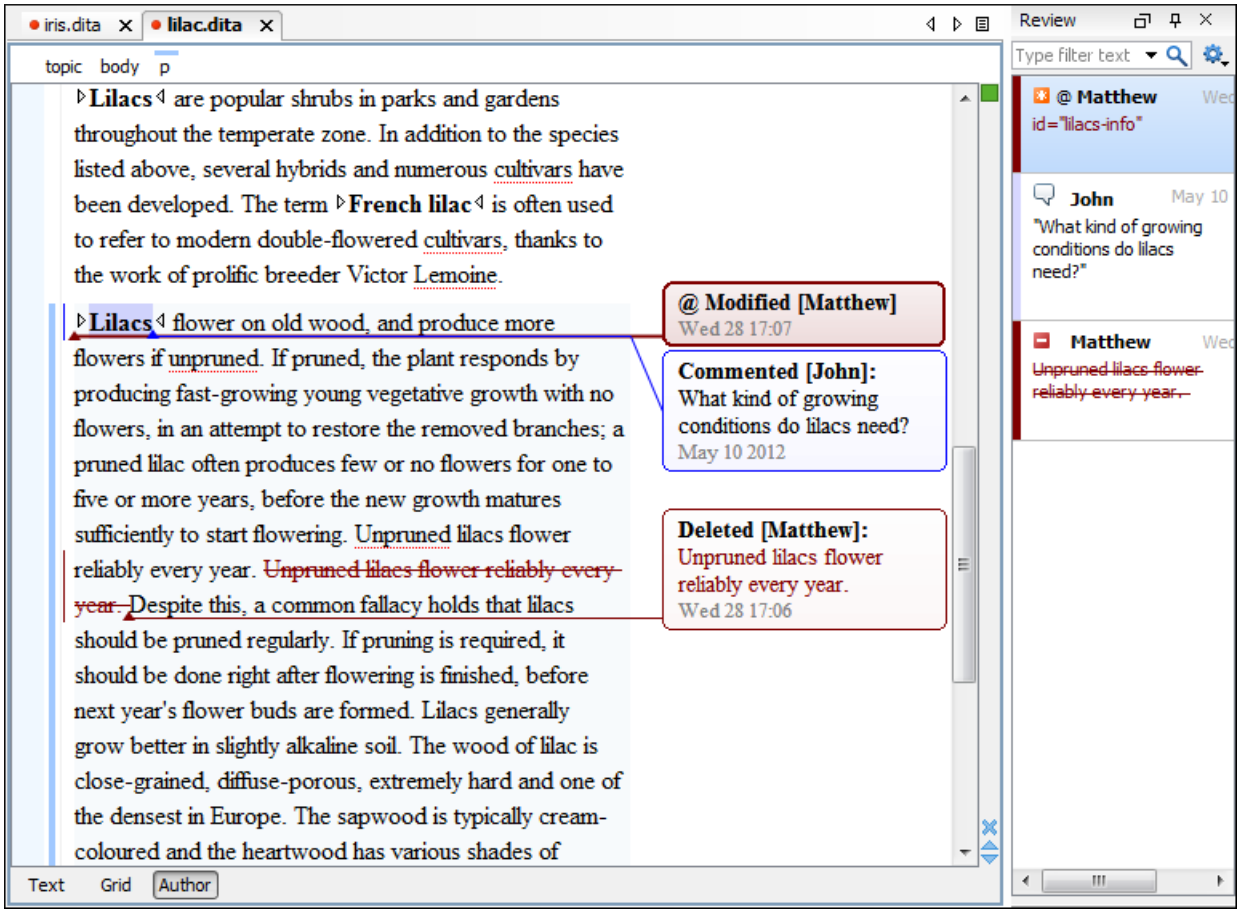


Figure 77: The Review View

To activate the **Review** view, do one of the following:

- Click the **Manage reviews** button on the **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 cursor 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 box.

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 box 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 Editor plugin comes with a preconfigured set of profiling attribute values for some of the most popular document types. These attributes can be redefined to match your specific needs. Also, you can define your own profiling attributes for a custom document type.

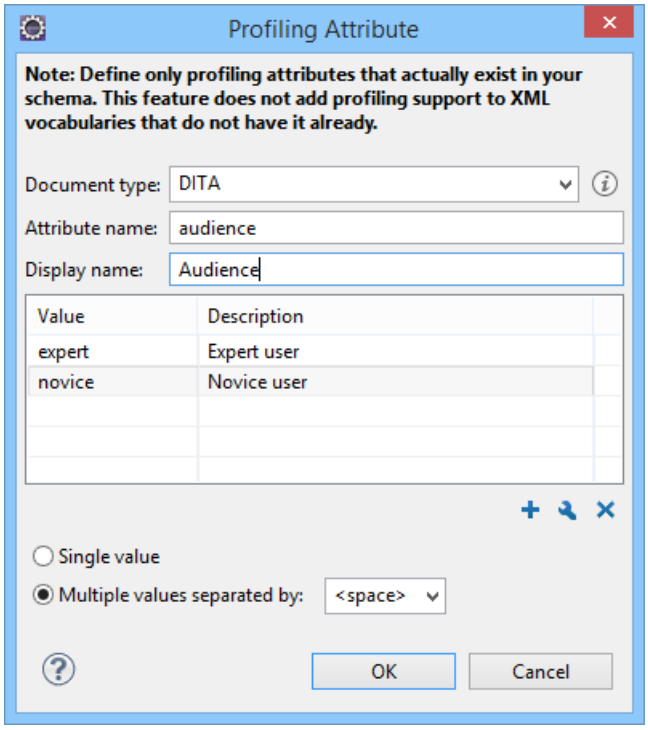
Create Profiling Attributes

Note: To ensure the validity of the document, the attribute must already be defined in the document DTD or schema before referencing it here.

To create custom profiling attributes for a specific document type, follow these steps:

1. *Open the **Preferences** dialog box* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text** .
2. In the **Profiling Attributes** area, press the **New** button.

   The **Profiling Attribute** dialog box is opened.



3. Fill-in the dialog box as follows:
   a) Choose the **Document type** on which the profiling attribute is applied. ***** and **?** can be 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) Specify the **Attribute name**.
   c) Specify a **Display name**. This field is optional, being used only as a descriptive rendering in profiling dialog boxes.
   d) Use the **New**, **Edit**, **Delete** buttons to add, edit, and delete possible values of the attribute. You can also specify and optional description for each attribute value.
   e) Choose whether the attribute accepts a **Single value** or **Multiple values separated by** a delimiter (*space*, *comma*, *semicolon*, or a custom one). A custom delimiter 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.

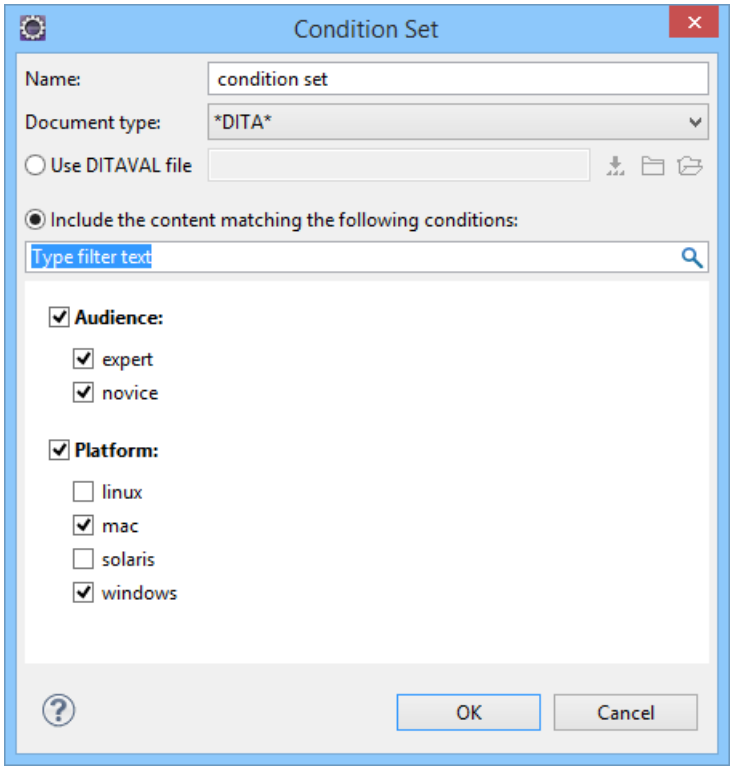
Create Profiling Condition Sets

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 box* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text**.
2. In the **Profiling Condition Sets** area, press the **New** button.

   The **Condition Set** dialog box is opened:

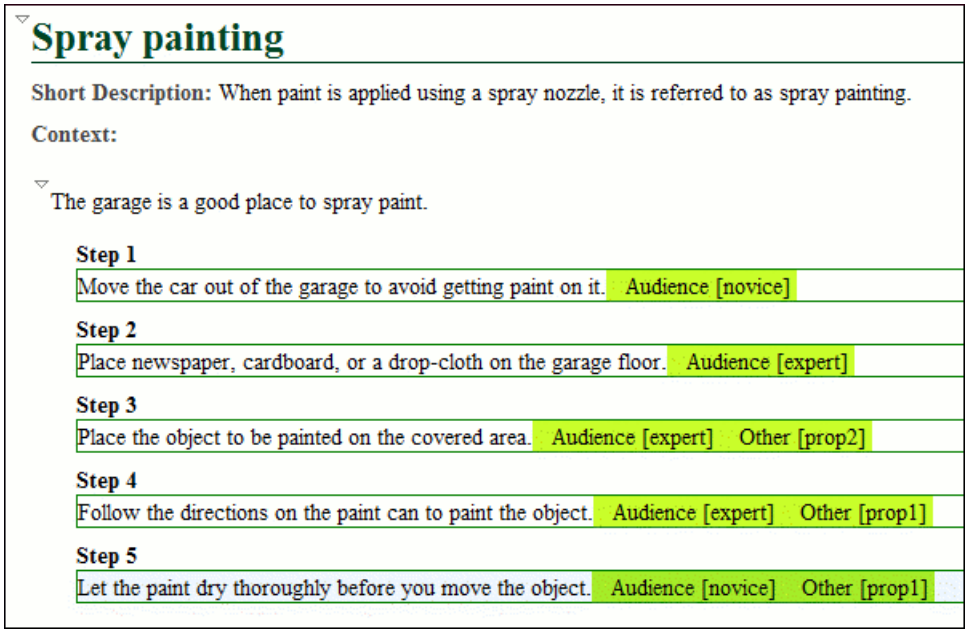


3. Fill-in the dialog box as follows:
   a) Type the condition set **Name**.
   b) Choose the **Document type** for which you have previously defined profiling attributes.
   c) If you want the Profiling Condition Set to reference a DITAVAL file, enable the **Use DITAVAL file** option and select the DITAVAL file from your disk.
   After choosing a document type, all profiling attributes and their possible values are listed in the central area of the dialog box.
   d) Define the combination of attribute values by selecting the appropriate checkboxes in the **Include the content matching the following conditions** section.

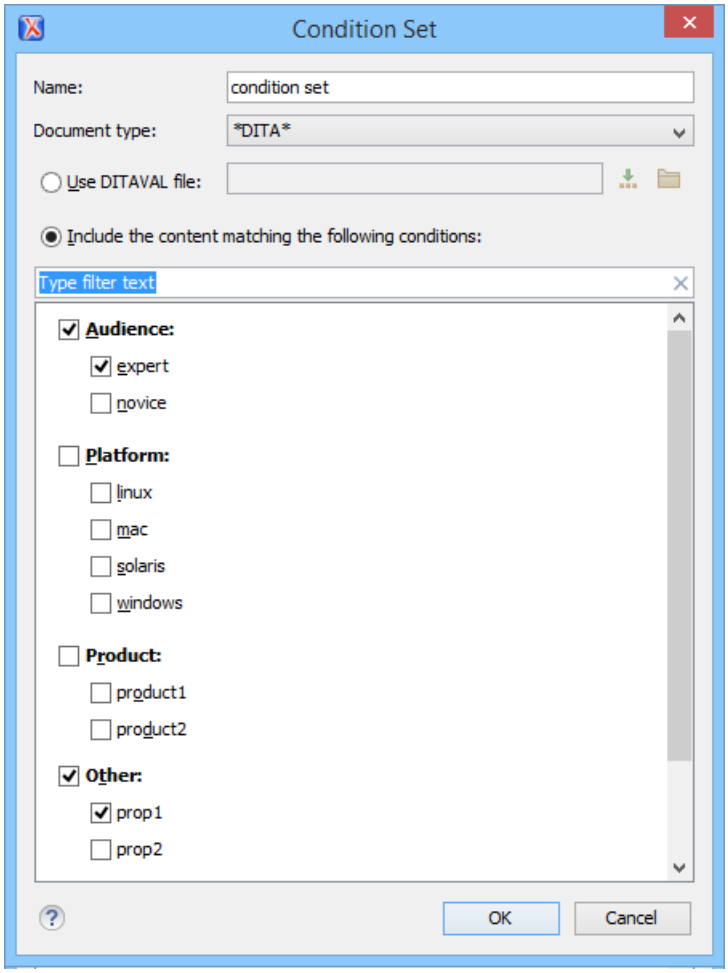
      If you have defined a lot of profiling attributes, you can use the **filter** text field to search for specific conditions.
4. Click **OK**.
5. Click **Apply** to save the condition set. All saved profiling condition sets are available in the ***Profiling / Conditional Text** toolbar drop-down menu*.

Apply Profiling Condition Sets

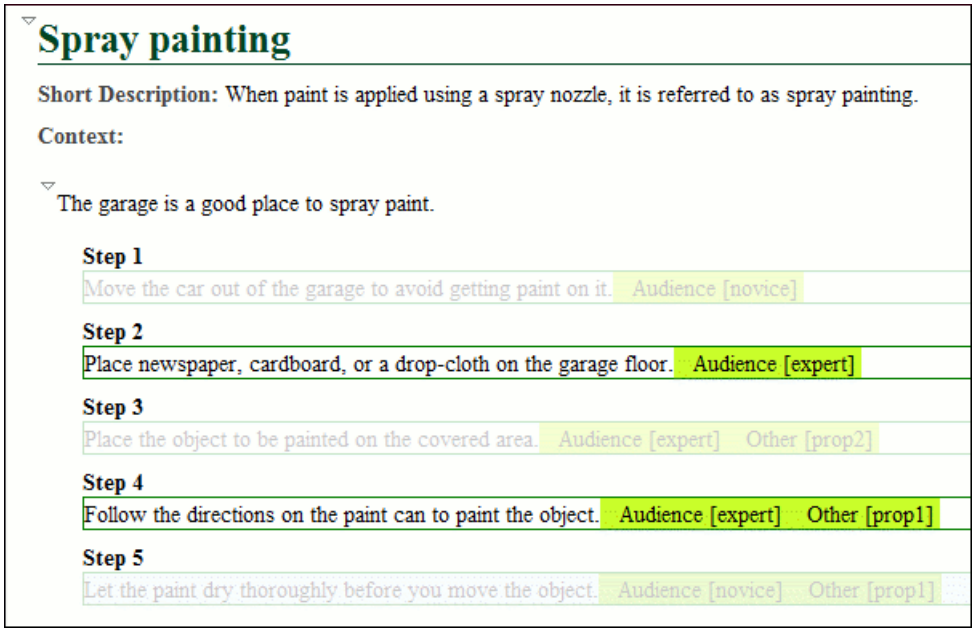
All defined Profiling Condition Sets are available as shortcuts in the Profiling / Conditional Text menu. Select a menu entry to apply the condition set. The filtered content is grayed-out in the **Author** mode, **Outline** view, and **DITA Maps Manager** view. An element is filtered-out when one of 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 the cursor position.

1. Invoke the **Edit Profiling Attributes** action from the contextual menu.
   The displayed dialog box shows all profiling attributes and their values, as defined in the framework (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 box, 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 box 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 Editor plugin 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 Toolbar 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 enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

Note: To remind you that document content is hidden, Oxygen XML Editor plugin displays labels showing the currently applied condition set. These labels are displayed in the **Author** mode 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 a condition set entry to activate it.

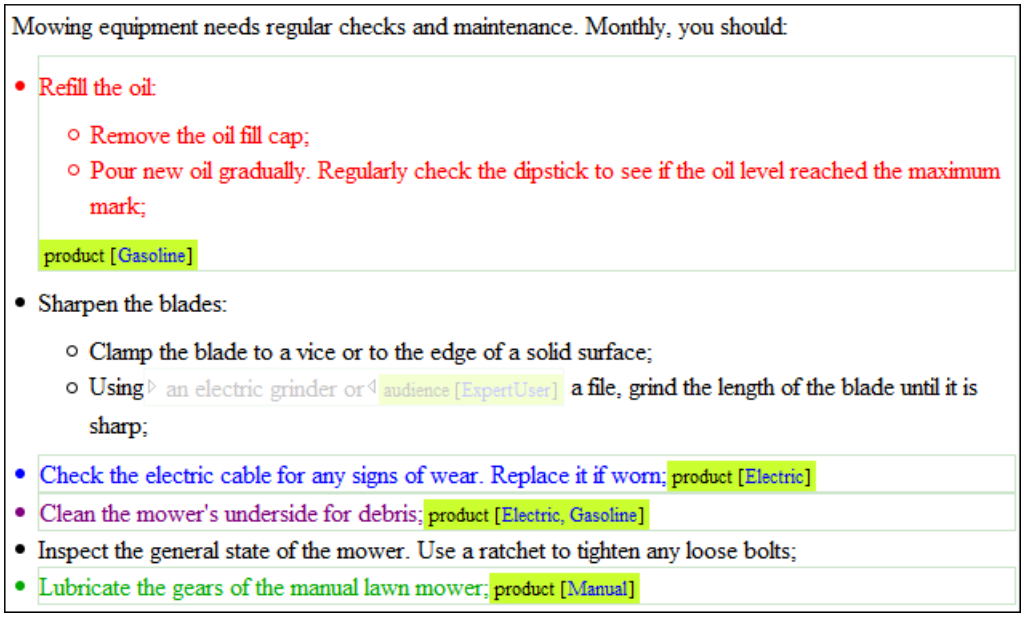
Profiling Settings

Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

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** mode editing area to mark profiled content so you can instantly spot different variants of the output.



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 drop-down menu*.
- Go to **Profiling Settings** from the ***Profiling / Conditional Text*** *toolbar drop-down 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** editing mode, 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 box and in the inline form control that allows you to quickly set the profiling attributes.

Table Layout and Operations

Oxygen XML Editor plugin 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.

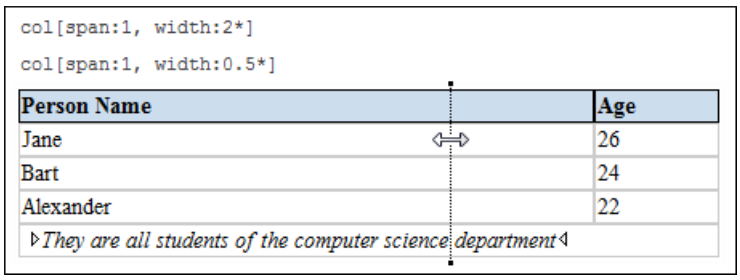


Figure 78: Resizing a Column in Oxygen XML Editor plugin Author Editor

- **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 if 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 Editor plugin 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 Editor plugin, 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 Editor plugin 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 Editor plugin 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.

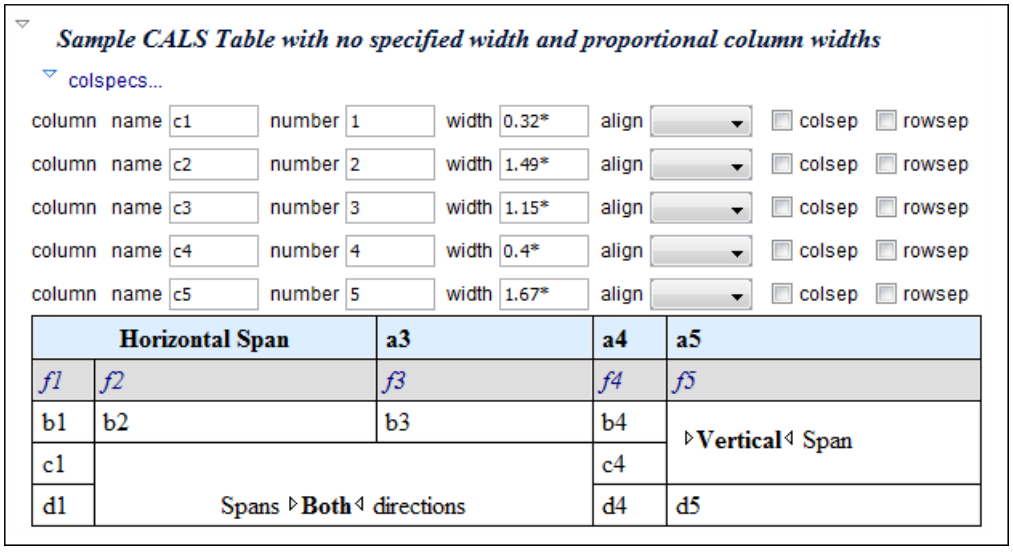


Figure 79: CALS Table in DocBook

Pasted Tables

Tables that are pasted into a DocBook file are automatically converted to the CALS model. If you want to overwrite this behavior and instruct Oxygen XML Editor plugin to convert them to HTML tables, set the `docbook.html.table` parameter to `1`. You can find this parameter in the following stylesheet:

- `[OXYGEN_DIR]/frameworks/docbook/resources/xhtml2db5Driver.xsl` for DocBook 5
- `[OXYGEN_DIR]/frameworks/docbook/resources/xhtml2db4Driver.xsl` for DocBook 4

Editing Table Component Properties in DocBook

To customize the look of a table, place the cursor anywhere in a table and invoke the **Table Properties (Ctrl + T (Command + T on OS X))** action from the **Table** submenu of the contextual menu or **DocBook** menu.

The **Table properties** dialog box 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 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** - The horizontal alignment, row and column separators, and the frame.
- **Row** - The row type, vertical alignment, and row separator.

- **Column** - The horizontal alignment, and column and row separators.
- **Cell** - The horizontal alignment, vertical alignment, and column and row separators.

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

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

XHTML Table Layout

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

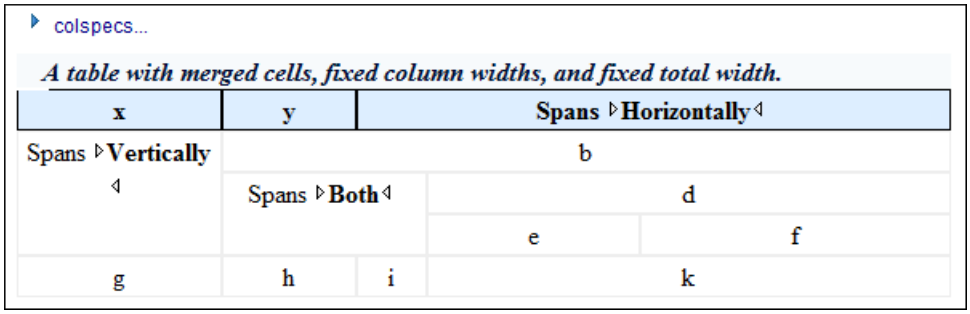


Figure 80: HTML table

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*`.

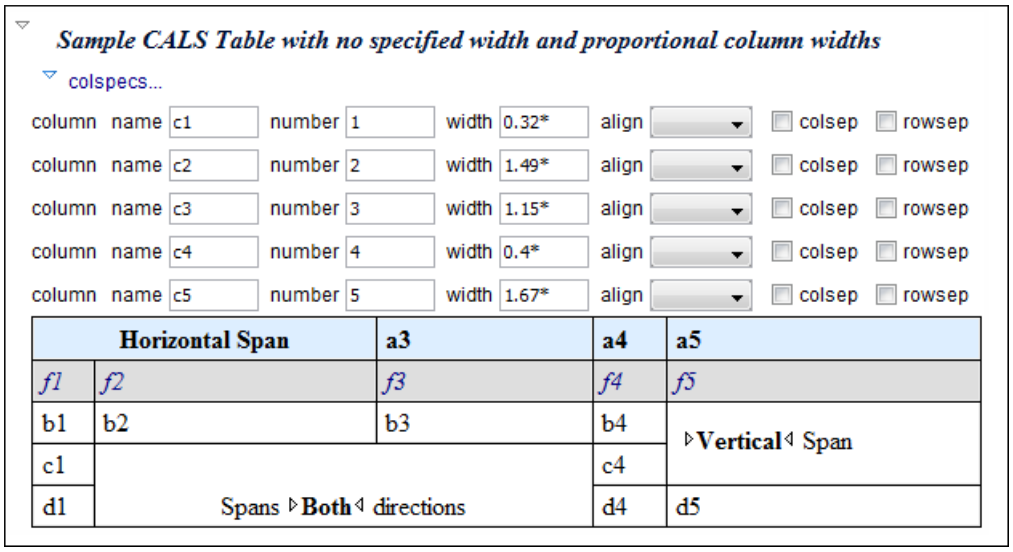


Figure 81: CALS table in DITA

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 82: 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 in DITA

To customize the look of a table, place the cursor anywhere in a table and invoke the **Table Properties (Ctrl + T (Command + T on OS X))** action from the **Table** submenu of the contextual menu or **DITA** menu.

The **Table properties** dialog box 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 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** - The horizontal alignment, row and column separators, and the frame.
- **Row** - The row type, vertical alignment, and row separator.
- **Column** - The horizontal alignment, and column and row separators.
- **Cell** - The horizontal alignment, vertical alignment, and column and row separators.

For a **Simple** table, you can customize the following:

- **Table** - The frame attribute.
- **Row** - The row type.

Sorting Content in Tables and List Items

Oxygen XML Editor plugin 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 list items*.

Sorting a Table

To sort rows in a table, select the entire table (or specific rows) and use the **Sort** action from the main toolbar or the contextual menu. This opens the **Sort** dialog box.

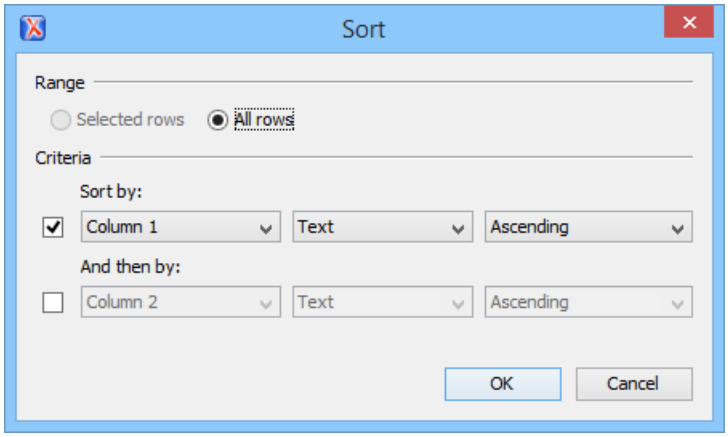


Figure 83: The "Sort" Dialog Box

This dialog box sets the range that is sorted and the sorting criteria. The range is automatically selected 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 **Criteria** section specifies the sorting criteria (a maximum of three sorting criteria are available), defined by the following:

- A name, which is collected from the column heading.
- The type of the information that is sorted. You can choose between the following:
  - **Text** - Alphanumeric characters.
  - **Numeric** - Regular integer or floating point numbers are accepted.
  - **Date** - Default date and time formats from the local OS are accepted (such as *short*, *medium*, *long*, *full*, *xs:date*, and *xs:dateTime*).
- The sorting direction (either *ascending* or *descending*).

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

After you finish configuring the options in the **Sort** dialog box, click **OK** to complete the sorting operation. If you want to revert to the initial order of your content, press **Ctrl Z (Meta Z on OS X)** on your keyboard.

Note: The sorting support takes the value of the `xml:lang` attribute into account 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. This opens the **Sort** dialog box.

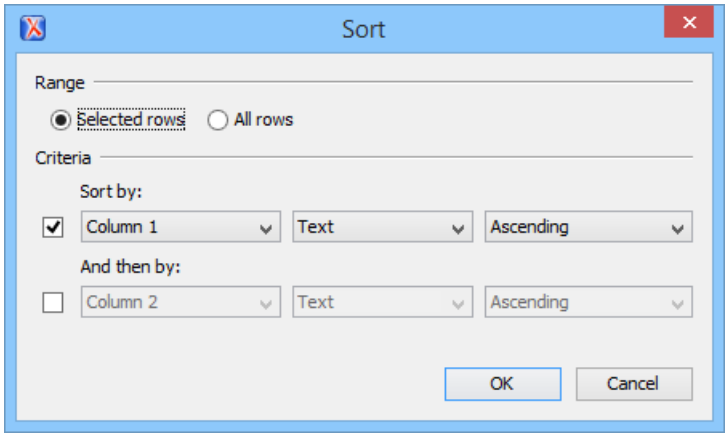


Figure 84: Sort Selected Rows

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

The **Sort** dialog box also allows you to apply the sorting operation to the entire table, using the **All rows** option.

The **Criteria** section specifies the sorting criteria (a maximum of three sorting criteria are available), defined by the following:

- A name, which is collected from the column heading.
- The type of the information that is sorted. You can choose between the following:
  - **Text** - Alphanumeric characters.
  - **Numeric** - Regular integer or floating point numbers are accepted.
  - **Date** - Default date and time formats from the local OS are accepted (such as *short*, *medium*, *long*, *full*, *xs:date*, and *xs:dateTime*).
- The sorting direction (either *ascending* or *descending*).

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

After you finish configuring the options in the **Sort** dialog box, click **OK** to complete the sorting operation. If you want to revert to the initial order of your content, press **Ctrl Z (Meta Z on OS X)** on your keyboard.

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

Sort Using Multiple Criteria

You can also sort an entire table or a selection of its rows based on multiple sorting criteria. To do so, enable the rest of boxes in the **Criteria** section of the **Sort** dialog box, configure the applicable items, and click **OK** to complete the sorting operation.

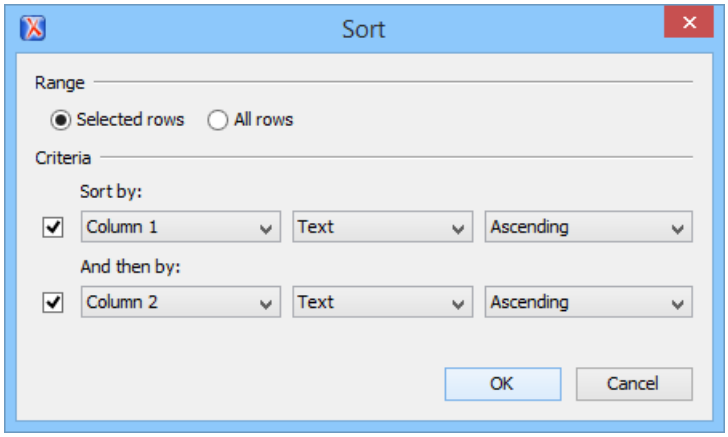


Figure 85: Sorting Based on Multiple Criteria

Sorting a Table that Contains Merged Cells

If 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 box keeps the **All rows** option disabled even if you perform the sorting operation over a selection of rows.

Sorting List Items

A sorting operation can be performed on various types of lists and list items. Oxygen XML Editor plugin provides support for sorting the following types of lists:

- Ordered list (`ol`)
- Unordered list (`ul`)
- Parameter list (`parml`)
- Simple list (`sl`)
- Required conditions (`reqconds`)
- Supplies list (`supplyli`)
- Spare parts list (`sparesli`)
- Safety conditions (`safety`)
- Definitions list (`dl`)

The sorting mechanism works on an entire list or on a selection of list items. To sort items in a list, select the items or list and use the **Sort** action from the main toolbar or the contextual menu. This opens the **Sort** dialog box.

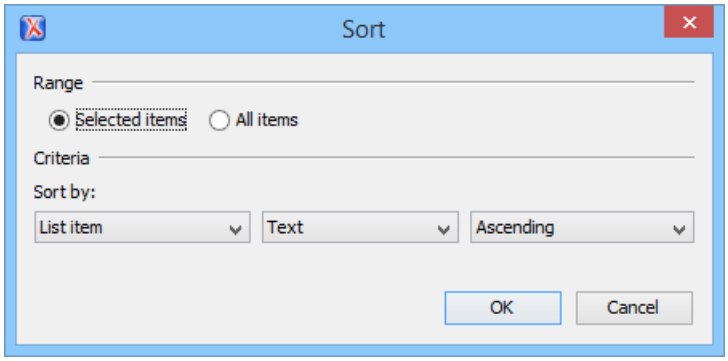


Figure 86: Sorting List Items

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

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

The **Criteria** section specifies the sorting criteria, defined by the following:

- The name of the type of item being sorted.
- The type of the information that is sorted. You can choose between the following:
  - **Text** - Alphanumeric characters.
  - **Numeric** - Regular integer or floating point numbers are accepted.
  - **Date** - Default date and time formats from the local OS are accepted (such as *short*, *medium*, *long*, *full*, *xs:date*, and *xs:dateTime*).
- The sorting direction (either *ascending* or *descending*).

After you finish configuring the options in the **Sort** dialog box, click **OK** to complete the sorting operation. If you want to revert to the initial order of your content, press **Ctrl Z (Meta Z on OS X)** on your keyboard.

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

Image Rendering

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

Table 4: 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 Editor plugin **Author** mode displays a warning message that contains the reason why this is happening. Possible causes include the following:

- The image is too large. Enable the *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 both the width and height of an image proportionally, use the *scale* attribute.

Note: As a Java application, Oxygen XML Editor plugin uses the *Java Advanced Imaging* API that provides a pluggable support for new image types. If you have an *ImageIO* library that supports additional image formats, just copy this library to the `[OXYGEN_DIR]/lib` directory.

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

Follow this procedure:

1. Start Oxygen XML Editor plugin and open the **Help** > **About** dialog box. Click the **Installation Details** button, go to the **Configuration** tab, and look for the *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).

   Note that the JAI API is not the same thing as JAI Image I/O. Make sure you have installed the latter.
3. Run 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.

Customize Oxygen XML Editor plugin to Render CGM Images (Experimental Support)

Oxygen XML Editor plugin provides experimental support for CGM 1.0 images.

Attention: Image hotspots are not supported.

Since this 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 the `[OXYGEN_DIR]\lib` directory.
3. Open `OXYGEN_PLUGIN_DIR/META-INF/MANIFEST.MF` and add a reference to the JAR library in the `Bundle-ClassPath` entry.
4. Restart Eclipse in clean mode (edit the shortcut you use to start Eclipse and add `-clean` as the first argument.)

Customize Oxygen XML Editor plugin to Render PDF Images (Experimental Support)

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

To enable the rendering of PDF images in **Author** mode, follow this procedure:

1. Go to *http://pdfbox.apache.org/downloads.html* and download the pre-built PDFBox standalone binary JAR files `pdfbox-1.8.9.jar`, `fontbox-1.8.9.jar`, and `jempbox-1.8.9.jar`.
2. Copy the downloaded JAR libraries to the `[OXYGEN_DIR]\lib` directory.
3. Open `OXYGEN_PLUGIN_DIR/META-INF/MANIFEST.MF` and add a reference to the JAR libraries in the `Bundle-ClassPath` entry.
4. Restart Eclipse in clean mode (edit the shortcut you use to start Eclipse and add `-clean` as the first argument.)

Customize Oxygen XML Editor plugin to Render PSD Images

Oxygen XML Editor plugin provides support for rendering PSD (Adobe Photoshop) images.

To enable the rendering of PSD images in **Author** mode, follow this procedure:

1. Download the following JAR files:
   - *http://search.maven.org/remotecontent?filepath=com/twelvemonkeys/common/common-lang/3.1.0/common-lang-3.1.0.jar*
   - *http://search.maven.org/remotecontent?filepath=com/twelvemonkeys/common/common-io/3.1.0/common-io-3.1.0.jar*
   - *http://search.maven.org/remotecontent?filepath=com/twelvemonkeys/common/common-image/3.1.0/common-image-3.1.0.jar*
   - *http://search.maven.org/remotecontent?filepath=com/twelvemonkeys/imageio/imageio-core/3.1.0/imageio-core-3.1.0.jar*
   - *http://search.maven.org/remotecontent?filepath=com/twelvemonkeys/imageio/imageio-metadata/3.1.0/imageio-metadata-3.1.0.jar*
   - *http://search.maven.org/remotecontent?filepath=com/twelvemonkeys/imageio/imageio-psd/3.1.0/imageio-psd-3.1.0.jar*
2. Copy the downloaded JAR libraries to the `[OXYGEN_DIR]\lib` directory.
3. Open `OXYGEN_PLUGIN_DIR/META-INF/MANIFEST.MF` and add a reference to the JAR libraries in the `Bundle-ClassPath` entry.
4. Restart Eclipse in clean mode (edit the shortcut you use to start Eclipse and add `-clean` as the first argument.)

Customize Oxygen XML Editor plugin to Render EPS and AI Images

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

- The EPS or AI image does not include the preview picture. Oxygen XML Editor plugin 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 Editor plugin 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** action from the toolbar and choose the image file you want to insert. Oxygen XML Editor plugin 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 the `file:/C:/project/xml/doc1.xml` document, Oxygen XML Editor plugin 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** 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** editing mode. 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 Editor plugin will automatically update the reference to the new image.

- Copy the image from another application (such as an image editor) and paste it into your document. Oxygen XML Editor plugin prompts you to first save it. After saving the image, a reference to that file path is inserted at the paste 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.

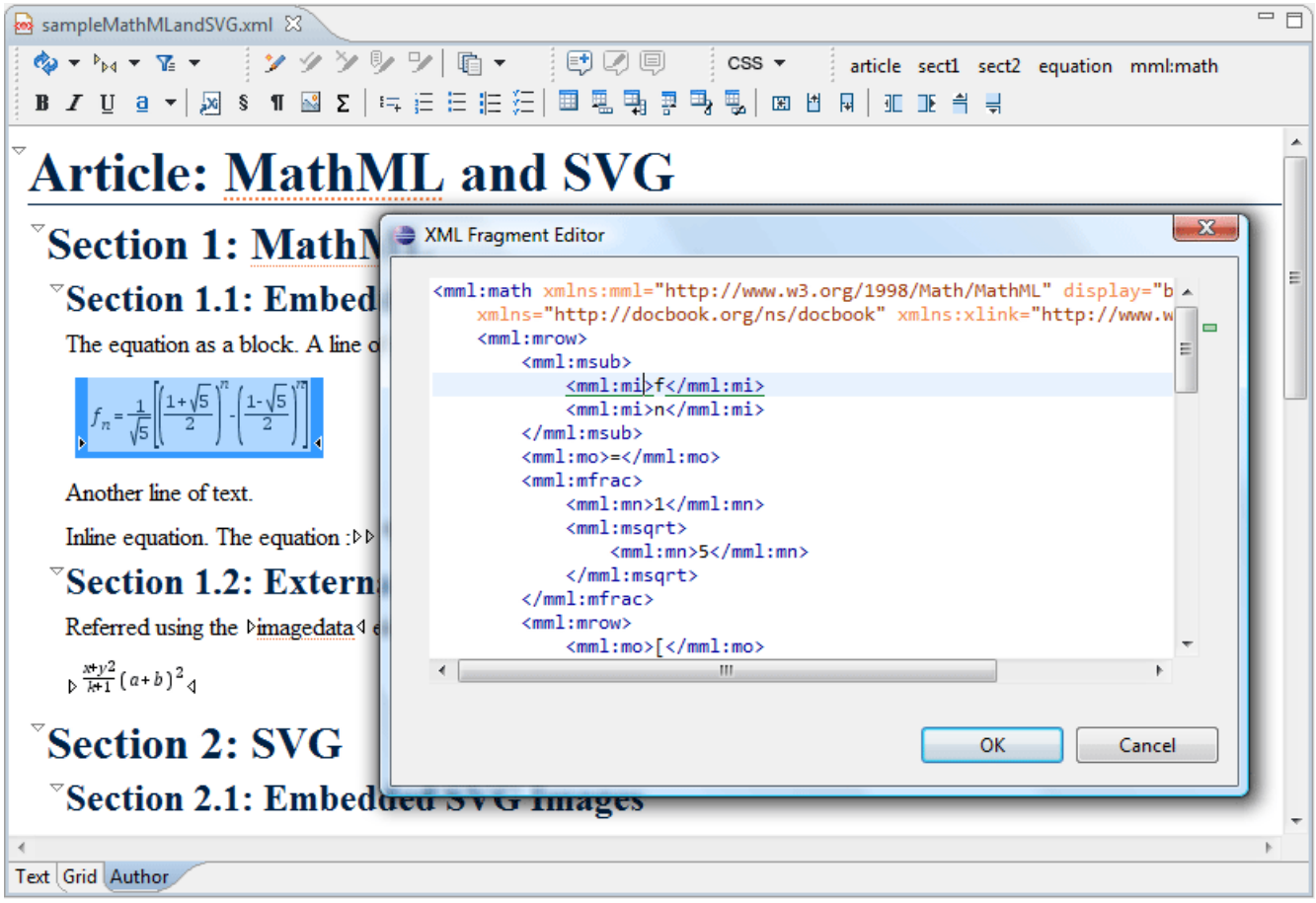


Figure 87: The default *MathML* editor

Configure the *MathFlow* Editor

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

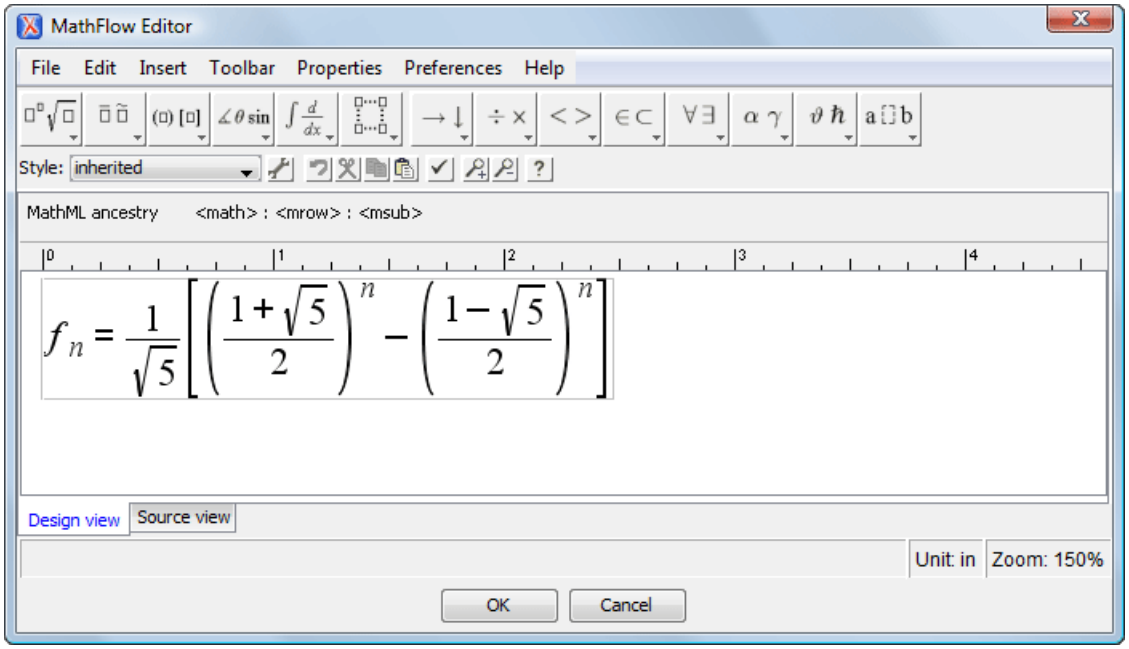


Figure 88: The default MathML editor

1. Install *MathFlow* Components (*MathFlow* SDK).
2. On Windows make sure there is a copy of the *FLEXlm* DLL, which 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***.

MathML Equations in HTML Output

Currently, only **Firefox** can render **MathML** equations embedded in the **HTML** code. *MathJax* is a solution to properly view MathML equations embedded in **HTML** content in a variety of browsers.

If you have DocBook or DITA content that has embedded **MathML** equations and you want to properly view the equations in published HTML output types (WebHelp, CHM, EPUB, etc.), you need to add a reference to the MathJax script in the **head** element of all HTML files that have the equation embedded.

For example:

```
<script type="text/javascript" src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
 </script>
```

For DITA documents, you can also 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 such as IE, Chrome, and Firefox.

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.

To refresh the content of the referenced resources you can use the **Refresh references** action. However, this action will not refresh the expanded external entities, for which you will need to use the **Reload** action.

Editing Attributes in Author Mode In-Place

You can easily edit attributes in **Author** mode by using the ***Attributes View*** and Oxygen XML Editor plugin also allows you to edit attribute and element values in-place, directly in the **Author** mode, using an in-place attribute editor.

In-place Attributes Editor

Oxygen XML Editor plugin includes an in-place attributes editor in **Author** mode. To edit the attributes of an XML element in-place, do one of the following:

- Select an element or place the cursor 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**, or **Inline Tags**.

This opens an in-place attributes editor that contains the same content as the **Attributes** view. By default, this editor presents the **Name** and **Value** fields, with the list of all the possible attributes collapsed.

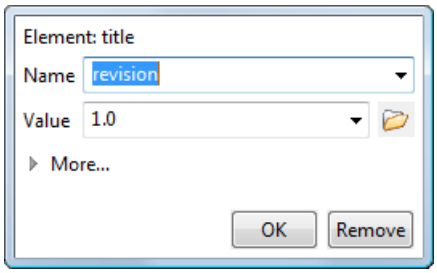


Figure 89: In-place Attributes Editor

Name Combo Box

Use this combo box to select an attribute. The drop-down list displays the list of possible attributes allowed by the schema of the document, as in the **Attributes** view.

Value Combo Box

Use this combo box to add, edit, or select the value of an attribute. If the selected attribute has predefined values in the schema, the drop-down list displays those possible values.

If you click **More** while in the collapsed version, it is expanded to the full version of the in-place attribute editor.

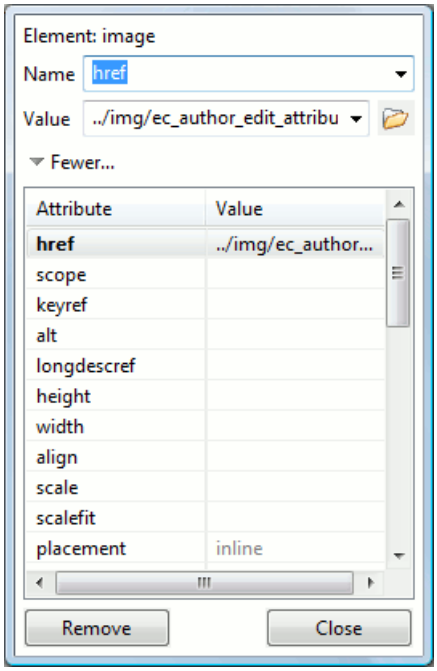


Figure 90: In-place Attributes Editor (Full Version)

The full version includes a table grid, similar to the **Atributes** view, that presents all the attributes for the selected element.

Generating IDs for Elements in Author Mode

Oxygen XML Editor plugin allows you to manually assign or edit values of `id` attributes in **Author** mode by using the ***Attributes View*** or an *in-place attribute editor*. Oxygen XML Editor plugin also includes mechanisms to generate ID values for elements, either on-request or automatically, in DITA, DocBook, or TEI documents.

Generate IDs On-Request

You can generate ID values for specific elements on-request. To do so, select the element for which you want to generate an ID (or place the cursor inside the element) and select the **Generate IDs** action from the contextual menu or the framework-specific menu (**DITA**, **DocBook**, or **TEI**). This action generates a unique ID for the current element. If you invoke the action on a block of selected content, the action will generate IDs for all top-level elements and elements that are listed in the ***ID Options*** *dialog box* that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

Automatically Generate IDs

Oxygen XML Editor plugin includes an option to automatically add unique ID values to certain elements when they are created in **Author** mode. The **Auto generate IDs for elements** option can be found in the ***ID Options*** *dialog box* that is displayed when you select the **ID Options** action from the framework-specific menu (**DITA**, **DocBook**, or **TEI**). If enabled, Oxygen XML Editor plugin automatically generates unique ID values for elements that are listed in this dialog box. You can use this dialog box to customize the format of the ID values and choose which elements will have their ID values automatically generated (for example, you can customize the list of elements to include those that you most often need to identify).

ID Options Dialog Box

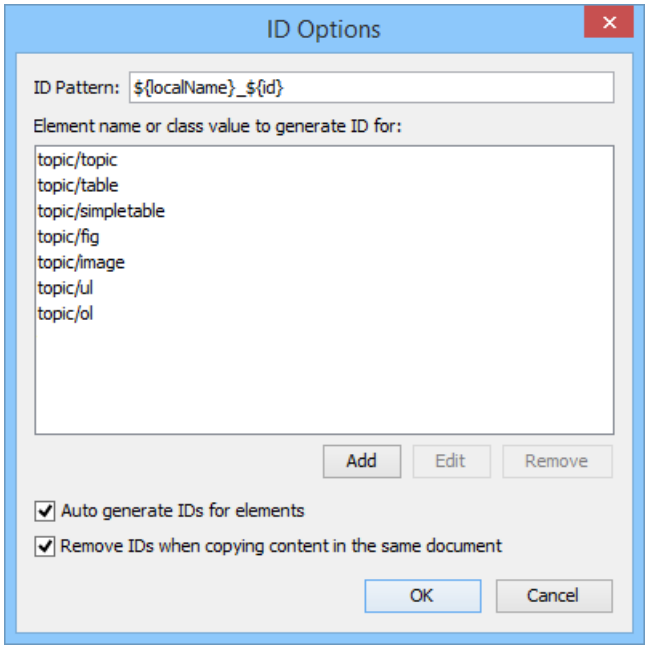


Figure 91: The ID Options Dialog Box

The **ID Options** dialog box allows you to configure the following options in regards to generating ID values:

ID Pattern

The pattern for the ID values that will be generated. This text field can be customized using constant strings and most of the supported *Editor Variables* on page 594.

Element name or class value to generate ID for

The elements for which ID values will be generated, specified using class attribute values. To customize the list, use the **Add**, **Edit**, or **Remove** buttons.

Auto generate IDs for elements

If enabled, Oxygen XML Editor plugin will automatically generate unique IDs for the elements listed in this dialog box when they are created in **Author** mode.

Remove IDs when copying content in the same document

Allows you to control whether or not pasted elements that are listed in this dialog box should retain their existing IDs when copying content in the same document. To retain the element IDs, disable this option.

Duplicating Elements with Existing IDs

If you duplicate elements with existing IDs (for example, through copy/paste or drag/drop actions), all IDs are removed at the resolution of the operation. However, you can use the options in the **ID Options** dialog box to change this behavior. The options in this dialog box affect duplicated elements with existing IDs in the following ways:

Note: Only the elements listed in this dialog box are affected by these options. Therefore, if you want to use these options to preserve IDs or generate new ones, you must first add the elements to be duplicated to the list in this dialog box.

- If the **Auto generate IDs for elements** option is enabled and you duplicate elements with existing IDs, Oxygen XML Editor plugin assigns new, unique ID values to the duplicates.
- If the **Auto generate IDs for elements** option is disabled and you duplicate elements with existing IDs, the ID values are removed from the duplicates. However, when elements are duplicated in the same document, this option has no effect and IDs are preserved if the **Remove IDs when copying content in the same document** option is disabled.

- If the **Remove IDs when copying content in the same document** option is enabled, the ID values are removed from elements that are duplicated in the same document. However, enabling this option has no effect if the **Auto generate IDs for elements** option is enabled.
- If the **Remove IDs when copying content in the same document** option is disabled, the ID values are preserved when elements are duplicated in the same document. This option has no affect on elements that are duplicated in other documents.

Using Form Controls in Author Mode

You can use form controls in **Author** mode in a variety of ways to make it easier to capture, organize, and edit content. Oxygen XML Editor plugin includes *built-in form controls* that can be used by content authors in **Author** mode. The types of built-in form controls that you can use include the following:

- *Text Field* - A graphical user interface box that allows you to enter a single line of text.
- *Combo Box* - A graphical user interface object that can be a drop-down menu or a combination of a drop-down menu and a single-line text field.
- *Check Box* - A graphical user interface box that you can click to select or deselect a value.
- *Pop-up* - A contextual menu that provides quick access to various actions.
- *Button* - A graphical user interface object that performs a specific action.
- *Button Group* - A graphical user interface group of buttons (such as radio buttons) that perform specific actions.
- *Text Area* - A box that allows you to enter multiple lines of text.
- *URL Chooser* - A dialog box that allows you to select the location of local or remote resources.
- *Date Picker* - A form control object that allows you to select a date in a specified format.
- *HTML Content* - A graphical user interface box that is used for rendering HTML content.

You can also *implement custom form controls* for more specific needs.

The following image is an example of how form controls can be used by content authors in **Author** mode. It includes several button form controls, a combo box, and a text field. The  icon is a button form control that is assigned a specific action that changes the layout to an editing mode. The **[+]** and **[-]** icons are also button form controls that are assigned specific actions to add or delete records from the document. The *Direct manager* row includes a combo box form control that is both a drop-down menu and an editable text field, while the *Homepage* row includes a simple editable text field form control.

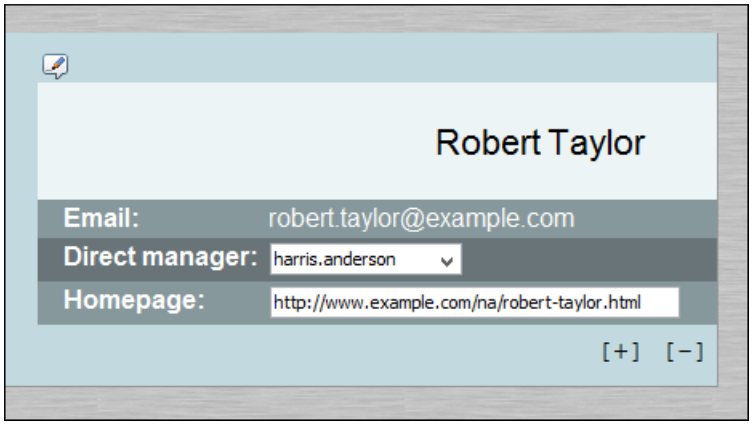


Figure 92: Example of Form Controls in Author Mode

You can use your imagination to envision the multitude of ways that you can use form controls to make the editing experience for content authors easier and more efficient. As a working example, a bundled *samples* project (located in the `samples` folder inside the Oxygen XML Editor plugin installation directory) contains a file called `personal.xml` that contains form controls. You can use this file, along with its corresponding `personal.css` file (form controls are defined in the CSS) to experiment with an example of how form controls can be implemented in **Author** mode.

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 Editor plugin.

Supported Schema Types for XML Documents

The supported schema types are:

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

Setting a Default Schema

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

- The *validation scenario* associated with the document.
- The validation scenario associated with the document type (if defined).
- The document schema declaration.

  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.

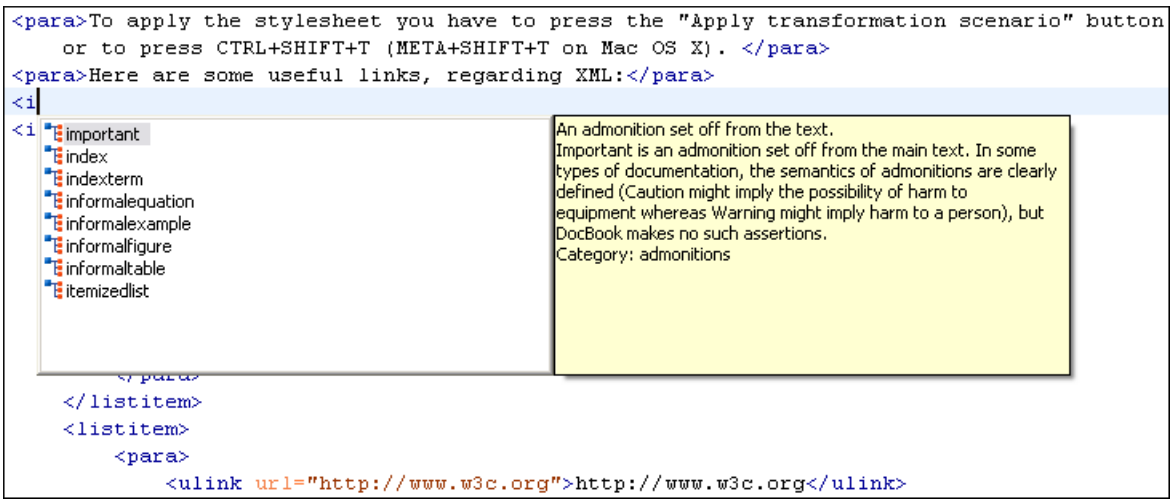


Figure 93: Content Completion Driven by DocBook DTD

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.

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

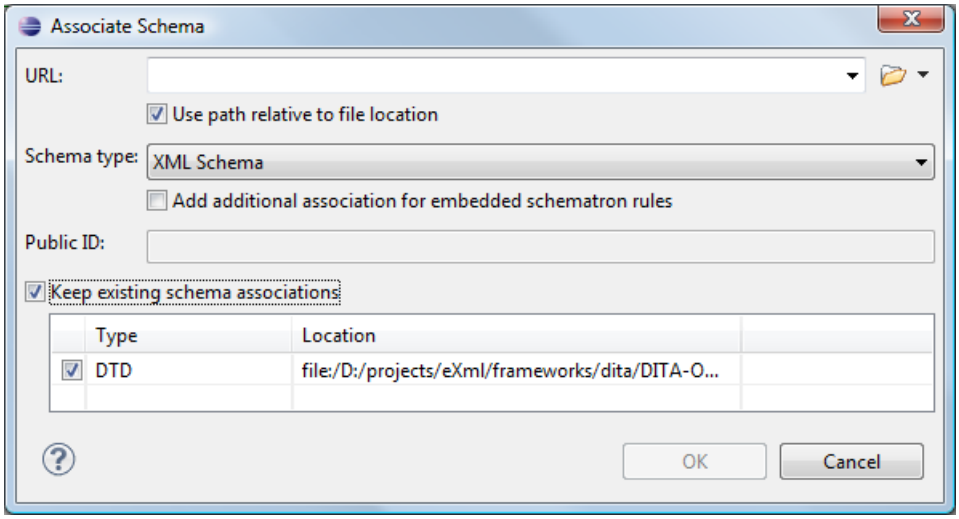


Figure 94: The Associate Schema Dialog Box

The available options are:

- **URL** - Contains a predefined set of schemas that are used more often and it also keeps a history of the last used schemas. The URL must point to the schema file which can be loaded from the local disk or from a remote server through HTTP(S), FTP(S).

- **Schema type** - Selected automatically from the list of possible types in the **Schema type** combo box (XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, NVDL) based on the extension of the schema file that was entered in the **URL** field.
- **Public ID** - Specify a public ID if you have selected a DTD.
- **Add additional association for embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules, enable this option.
- **Use path relative to file location** - Enable this option if the XML instance document and the associated schema contain relative paths. The location of the schema file is inserted in the XML instance document as a relative file path. This practice allows you, for example, to share these documents with other users, without running into problems caused by different project locations on physical disk.
- **Keep existing schema associations** - Enable this option to keep the associations of the currently edited document with a Schema when you associate a new one.

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

- `xsi:schemaLocation` attribute, if the root element of the document sets a default namespace with an `xmlns` attribute.
- `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 includes the following attributes:

- `href` (required) - The schema file location.
- `type` - The content type of the schema. This is an optional attribute with the following possible values for each specified type:
  - DTD - The recommended value is `application/xml-dtd`.
  - W3C XML Schema - The recommended value is `application/xml`, or can be left unspecified.
  - RELAX NG XML Syntax - The recommended value is `application/xml`, or can be left unspecified.
  - RELAX NG Compact Syntax - The recommended value is `application/relax-ng-compact-syntax`.
  - Schematron - The recommended value is `application/xml`, or can be left unspecified.
  - NVDL - The recommended value is `application/xml`, or can be left unspecified.
- `schematypens` - The namespace for the schema language of the referenced schema with the following possible values:
  - DTD - Not specified.
  - W3C XML Schema - The recommended value is `http://www.w3.org/2001/XMLSchema`.
  - RELAX NG XML Syntax - The recommended value is `http://relaxng.org/ns/structure/1.0`.

  - RELAX NG Compact Syntax - Not specified.
  - Schematron - The recommended value is `http://purl.oclc.org/dsdl/schematron`.
  - NVDL - The recommended value is `http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0`.
- `phase` - The phase name for the validation function in Schematron schema. This is an optional attribute. To run all phases from the Schematron, use the special `#ALL` value. If the phase is not specified, the default phase that is configured in the Schematron will be applied.
- `title` - The title for the associated schema. This is an optional attribute.

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

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

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

Learning Document Structure

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

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

Create a DTD from Learned Document Structure

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

To create a DTD from the learned structure:

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

Content Completion Assistant

The intelligent **Content Completion Assistant** available in Oxygen XML Editor plugin enables rapid, in-line identification and insertion of structured language elements, attributes and, in some cases, their parameter options.

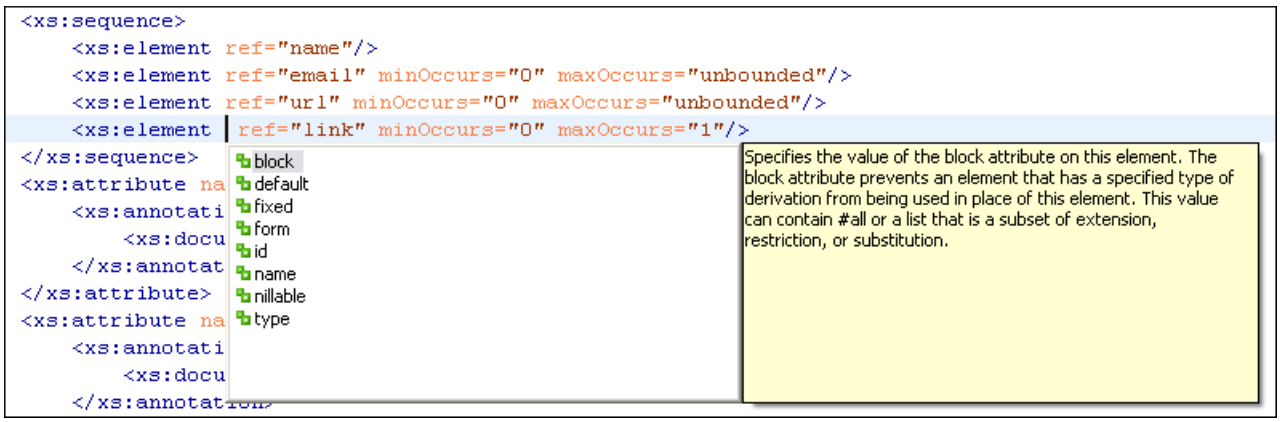


Figure 95: Content Completion Assistant

The functioning of the **Content Completion Assistant** feature is schema-driven (XML Schema, DTD, and RELAX NG). When Oxygen XML Editor plugin detects a schema, it logs its URL in the *Status view*.

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

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

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

When active, it displays a list of context-sensitive proposals valid at the current cursor position. Elements are selected 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, do one of the following:

- Press Enter or Tab key on your keyboard to insert both the start and end tags. The cursor is positioned inside the start tag, in a position suitable for inserting attributes.
- Press **Ctrl Enter (Meta Enter on OS X)** on your keyboard. Oxygen XML Editor plugin inserts both the start and end tags and positions the cursor between the tags, so you can start typing content.

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

After inserting an element, the cursor is positioned:

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

The **Content Completion Assistant** is displayed:

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

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

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

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

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

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

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

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

Set Schema for Content Completion

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

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

Content Completion in Documents with Relax NG Schemas

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

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

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

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

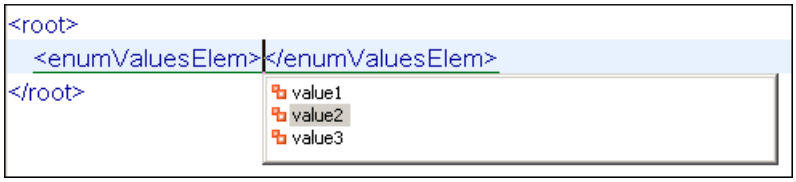


Figure 96: Content Completion assistant - element values in Relax NG documents

Schema Annotations

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

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

The schema annotations support is available 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 Editor plugin looks for an annotation in the type definition of that element or attribute.

Collecting Annotations from Relax NG Schemas

For Relax NG schema element / attribute annotation are made using the `<documentation>` element from the `http://relaxng.org/ns/compatibility/annotations/1.0` namespace. 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 behavior, enable the ***Use all Relax NG annotations as documentation*** option.

Collecting Annotation from DTDs

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

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

Content Completion Helper Views

Information about the current element being edited is also available in various views, such as the **Model** view, **Attributes** view, **Elements** view, and **Entities** view. By default, they are located on the right-hand side of the main editor window. These views, along with the powerful **Outline** view, provide spatial and insight information about the edited document and the current element.

The Model View

The **Model** view presents the structure of the currently selected tag, and its documentation, defined as annotation in the schema of the current document. To open the **Model** view, select it from the **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** menu.

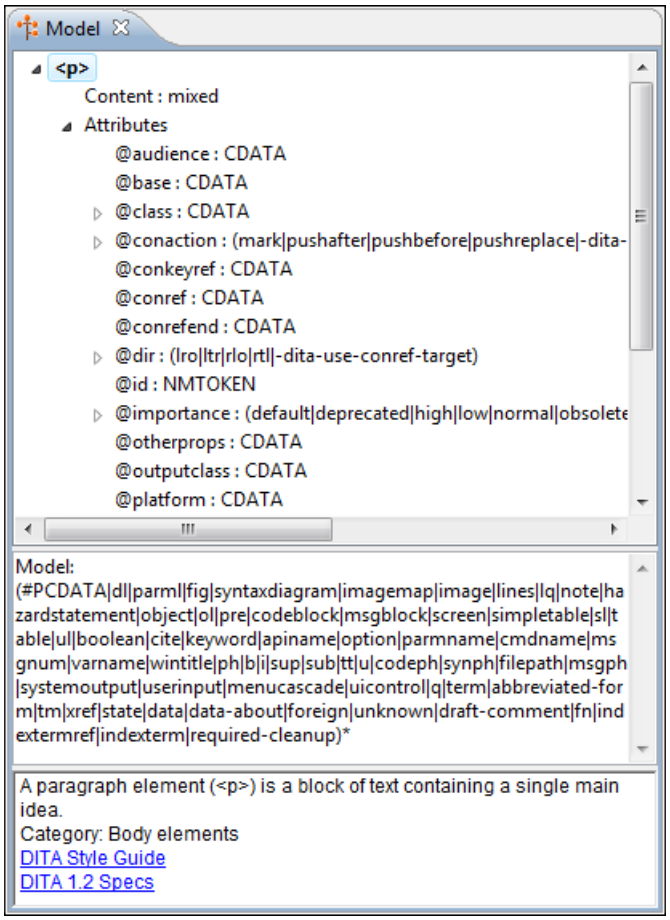


Figure 97: The Model View

The **Model** view is comprised of two sections, an element structure panel and an annotations panel.

Element Structure Panel

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

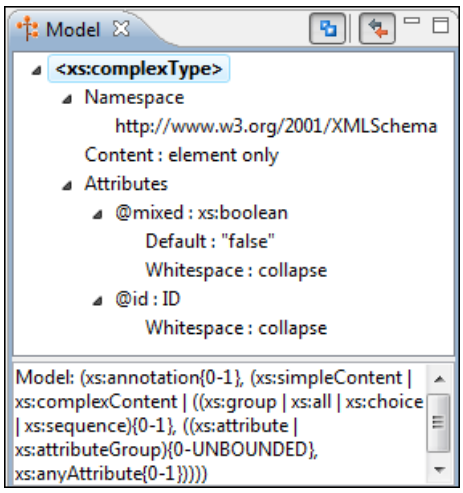


Figure 98: The Element Structure Panel

Annotation Panel

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

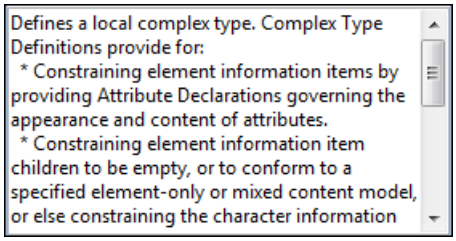


Figure 99: The Annotation panel

The Attributes View

The **Attributes** view presents all the attributes of the current element determined by the schema of the document.

You can use the **Attributes** view to insert attributes, edit their values, or add values to existing attributes.

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. If 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.

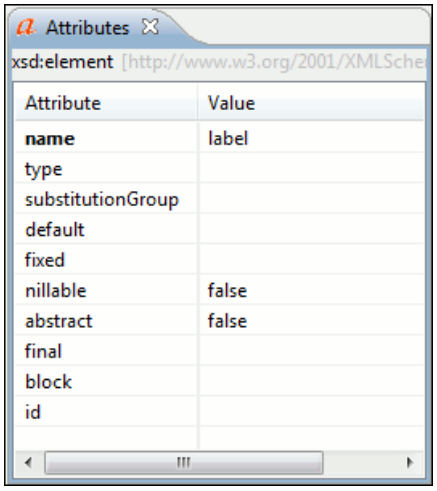


Figure 100: The Attributes View

Contextual Menu Actions in the Attributes View

The following actions are available in the contextual menu of the **Attributes** view when editing in **Text** mode:

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, if the attribute does not appear in the edited document.
- Empty, if the attribute does not appear in the edited document and has no default value set.

Paste

Depending on the content of the clipboard, the following cases are possible:

- If 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.
- If 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.
- If the clipboard only contains text, the value of the selected attribute is modified.

The Elements View

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

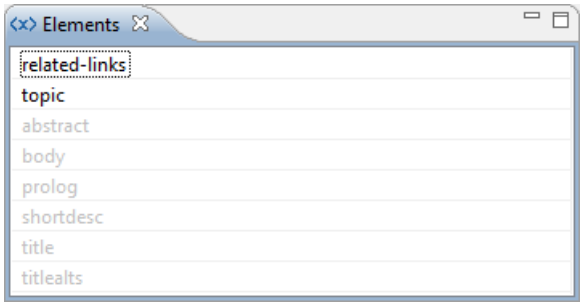


Figure 101: The Elements View

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 by clicking the column headers.

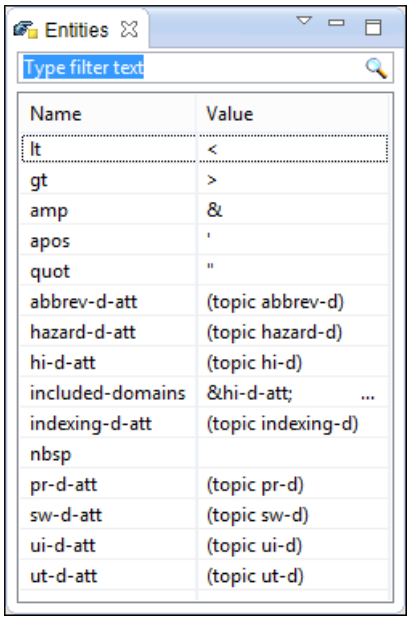


Figure 102: The Entities View

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

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

Code Templates

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

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

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

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

Configuring the Proposals in the Content Completion Assistant

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

To configure content completion proposals, follow these steps:

1. Create a new `resources` folder (if it does not already exist) in the frameworks directory for the document type. For instance: `OXYGEN_INSTALL_DIR/frameworks/dita/resources`.
2. *Open the **Preferences** dialog box* and go to **Document Type Association**. Edit the document type configuration for your XML vocabulary, and in the **Classpath** tab add a link to that `resources` folder.
3. Use the **New** document wizard to create a configuration file using the **Content Completion Configuration** file template.
4. Make the appropriate changes to your custom configuration file. The file template includes details about how each element and attribute is used in the configuration file.
5. Save the file in the `resources` folder, using the fixed name: `cc_value_config.xml`.
6. Re-open the application and open an XML document. In the **Content Completion Assistant** you should see your customizations.

The Configuration File

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

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

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

Example - Specifying Values Directly

```
<!-- Replaces the values for an element with the local name "lg", from the given namespace -->
<match elementName="lg" elementNS="http://www.oxygenxml.com/ns/samples">
    <items action="replace">
        <item value="stanza"/>
        <item value="refrain"/>
    </items>
</match>

<!-- Adds two values for an attribute with the local name "type", from any namespace -->
<match attributeName="type">
    <items>
        <item value="stanza"/>
        <item value="refrain"/>
    </items>
</match>
```

Example - Calling an External XSLT Script

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

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

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

Configuring Proposals in the Context for which the Content Completion was Invoked

A more complex scenario for configuring the content completion proposals would be if you want to choose the possible values to provide, depending on the context of the element in which the content completion was invoked.

Suppose that you want to propose certain possible values for one property (for example, *color*) and other values for another property (for example, *shape*). If the property represents a color, then the values should represent applicable colors, while if the property represents a shape, then the values should represent applicable shapes. See the following code snippets:

Your main document:

```
<sampleArticle>
  <!-- The possible values for @value should be "red" and "blue" -->
  <property name="color" value=""/>
  <!-- The possible values for @value should be "square" and "rectangle" -->
  <property name="shape" value=""/>
</sampleArticle>
```

The content completion configuration file:

```
<config xmlns="http://www.oxygenxml.com/ns/ccfilter/config">
    <match elementName="property" attributeName="value">
      <xslt href="get_values.xsl" useCache="false" action="replace"/>
    </match>
</config>
```

The stylesheet that defines the possible values based on the context of the property on which the content completion was invoked:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:saxon="http://saxon.sf.net/"
  exclude-result-prefixes="xs"
  version="2.0">

  <xsl:param name="documentSystemID" as="xs:string"></xsl:param>
  <xsl:param name="contextElementXPathExpression" as="xs:string"></xsl:param>

  <xsl:template name="start">
    <xsl:apply-templates select="doc($documentSystemID)"/>
  </xsl:template>

  <xsl:template match="/">
    <xsl:variable name="propertyElement"
        select="saxon:eval(saxon:expression($contextElementXPathExpression, ./*))"/>

    <items>
      <xsl:if test="$propertyElement/@name = 'color'">
        <item value='red'/>
        <item value='blue'/>
      </xsl:if>
      <xsl:if test="$propertyElement/@name = 'shape'">
        <item value='rectangle'/>
        <item value='square'/>
      </xsl:if>
    </items>
  </xsl:template>
</xsl:stylesheet>
```

The `contextElementXPathExpression` parameter will be bound to an XPath expression that identifies the element in the context for which the content completion was invoked.

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 all XML documents should be compatible. With HTML it is possible to create documents with lots of errors (for example, when you forget an end tag). One of the main reasons that various HTML browsers have performance and compatibility problems is that they have different methods of figuring out how to render a document when an HTML error is encountered. Using XML helps to eliminate such problems.

Even when creating XML documents, errors are very easily introduced. When working with large projects or a large number of files, the probability that errors will occur is even greater. Preventing and solving errors in your projects can

be time consuming and frustrating. Fortunately, Oxygen XML Editor plugin provides validation functions that enable you to easily identify errors and their 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, whitespace 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 (for example, an element in whose content the prefixed markup occurs). Furthermore, the attribute value in the innermost such declaration MUST NOT be an empty string.

To check if a document is *Namespace Well-Formed XML*, select the **Check Well-Formedness (Alt + Shift + V, W (Meta + Alt + V, W on OS X))** action from the **XML** menu or from the **Validation** toolbar drop-down menu. 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 selected files in the current project can be checked for well-formedness with a single action by selecting the **Check Well-Formedness** action from the **Validate** submenu when invoking the contextual menu in the *Navigator view*.

Validating XML Documents Against a Schema

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

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

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

Presenting Validation Errors in Text Mode

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, but the color will be yellow instead of red. Hovering over a validation error presents a tooltip message with more details about the error and *possible quick fixes* (if available for that error or warning).

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 you to locate them more easily. The ruler contains the following areas:

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

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

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

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

Presenting Validation Errors in Author Mode

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

Figure 103: Presenting Validation Errors in Author Mode

A fragment with a validation error is marked by underlining the error in red, and validation warnings are underlined in yellow.

Also, the ruler on the right side of the editor panel is designed to display the errors found during the validation process and to help you locate them in the document. The ruler contains the following:

- The top area - A success indicator square will turn green if the validation is successful, red if validation errors are found, or yellow if validation warnings are found.More details about the errors or warnings are displayed in a tool tip when you hover over indicator square. If there are numerous errors, only the first three are presented in the tool tip.
- The middle area - Errors are depicted with red markers, and warnings with yellow markers. If you want to limit the number of markers that are displayed, *open the **Preferences** dialog box* and go to **Editor** > **Document checking** > **Maximum number of validation highlights**.

  Clicking a marker will highlight the corresponding text area in the editor. The error or warning message is also displayed both in a tool tip (when hovering over the marker) and in the message area on the bottom of the editor panel.

  The validation status area at the bottom of the editor panel presents the message of the current validation error. Clicking the **Document checking options** button opens the ***Document checking user preferences*** page
- The bottom area - Two navigation arrows ( ) allow you to skip to the next or previous error. The same actions can be triggered from **Document** > **Automatic validation** > **Next error** (**Ctrl Period (Meta Period on OS X)**) and **Document** > **Automatic validation** > **Previous error** (**Ctrl Comma (Meta Comma on OS X)**).

Status messages from every validation action are logged in the ***Console** view* (the **Enable oXygen consoles** option must be enabled in **Preferences** > **View**).

Customizing Assert Error Messages

To customize 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 if 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 must match”.

Luckily most standards based DTDs, XML Schemas, 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 Editor plugin *can be configured* to mark validation errors in the document as you are editing. If you *enable the **Automatic validation** option*, all 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*. Hovering over a validation error presents a tooltip message with more details about the error.

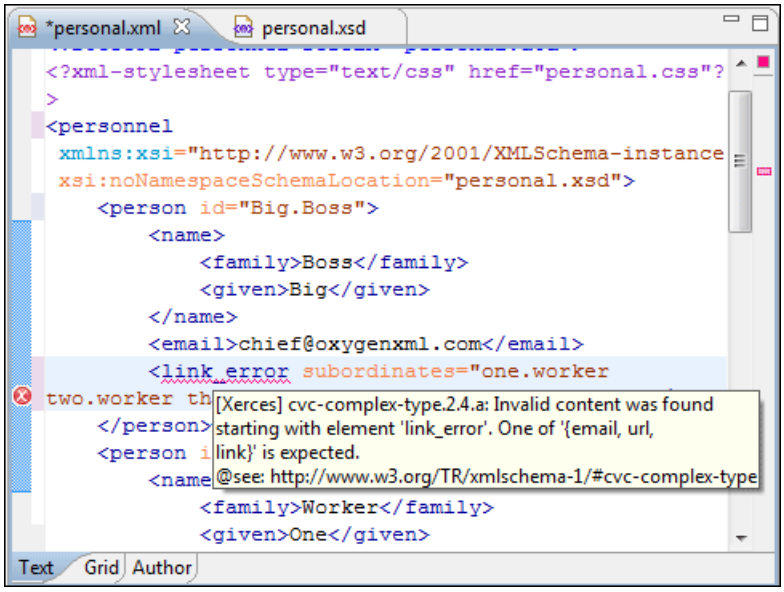


Figure 104: Automatic Validation on the Edited Document

Custom Validators

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

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

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

  For validation against an external DTD specified by URI in the XML document, 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 Editor plugin is installed in the default location on Windows (C:\Program Files) because the built-in XML catalog files are stored in the `frameworks` subfolder of the installation folder which in this case contains at least one space character in the file path.

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

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

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

- **Saxon-EE** - Included in Oxygen XML Editor plugin. It is associated to XML Editor and XSD Editor. It is able to validate XML Schema schemas and XML documents against XML Schema schemas. The validation is done according to the W3C XML Schema 1.0 or 1.1. This can be *configured in Preferences*.
- **MSXML 4.0** - Included in Oxygen XML Editor plugin (Windows edition only). It is associated to XML Editor, XSD Editor and XSL Editor. It is able to validate the edited document against XML Schema, internal DTD (included in the XML document), external DTD or a custom schema type.

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

Linked Output Messages of an External Engine

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

Linked messages have the following format:

- *Type*:[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 Editor plugin validates the main module of the document when an imported module is checked for errors.

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

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

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

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

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

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

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

How to Create a Validation Scenario

To create a validation scenario, follow these steps:

1. Select the **Configure Validation Scenario(s)** from the **XML** menu, or the toolbar, or from the **Validate** submenu when invoking the contextual menu on a file in the **Navigator** view.
   The **Configure Validation Scenario(s)** dialog box is displayed. It contains the following types of scenarios:
   - **Predefined** scenarios are organized in categories depending on the type of file they apply to. You can identify **Predefined** scenarios by a yellow key icon that marks them as *read-only*. If the predefined scenario is the default scenario of the framework, its name is written in bold font. You can use the options in the **Settings** drop-down menu to filter which scenarios are shown.
   - **User defined** scenarios are organized under a single category, but you can use the options in the **Settings** drop-down menu to filter them.

     Note: 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.

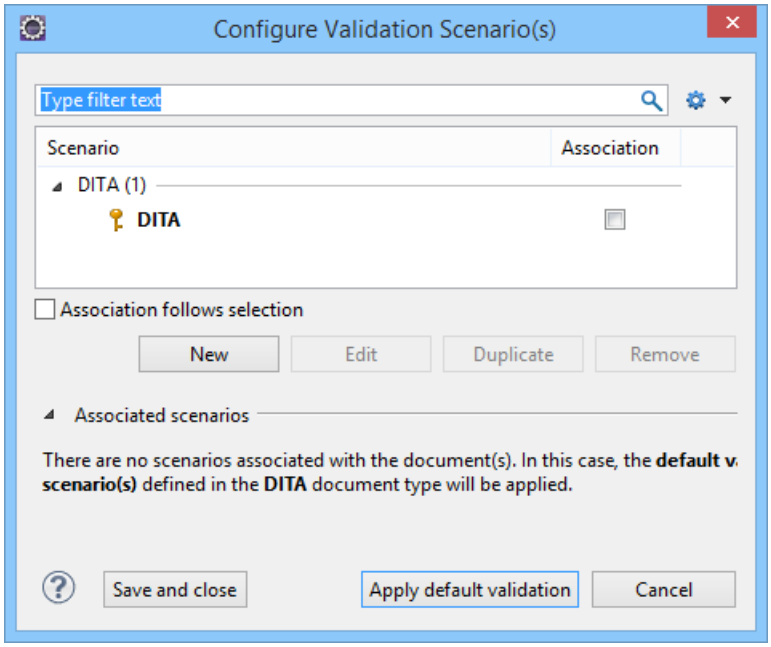


Figure 105: Configure Validation Scenario

2. To edit an existing scenario, select the scenario and press the **Edit** button. If you try to edit one of the *read-only* predefined scenarios, Oxygen XML Editor plugin creates a customizable duplicate (you can also use the **Duplicate** button).
3. To add a new scenario, press the **New** button.
   The **New scenarios** dialog box is displayed. It lists all the validation units for the scenario.

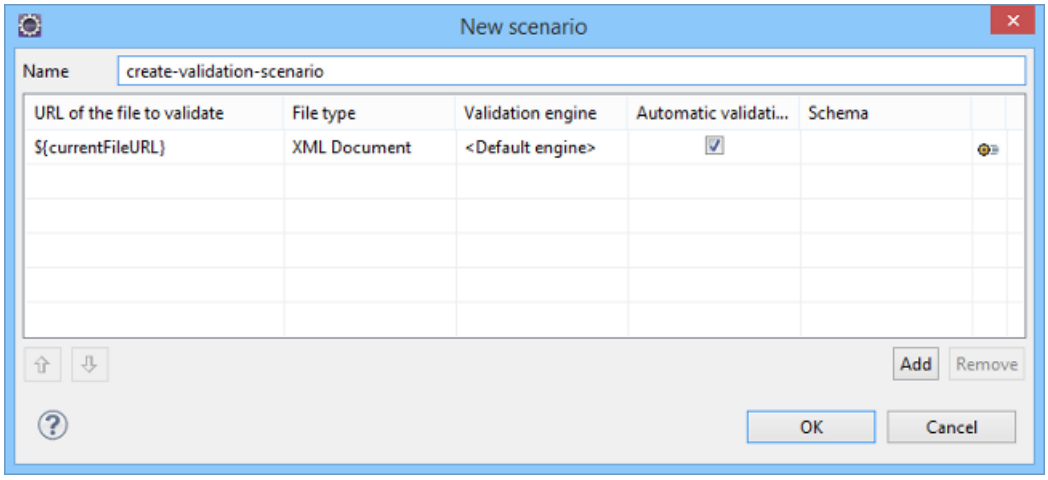


Figure 106: Add / Edit a Validation Unit

The **New scenario** dialog box includes the following information:

- **Name** - The name of the current validation scenario.
- **URL of the file to validate** - The URL of the main module that includes the current module. It is also the entry module of the validation process when the current one is validated.
- **File type** - The type of the document that is validated in the current validation unit. Oxygen XML Editor plugin automatically selects the file type depending on the value of the **URL of the file to validate** field.
- **Validation engine** - One of the engines available in Oxygen XML Editor plugin for validation of the type of document to which the current module belongs. **Default engine** means that the default engine is used to run the

validation and is set in the **Preferences** pages for the current document type (XML, XML Schema, XSLT, XQuery, etc.).

- **Automatic validation** - If this option is checked, the validation operation defined by this row is also applied by *the automatic validation feature*. If the **Automatic validation** feature is *disabled in Preferences*, then this option is ignored, as the Preference setting has a higher priority.
- **Schema** - This option becomes active when you set the **File type** to **XML Document**.
- **Settings** - Opens the **Specify Schema** dialog box that allows you to set a schema for validating XML documents, or a list of extensions for validating XSL or XQuery documents. You can also set a default phase for validation with a Schematron schema.

4. If you want to add a new validation unit, press the **Add** button.
5. To edit the URL of the main validation module, click its cell in the **URL of the file to validate** column.

   Specify the URL of the main module by doing one of the following:

   - Enter the URL in the text field or select it from the drop-down list.
   - Use the **Browse** drop-down button to browse for a local, remote, or archived file.
   - Use the **Insert Editor Variable** button to insert an *editor variable* or a *custom editor variable*.

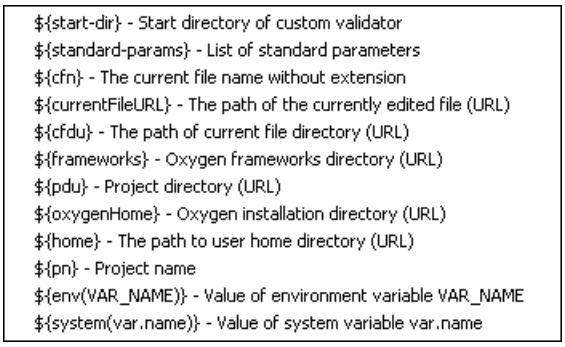


Figure 107: Insert an Editor Variable

6. Select the **File type** of the validated document.

   Note that this determines the list of possible validation engines.
7. Select the **Validation engine** by clicking its cell and selecting it from the drop-down list.
8. Select the **Automatic validation** option if you want the current unit to be used by *the automatic validation feature*.
9. Choose the schema to be used during validation (the schema detected after parsing the document or a custom one).
10. Press **OK**.

    The newly created validation scenario is now included in the list of scenarios in the **Configure Validation Scenario(s)** dialog box. You can select the scenario in this dialog box to associate it with the current document and press the **Apply associated** button to run the validation scenario.

Sharing Validation Scenarios

The validation scenarios and their settings can be shared with other users by *exporting them to a specialized scenarios file* that can then be imported.

Validation Actions in the User Interface

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

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

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

  Note: Automatic validation also caches the associated schema.

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

To open the schema used for validating the current document, select the **Open Associated Schema** action from the **XML** menu.

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

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

Resolving References to Remote Schemas with an XML Catalog

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

```
<?xml-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.

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 only the ones you need to edit in the focus. Expanding and collapsing works on individual elements. Expanding an element leaves the child elements unchanged.

Figure 108: Folding of the XML Elements

To toggle the folded state of an element, click the special mark displayed in the left part of the document editor, next to the start tag of that element.

The following other menu actions related to folding of XML elements are available from the contextual menu of the current editor (in **Author** mode they are also available from the **Folding** submenu of the contextual menu):

Collapse Other Folds (Ctrl NumPad / (Meta NumPad / on OS X))
Folds all the elements except the current element.

Collapse Child Folds (Ctrl + NumPad- (Meta + NumPad- on OS X))
Folds the elements indented with one level inside the current element.

Expand Child Folds (Ctrl NumPad+ (Meta NumPad+ on OS X))
Unfolds all child elements of the currently selected element.

Expand All (Ctrl NumPad * (Meta NumPad * on OS X))
Unfolds all elements in the current document.

To watch our video demonstration about the folding support in Oxygen XML Editor plugin, go to *http://oxygenxml.com/demo/FoldingSupport.html*.

Navigation Using the Outline View

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

By default it is displayed on screen, but if you closed it you can reopen it from **Window** > **Show View** > **Outline**.

The upper part of the **Outline** view contains a filter box that 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.

It also includes a **View menu** in the top-right corner that presents a variety of options to help you filter the view even further.

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 you to be aware of the document structure and the way tags are nested.

The **Outline** view allows you to:

- Insert or delete nodes using contextual 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 Single-Click (Command Single-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.

Drag and Drop Actions in Outline View

Entire XML elements can be moved or copied in the edited document using only the mouse in the **Outline** view with 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 a Preferences page.*

Tip: You can select and drag multiple nodes in the **Outline** view when editing in **Author** mode.

Document Tag Selection

The Outline view can also be used to search for a specific tag 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 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 the focus in the editor.

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

Finding and Replacing Text in the Current File

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

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

The Find All Elements Dialog Box

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

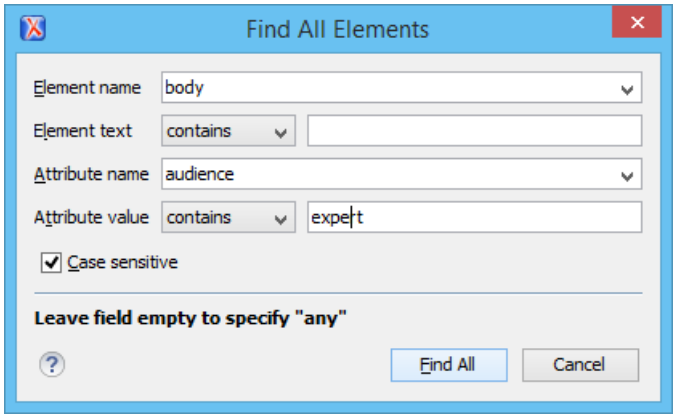


Figure 109: Find All Elements Dialog Box

The dialog box can perform the following actions:

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

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

The following fields are available in the dialog box:

- **Element name** - The qualified name of the target element to search for. You can use the drop-down menu to find an element or enter it manually. It is populated with valid element names collected from the associated schema. To specify *any* element name, leave the field empty.

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

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

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

- **Attribute value** - The drop-down menu beside this field allows you to specify that you are looking for an exact or partial match of the attribute value. For *any* or no attribute value, select **contains** from the drop-down menu and

leave the field empty. If you leave the field empty but select **equals** from the drop-down menu, only elements that have at least an attribute with an empty value will be found. Select **not contains** to find all elements that have attributes without a specified value.

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

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

Regular Expressions Syntax

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

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

There are some other notable differences that may cause unexpected results, including the following:

- 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 `*`.

Editing Large XML Documents

Consider the case of documenting a large project. It is likely that there will be several people involved. The resulting document can be few megabytes in size. The question becomes how to deal with this amount of data in such a way that work parallelism will not be affected.

Fortunately, XML provides two solutions for this: DTD entities and XInclude. A master document can be created, with references to the other document parts, containing the document sections. The users can edit the documents individually, then apply an XSLT stylesheet over the master and obtain the output files in various formats (for example, PDF or HTML).

Including Document Parts with DTD Entities

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

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

A master document looks like this:

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

```
<book>
<chapter> ...
```

The referenced 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 not define the schema again 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 Editor plugin install folder.

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

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

Main article file looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.docbook.org/xml/4.3/docbookx.dtd"
[ <!ENTITY % xinclude SYSTEM "../frameworks/docbook/dtd/xinclude.mod">
%xinclude;
```

```
]>
<article>
    <title>Install guide</title>
    <para>This is the install guide.</para>
    <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
                href="introduction.dita">
      <xi:fallback>
        <para>
          <emphasis>FIXME: MISSING XINCLUDE CONTENT</emphasis>
        </para>
      </xi:fallback>
    </xi:include>
</article>
```

In this example the following is of note:

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

Working with XML Catalogs

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

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

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

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

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

Table 5: Catalog Mappings

Document	Referenced Resource	Mappings
XML	DTD	*system* or *public* The ***Prefer*** option controls which one of the mappings should be used.
	XML Schema Relax NG Schematron NVDL	The following strategy is used (if one step fails to provide a resource, the next is applied): **1.** resolve the schema using *URI* catalog mappings. **2.** resolve the schema 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 the root *namespace* using *URI* catalog 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): **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).
Relax NG	Relax NG	

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

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

- User-defined catalogs set globally in the *XML Catalog preferences* page.
- User-defined catalogs set at document type level, in the **Catalog** tab from the ***Document Type** configuration dialog box*.
- Built-in catalogs.

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

Considering the following XML document code snippet:

```
<topic 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 Editor plugin resolves the location of a schema in the following order:

- First, it attempts to resolve the schema location as a URI (`uri`, `uriSuffix`, `rerwriteURI` mappings 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 location by processing the schema namespace as a URI (`uri`, `uriSuffix`, `rewriteURI` from the XML catalog). The namespace is taken into account only when the schema specified in the *schemaLocation* attribute was not resolved successfully. 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** > **Other** > **Oxygen XML Editor plugin** > **Resource Hierarchy/Dependencies**. As an alternative, right click the current document and either select **Resource Hierarchy** or **Resource Dependencies**.

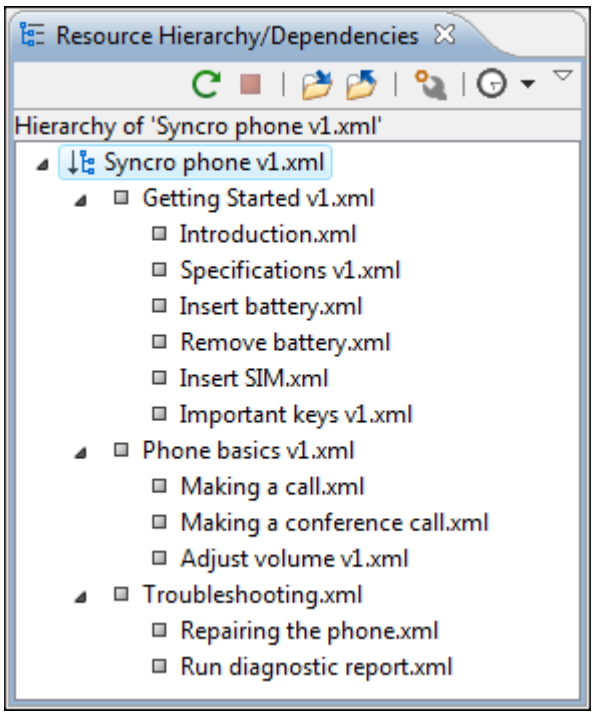


Figure 110: Resource Hierarchy/Dependencies View - Hierarchy for `Syncro phone v1.xml`

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

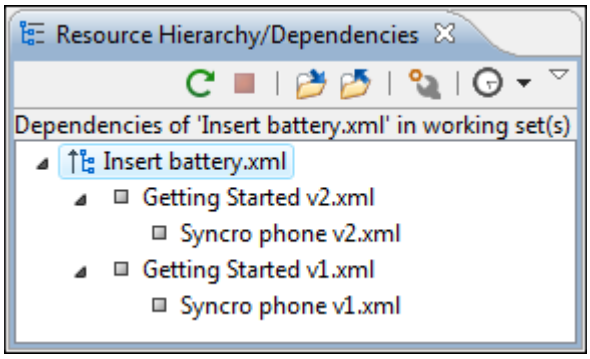


Figure 111: Resource Hierarchy/Dependencies View - Dependencies for `Insert battery.xml`

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

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

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

History

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

The contextual menu contains the following actions:

Open

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

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the **Master Files** directory*.

Expand All

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

Collapse All

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

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

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

Moving/Renaming XML Resources

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

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

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

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

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

Converting Between Schema Languages

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

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

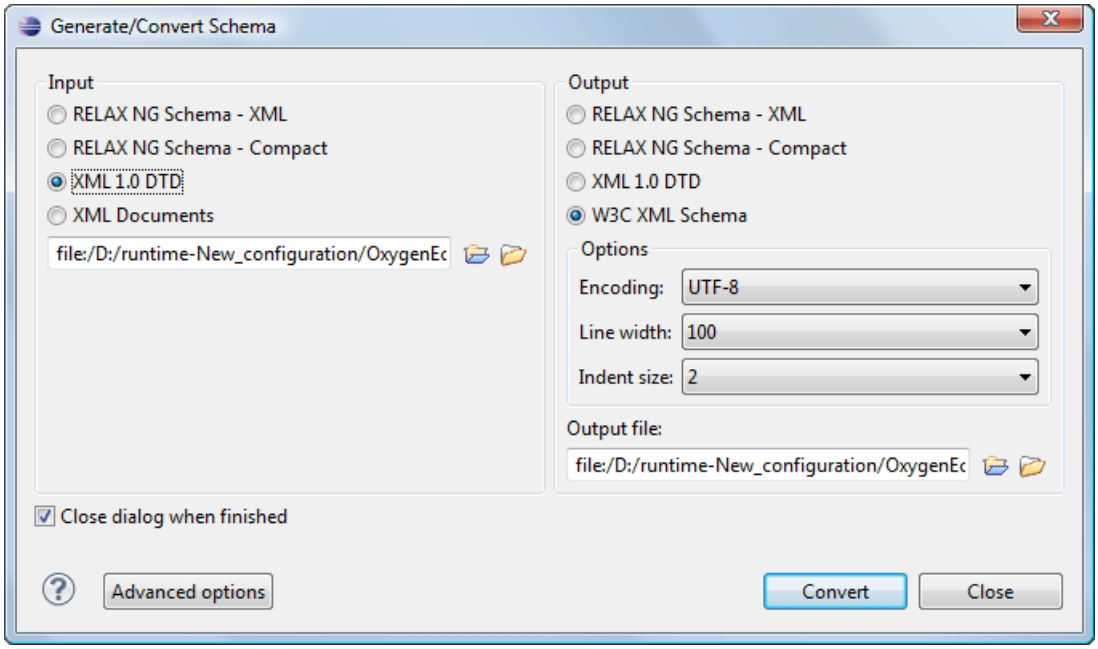


Figure 112: Convert a Schema to Other Schema Language

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

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

Input panel:

- **xmlns** - Specifies the default namespace, that is the namespace used for unqualified element names.
- **xmlns** - Each row specifies the prefix used for a namespace in the input schema.
- **colon-replacement** - Replaces colons in element names with the specified chars when constructing the names of definitions used to represent the element declarations and attribute list declarations in the DTD.
- **element-define** - Specifies how to construct the name of the definition representing an element declaration from the name of the element. The specified value must contain exactly one percent character. This percent character is replaced by the name of element (after colon replacement) and the result is used as the name of the definition.
- **inline-attlist** - Instructs the application not to generate definitions for attribute list declarations, but instead move attributes declared in attribute list declarations into the definitions generated for element declarations. This is the default behavior when the output language is XSD.
- **attlist-define** - Specifies how to construct the name of the definition representing an attribute list declaration from the name of the element. The specified value must contain exactly one percent character. This percent character is replaced by the name of element (after colon replacement) and the result is used as the name of the definition.
- **any-name** - Specifies the name of the definition generated for the content of elements declared in the DTD as having a content model of ANY.

- **strict-any** - Preserves the exact semantics of *ANY* content models by using an explicit choice of references to all declared elements. By default, the conversion engine uses a wildcard that allows any element
- **generate-start** - Specifies whether the conversion engine should generate a start element. DTD's do not indicate what elements are allowed as document elements. The conversion engine assumes that all elements that are defined but never referenced are allowed as document elements.
- **annotation-prefix** - Default values are represented using an annotation attribute `prefix:defaultValue` where prefix is the specified value and is bound to `http://relaxng.org/ns/compatibility/annotations/1.0` as defined by the RELAX NG DTD Compatibility Committee Specification. By default, the conversion engine will use a for prefix unless that conflicts with a prefix used in the DTD.

Output panel:

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

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 Editor plugin provides the support for defining the main module (or modules), allowing you to edit any file from the hierarchy in the context of the master XML files.

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

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

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

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

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

Search and Refactor Actions for IDs and IDREFS

Oxygen XML Editor plugin allows you to search for ID declarations and references (IDREFS) and to *define the scope of the search and refactor operations*. These operations are available for XML documents that have an associated DTD, XML Schema, or Relax NG schema. These operations are available through the search and refactor actions in the contextual menu. In **Text** mode, these actions are also available in the ***Quick Assist*** menu.

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 box. This dialog box 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 box, 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

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

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 Declarations

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

Note: Available in the **Text** mode only.

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***.

Note: Available in the **Text** mode only.

Search Occurrences in file

Searches for the declaration and references of the ID in the current document.

Tip: A quick way to go to the declaration of an ID in **Text** mode is to move the cursor over an ID reference and use the **Ctrl Single-Click (Command Single-Click on OS X)** navigation.

Selecting an ID for which you use search or refactor operations differs between the **Text** and **Author** modes. In the **Text** mode, you position the cursor inside the declaration or reference of an ID. In the **Author** mode, Oxygen XML Editor plugin collects all the IDs by analyzing each element from the path to the root. If more IDs are available, you are prompted to choose one of them.

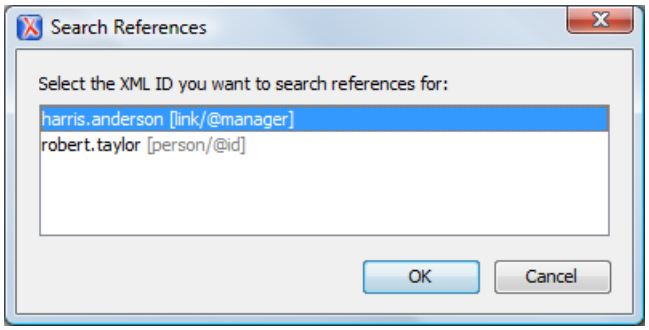


Figure 113: Selecting an ID in the Author Mode

Search and Refactor Operations Scope

The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the **Change scope** operation, available in the Quick Fix action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project or to one or multiple working sets. The **Use only Master Files, if enabled** check-box allows you to restrict the scope of the search and refactor operations to the resources from the **Master Files** directory. Click **read more** for details about the *Master Files support*.

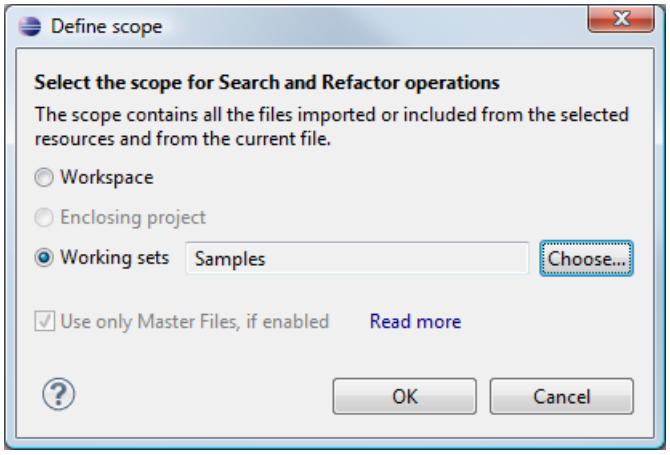


Figure 114: Change Scope Dialog Box

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

Viewing Status Information

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

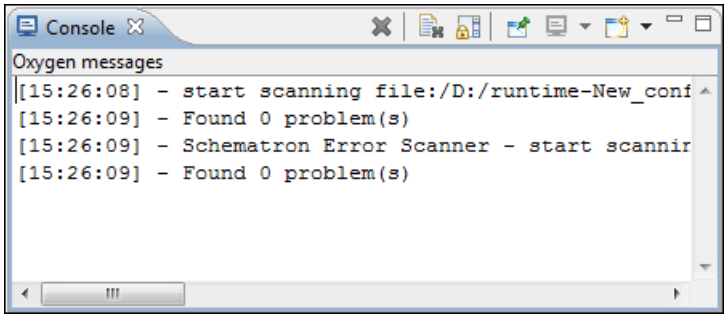


Figure 115: The Console view messages

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

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

Editor Highlights

An *editor highlight* is a text fragment emphasized by a colored background.

Highlights are generated in both **Text** and **Author** mode when the following actions generate results:

- Find/Replace
- Find All Elements
- XPath in Files

- Search References
- Search Declarations

By default, Oxygen XML Editor plugin uses a different color for each type of highlight (*XPath in Files*, *Find/Replace*, *Search References*, *Search Declarations*, etc.) 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 the highlights in the current document by using 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* at the bottom of the editor.

To remove the highlights, you can do the following:

- Click the **Remove all** button from bottom of the range ruler.
- Close the results tab at the bottom of the editor that contains the output of the action that generated the highlights.
- Click the **Remove all** button from the results panel at the bottom of the editor.

XML Quick Fixes

The Oxygen XML Editor plugin Quick Fix support helps you resolve errors that appear in an XML document by offering quick fixes to problems such as missing required attributes or invalid elements. Quick fixes are available in **Text** mode and **Author** mode

To activate this feature, hover over or place the cursor in the highlighted area of text where a validation error or warning occurs. If a Quick Fix is available for that particular error or warning, you can access the Quick Fix proposals with any of the following methods:

- When hovering over the error or warning, the proposals are presented in a tooltip popup window and the available quick fixes include a link that can be used to perform the fix.

Figure 116: Quick Fix Presented in a Tooltip in Text Mode

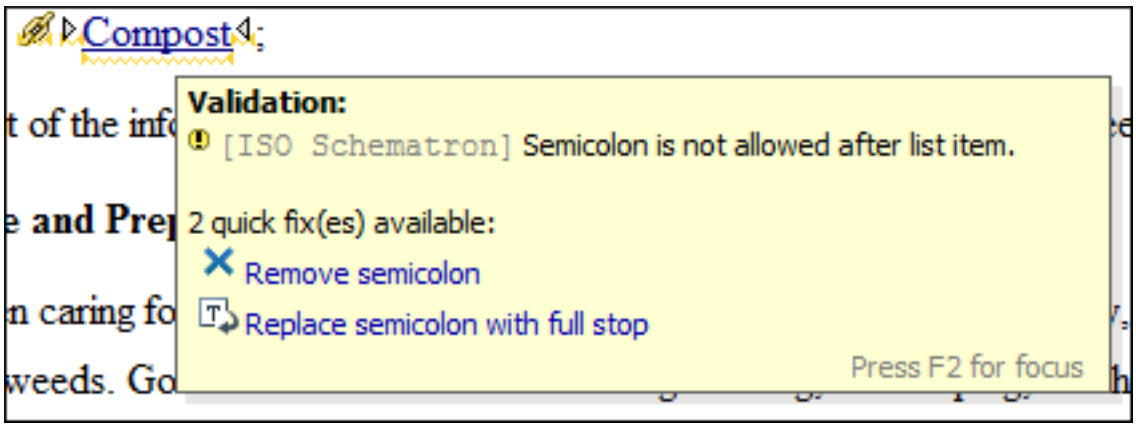


Figure 117: Quick Fix Presented in a Tooltip in Author Mode

- When hovering over the error or warning in **Author** mode, a small quick fix drop-down menu is presented. You can use the drop-down menu to display a list of available quick fixes to select from for the particular error or warning.

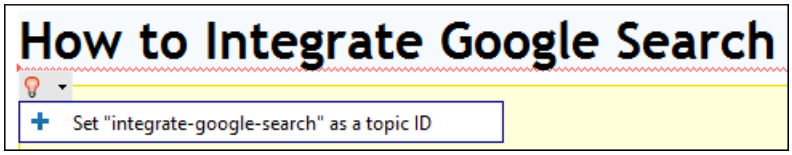


Figure 118: Quick Fix Drop-Down Menu in Author Mode

- If you place the cursor in the highlighted area where a validation error or warning occurs, a quick fix icon ( ) is displayed in the stripe on the left side of the editor. If you click this icon, Oxygen XML Editor plugin displays the list of available fixes.

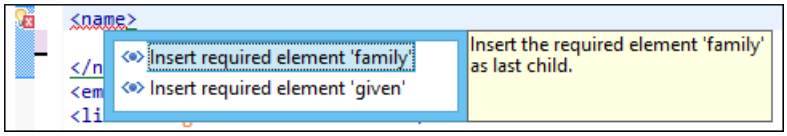


Figure 119: Quick Fix Menu Invoked by Clicking on the Icon

- With the cursor placed in the highlighted area of the error or warning, you can also invoke the quick fix menu by pressing **Ctrl 1 (Meta 1 on OS X)** on your keyboard.

Whenever you make a modification in the XML document or you apply a fix, the list of quick fixes is recomputed to ensure that you always have valid proposals.

Note: A quick fix that adds an element inserts it along with required and optional elements, and required and fixed attributes, depending on how the *Content Completion Assistant options* are configured.

Quick Fixes for XSD and Relax NG Errors

Oxygen XML Editor plugin offers quick fixes for common errors that appear in XML documents that are validated against XSD or Relax NG schemas.

Note: For XML documents validated against XSD schemas, the quick fixes are only available if you use the default Xerces validation engine.

Quick fixes are available in **Text** mode and **Author** mode.

Oxygen XML Editor plugin provides quick fixes for numerous types of problems, including the following:

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
The content of the element should be empty	Remove the element content
An element is not allowed to have child elements	Remove all child elements
Text is not allowed in the current element	Remove the text content
A required attribute is missing	Insert the required attribute
An attribute is not allowed to be set for the current element	Remove the attribute
The attribute value is invalid	Propose the correct attribute values
ID value is already defined	Generate a unique ID value
References to an invalid ID	Change the reference to an already defined ID

Schematron Quick Fixes (SQF)

Oxygen XML Editor plugin provides support for Schematron Quick Fixes (SQF). They help you resolve errors that appear in XML documents that are validated against Schematron schemas by offering you solution proposals. The Schematron Quick Fixes are an extension of the Schematron language and they allow you to define fixes for Schematron error messages. Specifically, they are associated with *assert* or *report* messages.

A typical use case is using Schematron Quick Fixes to assist technical writers with common editing tasks. For example, you can use Schematron rules to automatically report certain validation warnings (or errors) when performing regular editing tasks, such as inserting specific elements or changing IDs to match specific naming conventions. For more details and examples, please see the following blog: *http://blog.oxygenxml.com/2015/05/schematron-checks-to-help-technical.html*.

Displaying the Schematron Quick Fix Proposals

The defined Schematron Quick Fixes are displayed on validation errors in **Text** mode and **Author** mode.

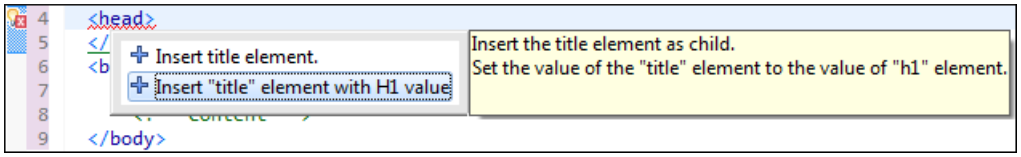


Figure 120: Example of a Schematron Quick Fix

Refactoring XML Documents

In the life cycle of XML documents there are instances when the XML structure needs to be changed to accommodate various needs. For example, when an associated schema is updated, an attribute may have been removed, or a new element added to the structure.

These types of situations cannot be resolved with a traditional *Find/Replace* tool, even if the tool accepts regular expressions. The problem becomes even more complicated if an XML document is computed or referenced from multiple modules, since multiple resources need to be changed.

To assist you with these types of refactoring tasks, Oxygen XML Editor plugin includes a specialized **XML Refactoring** tool that helps you manage the structure of your XML documents.

XML Refactoring Tool

The **XML Refactoring** tool is presented in the form of an easy to use wizard that is designed to reduce the time and effort required to perform various structure management tasks. For example, you can insert, delete, or rename an attribute in all instances of a particular element that is found in all documents within your project.

To access the tool, select the **XML Refactoring** action from one of the following locations:

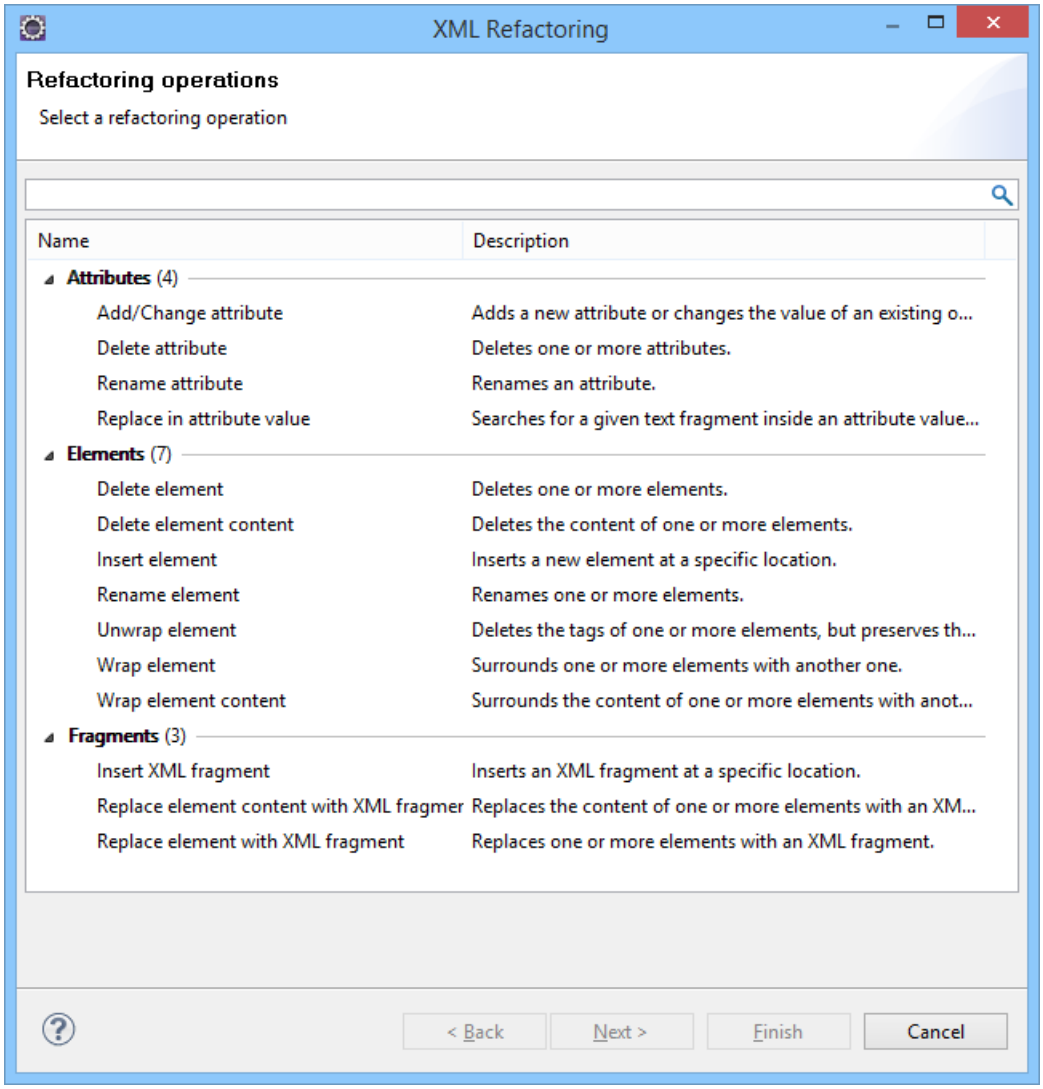
- The **XML Tools** menu.
- The **Refactoring** submenu from the contextual menu in the **Navigator** view.
- The **Refactoring** submenu from the contextual menu in the **DITA Maps Manager** view.

Note: The predefined refactoring operations are also available from the **Refactoring** submenu in the contextual menu of **Author** or **Text** mode. This is useful because by selecting the operations from the contextual menu, Oxygen XML Editor plugin considers the editing context to skip directly to the wizard page of the appropriate operation and to help you by preconfiguring some of the parameter values.

The tool includes the following wizard pages:

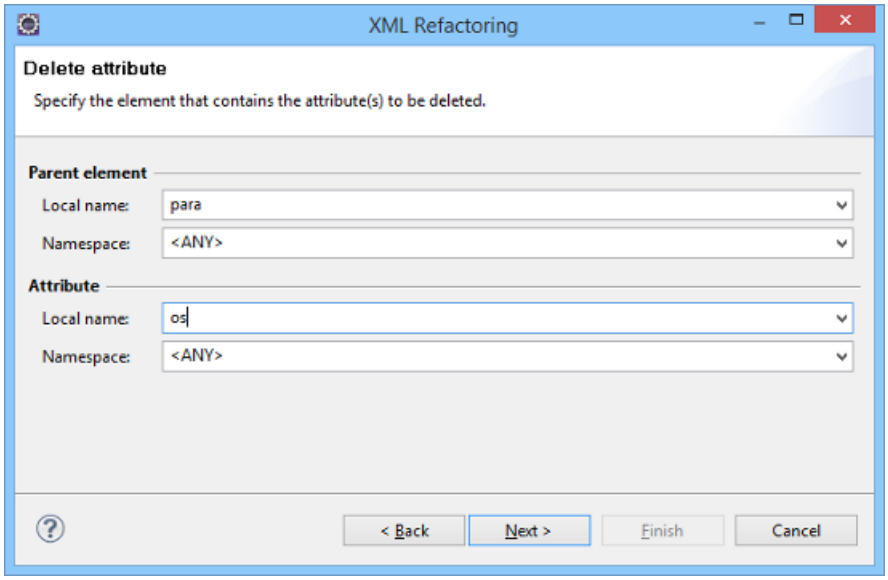
Refactoring operations

The first wizard page displays, and allows you to select, the available operations, which are grouped by category. To search for an operation, you can use the filter text box at the top of the page.



Configure Operation Parameters

The next wizard page allows you to specify the parameters for the refactoring operation. The parameters are specific to the type of refactoring operation that is being performed. For example, to delete an attribute you need to specify the parent element and the qualified name of the attribute to be removed.

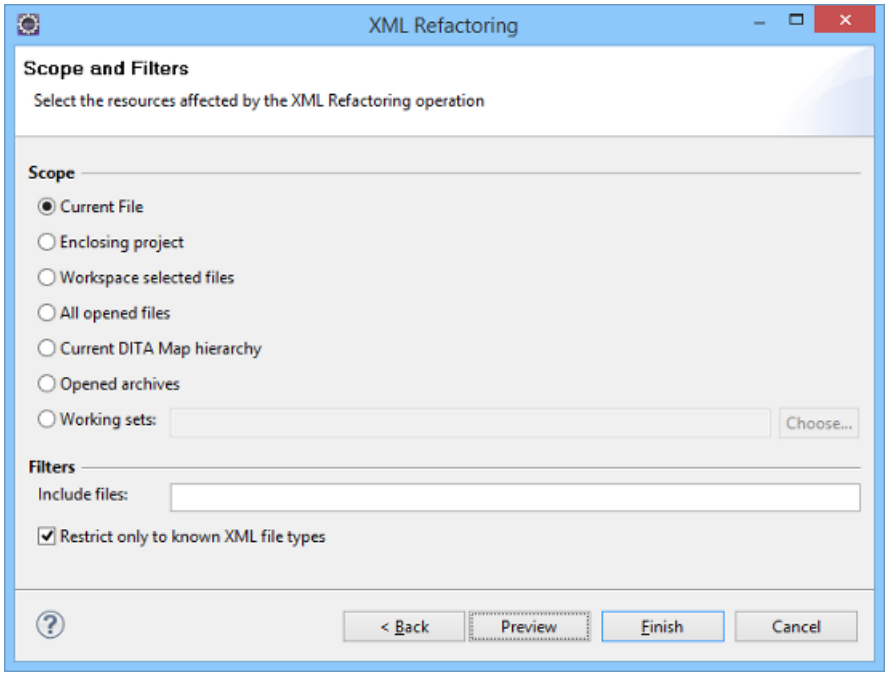


Scope and Filters

The last wizard page allows you to select the set of files that represent the input of the operation. You can select from predefined resource sets (such as the current file, your whole project, the current DITA map hierarchy, etc.) or you can define your own set of resources by creating a working set.

The **Filters** section includes the following options:

- **Include files** - Allows you to filter the selected resources by using a file pattern. For example, to restrict the operation to only analyze build files you could use `build*.xml` for the file pattern.
- **Restrict only to known XML file types** - When enabled, only resources with a known XML file type will be affected by the operation.



If an operation takes longer than expected you can use the **Stop** button in the progress bar to cancel the operation.

Note: It is recommended that you use the **Preview** button to review all the changes that will be made by the refactoring operation before applying the changes.

Warning: After clicking the **Finish** button, the operation will be processed and Oxygen XML Editor plugin provides no automatic means for reverting the operations. Any **Undo** action will only revert changes on the current document.

Predefined Refactoring Operations

The XML Refactoring tool includes a variety of predefined operations that can be used for common refactoring tasks. They are grouped by category in the **Refactoring operations** wizard page. You can also access the operations from the **Refactoring** submenu in the contextual menu of **Author** or **Text** mode. The operations are also grouped by category in this submenu (**Attributes**, **Elements**, and **Fragments**). When selecting the operations from the contextual menu, Oxygen XML Editor plugin considers the editing context to get the names and namespaces of the current element or attribute, and uses this information to preconfigure some of the parameter values for the selected refactoring operation.

The following predefined operations are available:

Refactoring Operations for *Attributes*

Add/Change attribute

Use this operation to change the value of an attribute or insert a new one. To perform this operation, specify the following parameters:

- The parent **Element** of the attribute to be changed, in the form of a local name from any namespace, a local name with a namespace prefix, or an XPath expression.
- The **Local name**, **Namespace**, and **Value** of the affected attribute.
- One of the following choices for the **Operation mode** in the *Options* section:
  - **Add the attribute in the parent elements where it is missing**
  - **Change the value in the parent elements where the atrribute already exists**
  - **Both**

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Delete attribute

Use this operation to remove one or more attributes. To perform this operation, specify the following parameters:

- The parent **Element** of the attribute to be deleted, in the form of a local name from any namespace, a local name with a namespace prefix, or an XPath expression.
- The name of the **Attribute** to be deleted.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Rename attribute

Use this operation to rename an attribute. Specify the following parameters in the **Rename attribute** dialog box:

- The parent **Element** of the attribute to be renamed, in the form of a local name from any namespace, a local name with a namespace prefix, or an XPath expression.
- The name of the **Attribute** to be renamed.
- **New local name** of the attribute.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Replace in attribute value

Use this operation to search for a text fragment inside an attribute value and change the fragment to a new value. To perform this operation, specify the following parameters:

- The parent **Element** of the attribute to be modified, in the form of a local name from any namespace, a local name with a namespace prefix, or an XPath expression.
- The name of the **Attribute** to be modified.
- The text fragments to **Find**. You can use Perl-like regular expressions when specifying the text to find.
- The text fragment to **Replace with**. This parameter can bind regular expression capturing groups (`$1, $2`, etc.) from the find pattern.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Refactoring Operations for *Elements*

Delete element

Use this operation to delete elements. To perform this operation, specify the following parameter:

- The target **Element** to be deleted, in the form of a local name from any namespace, a local name with a namespace prefix, or an XPath expression.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Delete element content

Use this operation to delete the content of elements. To perform this operation, specify the following parameter:

- The target **Element** in which its content is to be deleted, in the form of a local name from any namespace, a local name with a namespace prefix, or an XPath expression.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Insert element

Use this operation to insert new elements. To perform this operation, specify the following parameters:

- The **Local name** of the element to be inserted.
- The **Namespace** of the element to be inserted.
- The **XPath** location of an existing element where the new element will be inserted, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- The **Position**, in relation to the specified existing element, where the new element will be inserted. The possible selections in the drop-down menu are: **After**, **Before**, **First child**, or **Last child**.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Rename element

Use this operation to rename elements. To perform this operation, specify the following parameters:

- The **Target elements (XPath)** to be renamed, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- The **New local name** of the element.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Unwrap element

Use this operation to remove the surrounding tags of elements, while keeping the content unchanged. To perform this operation, specify the following parameters:

- The **Target elements (XPath)** for which its surrounding tags will be removed, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Wrap element

Use this operation to surround elements with element tags. To perform this operation, specify the following parameters:

- The **Target elements (XPath)** to be surrounded with tags, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- The **Local name** of the *Wrapper element*.
- The **Namespace** of the *Wrapper element*.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Wrap element content

Use this operation to surround the content of elements with element tags. To perform this operation, specify the following parameters:

- The **Target elements (XPath)** to surround its content with tags, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- The **Local name** of the *Wrapper element* in which its content will be wrapped.
- The **Namespace** of the *Wrapper element* in which its content will be wrapped.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Refactoring Operations for *Fragments*

Insert XML fragment

Use this operation to insert an XML fragment. To perform this operation, specify the following parameters:

- The **XML Fragment** to be inserted.
- The **XPath** location of an existing element where the fragment will be inserted, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- The **Position**, in relation to the specified existing element, where the fragment will be inserted. The possible selections in the drop-down menu are: **After**, **Before**, **First child**, or **Last child**.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Replace element content with XML fragment

Use this operation to replace the content of elements with an XML fragment. To perform this operation, specify the following parameters:

- The **Target elements (XPath)** for which its content will be replaced, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- The **XML Fragment** with which to replace the content of the target element.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Replace element with XML fragment

Use this operation to replace elements with an XML fragment. To perform this operation, specify the following parameters:

- The **Target elements (XPath)** to be replaced, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- The **XML Fragment** with which to replace the target element.

Tip: Use the link provided in the lower part of the wizard to open the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

Additional Notes

Note: There are some operations that allows `<ANY>` for the **local name** and **namespace** parameters. This value can be used to select an element or attribute regardless of its local name or namespace. Also, the `<NO_NAMESPACE>` value can be used to select nodes that do not belong to a namespace.

Note: Some operations have parameters that accept XPath expressions to match elements or attributes. In these XPath expressions you can only use the prefixes declared in the ***Options*** > ***Preferences*** > ***XML*** > ***XSLT-FO-XQUERY*** > ***XPath*** page. This preferences page can be easily opened by clicking the link in the note (**Each prefix used in an XPath expression must be declared in the Default prefix-namespace mappings section**) at the bottom of the **Configure Operation Parameters** wizard page.

Custom Refactoring Operations

If none of the predefined operations will help you accomplish a particular refactoring task, you can create a custom operation that is specific to your needs. For example, if you want to convert an attribute to an element and insert the element as the first child of the parent element, a custom refactoring operation needs to be created.

Note: The custom refactoring operations are only available in the Enterprise edition.

An XML Refactoring operation is defined as a pair of resources:

- An *XQuery Update script* or *XSLT stylesheet* that Oxygen XML Editor plugin will run in order to refactor the XML files.
- An *XML Operation Descriptor* file that contains information about the operation, such as the name, description, and parameters.

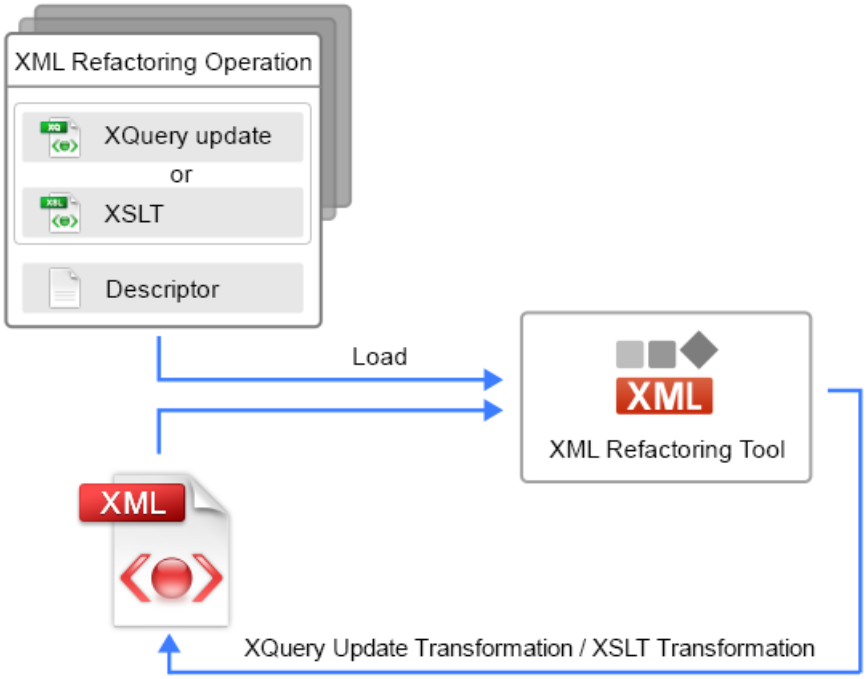


Figure 121: Diagram of an XML Refactoring Operation

All the defined custom operations are loaded by the XML Refactoring tool and presented in *the* ***Refactoring operations*** *wizard page*, along with the predefined built-in operations.

After the user chooses an operation and specifies its parameters, Oxygen XML Editor plugin processes an XQuery Update or XSLT transformation over the input file. This transformation is executed in a *safe mode*, which implies the following:

- When loading the document:
  - The XInclude mechanism is disabled. This means that the resources included by using XInclude will not be visible in the transformation.
  - The DTD entities will be processed without being expanded.
  - The associated DTD will be not loaded, so the default attributes declared in the DTD will not be visible in the transformation.
- When saving the updated XML document:
  - The `DOCTYPE` will be preserved.
  - The DTD entities will be preserved as they are in the original document when the document is saved.
  - The attribute values will be kept in their original form without being normalized.
  - The spaces between attributes are preserved. Basically, the spaces are lost by a regular XML serialization since they are not considered important.

The result of this transformation overrides the initial input file.

Note: To achieve some of the previous goals, the XML Refactoring mechanism adds several attributes that are interpreted internally. The attributes belong to the `http://oxygenxml.com/app/xml_refactory/additional_attributes` namespace. These attributes should not be taken into account when processing the input XML document, since they are discarded when the transformed document is serialized.

Restriction: *Comments* or *processing instructions* that are in any node before or after the root element cannot be modified by an XML Refactoring operation. In other words, XML Refactoring operations can only be performed on *comments* or *processing instructions* that are inside the root element.

Creating a Custom Refactoring Operation

To create a custom refactoring operation, follow these steps:

1. *Create an XQuery Update script or XSLT file*.
2. *Create an **XML Refactoring Operation Descriptor** file*.
3. Store both files in *one of the locations that Oxygen XML Editor plugin* scans when loading the custom operations.

   Once you run the **XML Refactoring** tool again, the custom operation appears in *the **Refactoring operations** wizard page*.

Custom Refactoring Script

The first step in creating a custom refactoring operation is to create an *XQuery Update script* or *XSLT stylesheet* that is needed to process the refactoring operations. The easiest way to create this script file is to use the **New** document wizard to create a new **XQuery** or **XSLT** file and you can use *our examples* to help you with the content.

There are cases when it is necessary to add parameters in the *XQuery script* or *XSLT stylesheet*. For instance, if you want to rename an element, you may want to declare an external parameter associated with the name of the element to be renamed. To allow you to specify the value for these parameters, they need to be declared in *the refactoring operation descriptor file* that is associated with this operation.

Note: The XQuery Update processing is disabled by default in Oxygen XML Editor plugin. Thus, if you want to create or edit an XQuery Update script you have to enable this facility by creating an *XQuery transformation scenario* and choose **Saxon EE** as the transformation engine. Also, you need to make sure the **Enable XQuery update** option is enabled in the *Saxon processor advanced options*.

Note: If you are using an XSLT file, XPath expressions that are passed as parameters will automatically be rewritten to conform with the mapping of the namespace prefixes declared in the ***XML /XSLT-FO-XQuery / XPath** preferences page*.

The next step in creating a custom refactoring operation is to *create a custom operation descriptor file*.

Custom Refactoring Operation Descriptor File

The second step in creating a custom refactoring operation is to create an operation descriptor file. The easiest way to do this is to use the **New** document wizard and choose the **XML Refactoring Operation Descriptor** template.

Introduction to the Descriptor File

This file contains information (such as `name`, `description`, and `id`) that is necessarily when loading an XML Refactoring operation . It also contains the path to the XQuery Update script or XSLT stylesheet that is associated with the particular operation through the `script` element.

You can specify a `category` for your custom operations to logically group certain operations. The `category` element is optional and if it is not included in the descriptor file, the default name of the category for the custom operations is *Other operations*.

The descriptor file is edited and validated against the following schema:
`frameworks/xml_refactoring/operation_descriptor.xsd`.

Declaring Parameters in the Descriptor File

If the XQuery Update script or XSLT stylesheet includes parameters, they should be declared in the **parameters** section of the descriptor file. All the parameters specified in this section of the descriptor file will be displayed in the **XML Refactoring** tool within *the **Configure Operation Parameters** wizard page* for that particular operation.

The value of the first `description` element in the **parameters** section will be displayed at the top of *the **Configure Operation Parameters** wizard page*.

To declare a parameter, specify the following information:

- **`label`** - This value is displayed in the user interface for the parameter.
- **`name`** - The parameter name used in the XQuery Update script or XSLT stylesheet and it should be the same as the one declared in the script.
- **`type`** - Defines the type of the parameter and how it will be rendered. There are several types available:
  - `TEXT` - Generic type used to specify a simple text fragment.
  - `XPATH` - Type of parameter whose value is an XPATH expression. For this type of parameter, Oxygen XML Editor plugin will use a text input with corresponding content completion and syntax highlighting.

    Note: The value of this parameter is transferred as plain text to the XQuery Update or XSLT transformation without being evaluated. You should evaluate the XPath expression inside the XQuery Update script or XSLT stylesheet. For example, you could use the `saxon:evaluate` Saxon extension function.

    Note: A relative XPath expression is converted to an absolute XPath expression by adding `//` before it (`//XPathExp`). This conversion is done before transferring the XPath expression to the XML refactoring engine.

    Note: When writing XPath expressions, you can only use prefixes declared in the ***Options*** > ***Preferences*** > ***XML*** > ***XSLT-FO-XQUERY*** > ***XPath*** options page.

  - `NAMESPACE` - Used for editing namespace values.
  - `REG_EXP_FIND` - Used when you want to match a certain text by using Perl-like regular expressions.
  - `REG_EXP_REPLACE` - Used along with `REG_EXP_FIND` to specify the replacement string.
  - `XML_FRAGMENT` - This type is used when you want to specify an XML fragment. For this type, Oxygen XML Editor plugin will display a text area specialized for inserting XML documents.
  - `NC_NAME` - The parameter for `NC_NAME` values. It is useful when you want to specify the local part of a *QName* for an element or attribute.
  - `BOOLEAN` - Used to edit boolean parameters.
  - `TEXT_CHOICE` - It is useful for parameters whose value should be from a list of possible values. Oxygen XML Editor plugin renders each possible value as a radio button option.
- **`description`** - The description of the parameter. It is used by the application to display a tooltip when you hover over the parameter.
- **`possibleValues`** - Contains the list with possible values for the parameter and you can specify the default value, as in the following example:

```
<possibleValues onlyPossibleValuesAllowed="true">
    <value name="before">Before</value>
    <value name="after"default="true">After</value>
    <value name="firstChild">First child</value>
    <value name="lastChild">Last child</value>
</possibleValues>
```

Specialized Parameters to Match Elements or Attributes

If you want to match elements or attributes, you can use some specialized parameters, in which case Oxygen XML Editor plugin will propose all declared elements or attributes based on the schema associated with the currently edited file. The following specialized parameters are supported:

`elementLocation`

This parameter is used to match elements. For this type of parameter, the application displays a text field where you can enter the element name or an XPath expression. The text from the `label` attribute is displayed in the application as the label of the text field. The `name` attribute is used to specify the name of the parameter from the XQuery

Update script or XSLT stylesheet. If the value of the `useCurrentContext` attribute is set to `true`, the element name from the cursor position is used as proposed values for this parameter.

Example of an `elementLocation`:

```
<elementLocation name="elem_loc" useCurrentContext="false">
                <element label="Element location">
                    <description>Element location description.</description>
                </element>
</ ElementLocation>
```

attributeLocation

This parameter is used to match attributes. For this type of parameter, the application displays two text fields where you can enter the parent element name and the attribute name (both text fields accept XPath expressions for a finer match). The text from the `label` attributes is displayed in the application as the label of the associated text fields. The `name` attribute is used to specify the name of the parameter from the XQuery Update script or XSLT stylesheet. The value of this parameter is an XPath expression that is computed by using the values of the expression from the `element` and `attribute` text fields. For example, if `section` is entered for the element and a `title` is entered for the attribute, the XPath expression would be computed as `//section/@title`. If the value of the `useCurrentContext` attribute is set to `true`, the element and attribute name from the cursor position is used as proposed values for the operation parameters.

Example of an `attributeLocation`:

```
<attributeLocation name="attr_xpath" useCurrentContext="true">
                <element label="Element path">
                    <description>Element path description.</description>
                </element>
                <attribute label="Attribute" >
                    <description>Attribute path description.</description>
                </attribute>
</ AttributeLocation>
```

elementParameter

This parameter is used to specify elements by local name and namespace. For this type of parameter, the application displays two combo boxes with elements and namespaces collected from the associated schema of the currently edited file. The text from the `label` attribute is displayed in the application as label of the associated combo. The `name` attribute is used to specify the name of the parameter from the XQuery Update script or XSLT stylesheet. If you specify the `allowsAny` attribute, the application will propose <*ANY*> as a possible value for the **Name** and **Namespace** combo boxes. You can also use the `useCurrentContext` attribute and if its value is set to `true`, the element name and namespace from the cursor position is used as proposed values for the operation parameters.

Example of an `elementParameter`:

```
<elementParameter id="elemID">
    <localName label="Name" name="element_localName" allowsAny="true" useCurrentContext="true">
        <description>The local name of the attribute's parent element.</description>
    </localName>
    <namespace label="Namespace" name="element_namespace" allowsAny="true">
        <description>The local name of the attribute's parent element</description>
    </namespace>
</elementParameter>
```

attributeParameter

This parameter is used to specify attributes by local name and namespace. For this type of parameter, the application displays two combo boxes with attributes and their namespaces collected from the associated schema of the currently edited file. The text from the `label` attribute is displayed in the application as the label of the associated combo box. You can also use the `useCurrentContext` attribute and if its value is set to `true`, the attribute name and namespace from the cursor position is used as proposed values for the operation parameters.

Note: An `attributeParameter` is dependant upon an `elementParameter`. The list of attributes and namespaces are computed based on the selection in the `elementParameter` combo boxes.

Example of an `attributeParameter`:

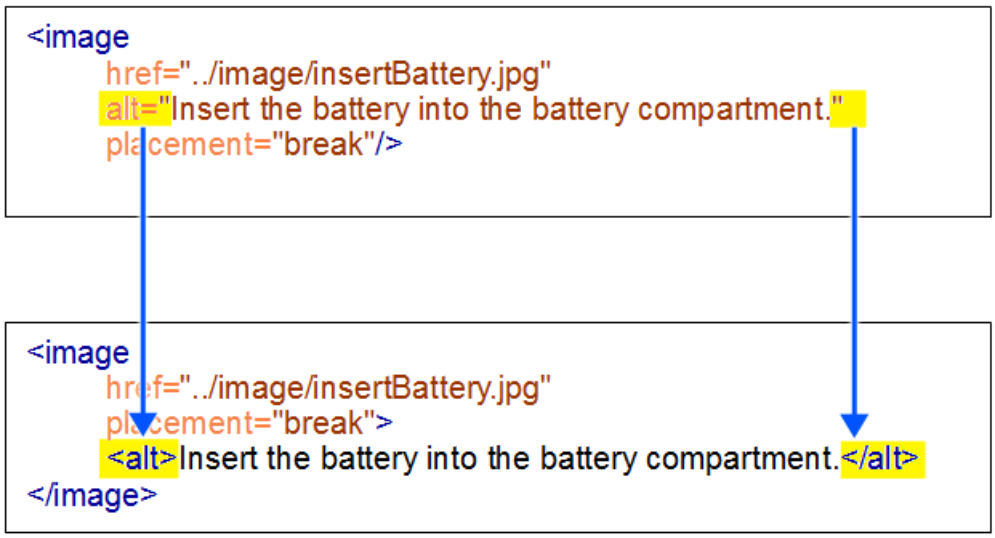
```
<attributeParameter dependsOn="elemID">
    <localName label="Name" name="attribute_localName" useCurrentContext="true">
        <description>The name of the attribute to be converted.</description>
    </localName>
    <namespace label="Namespace" name="attribute_namespace" allowsAny="true">
        <description>The namespace of the attribute to be converted.</description>
    </namespace>
</attributeParameter>
```

Note: All predefined operations are loaded from the `[OXYGEN_DIR]/refactoring` folder.

Example of an XML Refactoring Operation

To demonstrate creating a custom operation, consider that we have a task where we need to convert an attribute into an element and insert it inside another element. A specific example would be if you have a project with a variety of `image` elements where a deprecated `alt` attribute was used for the description and you want to convert all instances of that attribute into an element with the same name and insert it as the first child of the `image` element.

Thus, our task is to convert this attribute into an element with the same name and insert it as the first child of the image element.



A new custom XML refactoring operation requires:

- An *XQuery Update script* or *XSLT stylesheet*.
- An *XML Refactoring operation descriptor file* that contains the path to the XQuery Update script or XSLT stylesheet.

Example of an XQuery Update Script for Creating a Custom Operation to *Convert an Attribute to an Element*

The XQuery Update script does the following:

- Iterates over all elements from the document that have the specified local name and namespace.
- Finds the attribute that will be converted to an element.
- Computes the *QName* of the new element to be inserted and inserts it as the first child of the parent element.

```
(:
    XQuery document used to implement 'Convert attribute to element' operation from XML Refactoring tool.
:)

declare namespace output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method   "xml";
declare option output:indent   "no";
```

```
(: Local name of the attribute's parent element. :)
declare variable $element_localName as xs:string external;

(: Namespace of the attribute's parent element. :)
declare variable $element_namespace as xs:string external;

(: The local name of the attribute to be converted :)
declare variable $attribute_localName as xs:string external;

(: The namespace of the attribute to be converted :)
declare variable $attribute_namespace as xs:string external;

(: Local name of the new element. :)
declare variable $new_element_localName as xs:string external;

(: Namespace of the new element. :)
declare variable $new_element_namespace as xs:string external;

(: Convert attribute to element:)
for $node in //*
(: Find the attribute to convert :)
let $attribute :=
    $node/@*[local-name() = $attribute_localName and
    ($attribute_namespace = '<ANY>' or $attribute_namespace = namespace-uri())]

(: Compute the prefix for the new element to insert :)
let $prefix :=
    for $p in in-scope-prefixes($node)
      where $new_element_namespace = namespace-uri-for-prefix($p, $node)
return $p

(: Compute the qname for the new element to insert :)
let $new_element_qName :=
    if (empty($prefix) or $prefix[1] = '') then $new_element_localName
    else $prefix[1] || ':' || $new_element_localName

  where ('<ANY>' = $element_localName or local-name($node) = $element_localName) and
        ($element_namespace = '<ANY>' or $element_namespace = namespace-uri($node))

  return
      if (exists($attribute)) then
        (insert node element {QName($new_element_namespace, $new_element_qName)}
        {string($attribute)} as first into $node,
        delete node $attribute)
        else ()
```

Example of an XSLT Script for Creating a Custom Operation to *Convert an Attribute to an Element*

The XSLT stylesheet does the following:

- Iterates over all elements from the document that have the specified local name and namespace.
- Finds the attribute that will be converted to an element.
- Adds the new element as the first child of the parent element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs"
    xmlns:xr="http://www.oxygenxml.com/ns/xmlRefactoring"
    version="2.0">

    <xsl:import href="http://www.oxygenxml.com/ns/xmlRefactoring/resources/commons.xsl"/>

    <xsl:param name="element_localName" as="xs:string" required="yes"/>
    <xsl:param name="element_namespace" as="xs:string" required="yes"/>
    <xsl:param name="attribute_localName" as="xs:string" required="yes"/>
    <xsl:param name="attribute_namespace" as="xs:string" required="yes"/>
    <xsl:param name="new_element_localName" as="xs:string" required="yes"/>
    <xsl:param name="new_element_namespace" as="xs:string" required="yes"/>

    <xsl:template match="node() | @*">
        <xsl:copy>
            <xsl:apply-templates select="node() | @*"/>
        </xsl:copy>
    </xsl:template>
    <xsl:template match="//*[xr:check-local-name($element_localName, ., true()) and
        xr:check-namespace-uri($element_namespace, .)]">

        <xsl:variable name="attributeToConvert"
            select="@*[xr:check-local-name($attribute_localName, ., true()) and
            xr:check-namespace-uri($attribute_namespace, .)]"/>

        <xsl:choose>
            <xsl:when test="empty($attributeToConvert)">
```

```
                <xsl:copy>
                    <xsl:apply-templates select="node() | @*"/>
                </xsl:copy>
            </xsl:when>
            <xsl:otherwise>
                <xsl:copy>
                    <xsl:for-each select="@*[empty(. intersect $attributeToConvert)]">
                        <xsl:copy-of select="."/>
                    </xsl:for-each>
                    <!-- The new element namespace -->
                    <xsl:variable name="nsURI" as="xs:string">
                        <xsl:choose>
                            <xsl:when test="$new_element_namespace eq $xr:NO-NAMESPACE">
                                <xsl:value-of select="''"/>
                            </xsl:when>
                            <xsl:otherwise>
                                <xsl:value-of select="$new_element_namespace"/>
                            </xsl:otherwise>
                        </xsl:choose>
                    </xsl:variable>
                    <xsl:element name="{$new_element_localName}" namespace="{$nsURI}">
                        <xsl:value-of select="$attributeToConvert"/>
                    </xsl:element>
                    <xsl:apply-templates select="node()"/>
                </xsl:copy>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>
</xsl:stylesheet>
```

Note: The XSLT stylesheet imports a module library that contains utility functions and variables. The location of this module is resolved via an XML catalog set in the *XML Refactoring* framework.

Example of an Operation Descriptor File for Creating a Custom Operation to *Convert an Attribute to an Element*

After you have developed the XQuery script or XSLT stylesheet, you have to create an XML Refactoring operation descriptor. This descriptor is used by the application to load the operation details such as name, description, or parameters.

```
<?xml version="1.0" encoding="UTF-8"?>

<refactoringOperationDescriptor
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.oxygenxml.com/ns/xmlRefactoring"
    id="convert-attribute-to-element"
    name="Convert attribute to element">
   <description>Converts the specified attribute to an element. The new element will be inserted as first child
 of the attribute's parent element.</description>
    <!-- For the XSLT stylesheet option uncomment the following line and comment the line referring the XQuery
 Update script -->
    <!-- <script type="XSLT" href="convert-attribute-to-element.xsl"/> -->
    <script type="XQUERY_UPDATE" href="convert-attribute-to-element.xq"/>
    <parameters>
        <description>Specify the attribute to be converted to element.</description>
        <section label="Parent element">
            <elementParameter id="elemID">
                <localName label="Name" name="element_localName" allowsAny="true">
                    <description>The local name of the attribute's parent element.</description>
                </localName>
                <namespace label="Namespace" name="element_namespace" allowsAny="true">
                    <description>The local name of the attribute's parent element</description>
                </namespace>
            </elementParameter>
        </section>
        <section label="Attribute">
            <attributeParameter dependsOn="elemID">
                <localName label="Name" name="attribute_localName">
                    <description>The name of the attribute to be converted.</description>
                </localName>
                <namespace label="Namespace" name="attribute_namespace" allowsAny="true">
                    <description>The namespace of the attribute to be converted.</description>
                </namespace>
            </attributeParameter>
        </section>
        <section label="New element">
            <elementParameter>
                <localName label="Name" name="new_element_localName">
                    <description>The name of the new element.</description>
                </localName>
                <namespace label="Namespace" name="new_element_namespace">
                    <description>The namespace of the new element.</description>
                </namespace>
```

```
            </elementParameter>
        </section>
    </parameters>
</refactoringOperationDescriptor>
```

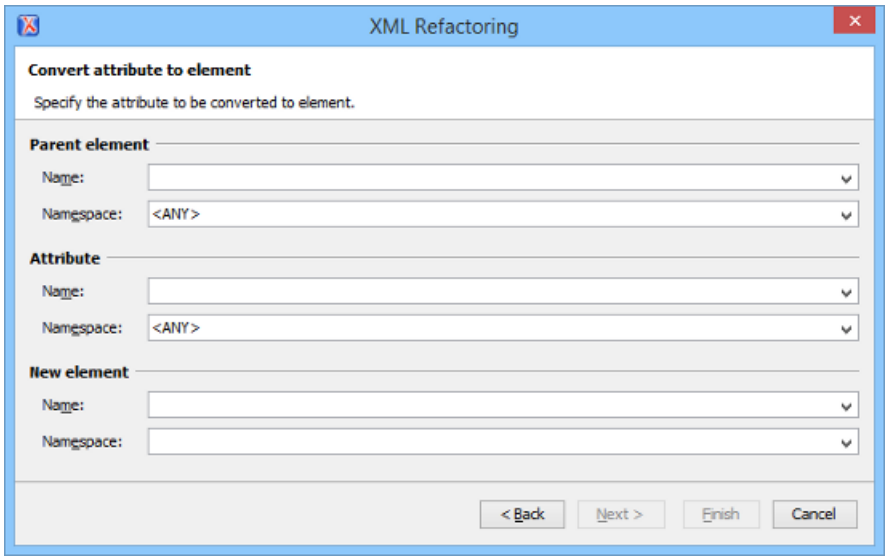
Note: If you are using an XSLT file, the line with the `script` element would look like this:

```
<script type="XSLT" href="convert-attribute-to-element.xsl"/>
```

Results

After you have created these files, copy them into a folder *scanned by Oxygen XML Editor plugin when it loads the custom operation*. When the XML Refactoring tool is started again, you will see the created operation.

Since various parameters can be specified, this custom operation can also be used for other similar tasks. The following image shows the parameters that can be specified in our example of the custom operation to convert an attribute to an element:



Storing and Sharing Refactoring Operations

Oxygen XML Editor plugin scans the following locations when looking for XML Refactoring operations to provide flexibility:

- A `refactoring` folder, created inside a directory that is associated to a *framework* you are customizing.
- Any folder. In this case, you need to *open the **Preferences** dialog box*, go to **XML** > **XML Refactoring**, and specify the same folder in the **Load additional refactoring operations from** text box.
- The `refactoring` folder from the Oxygen XML Editor plugin installation directory (`[OXYGEN INSTALLATION DIRECTORY]/refactoring/`).

Sharing Custom Refactoring Operations

The purpose of Oxygen XML Editor plugin scanning multiple locations for the XML Refactoring operations is to provide more flexibility for developers who want to share the refactoring operations with the other team members. Depending on your particular use case, you can attach the custom refactoring operations to other resources, such as frameworks or projects.

After storing custom operations, you can share them with other users by sharing the resources.

Localizing XML Refactoring Operations

Oxygen XML Editor plugin includes localization support for the XML refactoring operations.

The translation keys for the built-in refactoring operations are located in `[OXYGEN INSTALLATION DIRECTORY]/refactoring/i18n/translation.xml`.

The localization support is also available for custom refactoring operations. The following information can be translated:

- The operation `name`, `description`, and `category`.
- The `description` of the `parameters` element.
- The `label`, `description`, and `possibleValues` for each `parameter`.

Translated refactoring information uses the following form:

```
${i18n(translation_key)}
```

Oxygen XML Editor plugin scans the following locations to find the `translation.xml` files that are used to load the translation keys:

- A `refactoring/i18n` folder, created inside a directory that is associated to a customized *framework*.
- A `i18n` folder, created inside a directory that is associated to a customized *framework*.
- An `i18n` folder inside any specified folder. In this case, you need to *open the **Preferences** dialog box*, go to **XML** > **XML Refactoring**, and specify the folder in the **Load additional refactoring operations from** text box.
- The `refactoring/i18n` folder from the Oxygen XML Editor plugin installation directory (`[OXYGEN INSTALLATION DIRECTORY]/refactoring/i18n`).

Example of a Refactoring Operation Descriptor File with *i18n* Support

```
<?xml version="1.0" encoding="UTF-8"?>

<refactoringOperationDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://oxygenxml.com/app/xml_refactory
http://oxygenxml.com/app/xml_refactory/operation_descriptor.xsd"
    xmlns="http://oxygenxml.com/app/xml_refactory" id="remove_text_content"
name="${i18n(Remove_text_content)}">
    <description>${i18n(Remove_text_content_description)}</description>
    <script type="XQUERY_UPDATE" href="remove_text_content.xq"/>
    <parameters>
        <description>${i18n(parameters_description)}</description>
        <parameter label="${i18n(Element_name)}" name="element_localName" type="NC_NAME">
            <description>${i18n(Element_name_descriptor)}</description>
            <possibleValues>
                <value default="true" name="value1">${i18n(value_1)}</value>
                <value name="value2">${i18n(value_2)}</value>
            </possibleValues>
        </parameter>
    </parameters>
</refactoringOperationDescriptor>
```

Editing XSLT Stylesheets

This section explains the features of the XSLT editor.

To watch our video demonstration about basic XSLT editing and transformation scenarios in Oxygen XML Editor plugin, go to *http://oxygenxml.com/demo/XSL_Editing.html*.

Validating XSLT Stylesheets

Oxygen XML Editor plugin performs the validation of XSLT documents with the help of an XSLT processor *that you can configure in the preferences pages* according to the XSLT version. For XSLT 1.0, the options are: Xalan, Saxon 6.5.5, Saxon 9.6.0.7 and *a JAXP transformer specified by the main Java class.* For XSLT 2.0, the options are: Saxon 9.6.0.7 and *a JAXP transformer specified by the main Java class.* For XSLT 3.0, the options are Saxon 9.6.0.7 and *a JAXP transformer specified by the main Java class.*

Creating a Validation Scenario for XSLT Stylesheets

You can validate an XSLT document using the engine defined in the transformation scenario, or a custom validation scenario. If you choose to validate using the engine from transformation scenario, and a transformation scenario is not

associated with the current document or the engine has no validation support, the default engine is used. To set the default engine, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XSLT**.

You can also create new validation scenarios or edit existing ones, and you can add jars and classes that contain extension functions. To create or edit a validation scenario for an XSLT stylesheet, follow these steps:

1. With the XSLT file opened in Oxygen XML Editor plugin, select the **Configure Validation Scenario(s)** from the **XML** menu, or the toolbar, or from the **Validate** submenu when invoking the contextual menu on the XSLT file in the **Navigator** view.
   The **Configure Validation Scenario(s)** dialog box is displayed. It contains the existing scenarios, organized in categories depending on the type of file they apply to. You can use the options in the **Settings** drop-down menu to filter which scenarios are shown.
2. To edit an existing scenario, select the scenario and press the **Edit** button. If you try to edit one of the *read-only* predefined scenarios, Oxygen XML Editor plugin creates a customizable duplicate (you can also use the **Duplicate** button).
3. To add a new scenario, press the **New** button.
   The **New scenarios** dialog box is displayed. It lists all validation units of the scenario.

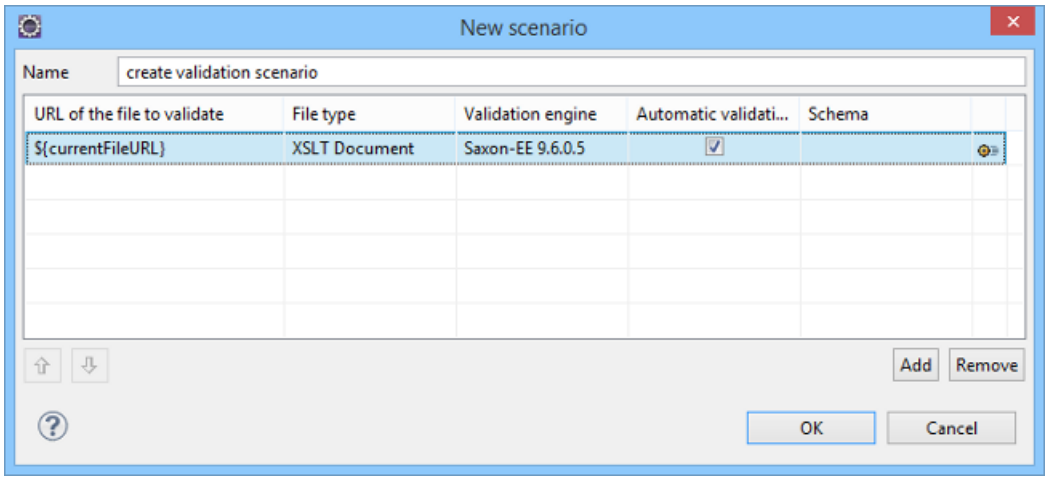


Figure 122: Add / Edit a Validation Unit

4. Configure the following information in this dialog box:
   a) **Name** - The name of the validation scenario.
   b) **URL of the file to validate** - In most cases leave this field as the default selection (the URL of the current file). If you want to specify a different URL, click its cell and enter the URL in the text field, select it from the drop-down list, or use the **Browse** drop-down menu or **Insert Editor Variable** button.
   c) **File type** - The file type should be `XSLT Document`.
   d) **Validation engine** - Click the cell to select a validation engine. You must select an engine to be able to add or edit extensions.
   e) **Automatic validation** - If this option is checked, the validation operation defined by this row is also used by *the automatic validation feature*.
5. To add or edit extensions, click the **Edit extensions** button. This button is only available if the **File type** is set as `XSLT Document` and a **Validation engine** is chosen.
   The **Libraries** dialog box is opened. It is used to specify the jars and classes that contain extension functions called from the XSLT file of the current validation scenario. They will be searched, in the specified extensions, in the order displayed in this dialog box. To change the order of the items, select the item and press the **Move up** or **Move down** buttons.
6. Press **OK** to close the **New scenario** dialog box.

The newly created validation scenario is now included in the list of scenarios in the **Configure Validation Scenario(s)** dialog box. You can select the scenario in this dialog box to associate it with the current XSLT document and press the **Apply associated** button to run the validation scenario.

Validating XSLT Stylesheets with Custom Engines

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

By default, there are two validators that are configured for XSLT stylesheets:

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

Editing XSLT Stylesheets in the Master Files Context

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

You cat set a main XSLT stylesheet either using the *master files support from the **Navigator** view*, or using a validation scenario.

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

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

- Correct validation of a module in the context of a larger stylesheet structure.
- **Content Completion Assistant** displays all components valid in the current context.
- The **Outline** displays the components collected from the entire stylesheet structure.

To watch our video demonstration about editing XSLT stylesheets in the master files context, go to *http://oxygenxml.com/demo/MasterFilesSupport.html*.

Syntax Highlight

The XSL editor renders the CSS and JS scripts, and XPath expressions with dedicated coloring schemes. To customize the coloring schemes, *open the **Preferences** dialog box* and go to **Editor** > **Syntax Highlight**.

Content Completion in XSLT Stylesheets

The items in the list of proposals offered by the **Content Completion Assistant** are context-sensitive. The proposed items are valid at the current cursor position. You can enhance the list of proposals by specifying an additional schema. This schema is *defined by the user in the Content Completion / XSL preferences* page and can be: XML Schema, DTD, RELAX NG schema, or NVDL schema.

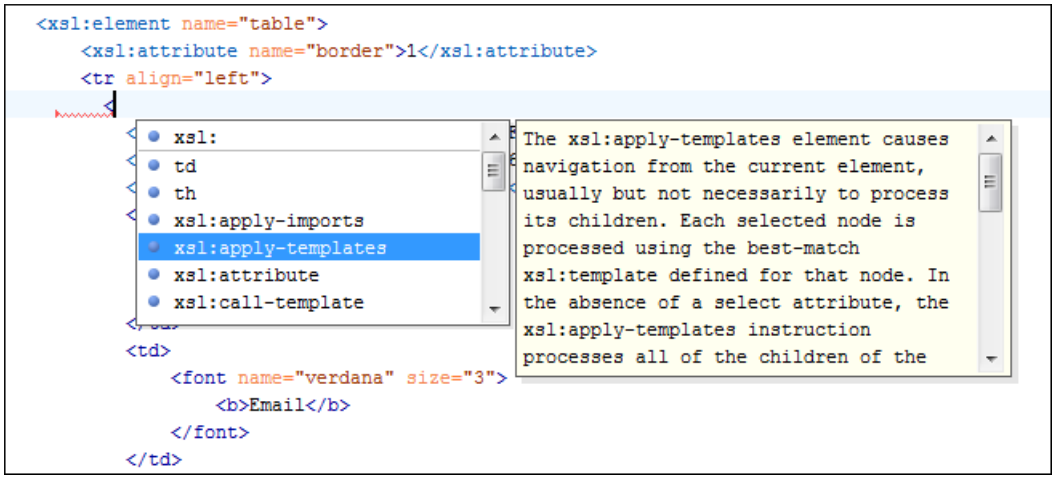


Figure 123: XSLT Content Completion Window

The **Content Completion Assistant** proposes numerous item types (such as templates, variables, parameters, keys, etc.) that are defined in the current stylesheet, and in the imported and included XSLT stylesheets. The **Content Completion Assistant** also includes *code templates that can be used to quickly insert code fragments* into stylesheets.

Note: For XSL and XSD resources, the **Content Completion Assistant** collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to *Defining Master Files at Project Level*.

The extension functions built in the Saxon 6.5.5 and 9.6.0.7 transformation engines are presented in the content completion list only if the Saxon namespace (*http://saxon.sf.net* for XSLT version 2.0 / 3.0 or *http://icl.com/saxon* for XSLT version 1.0) is declared and one of the following conditions is true:

- The edited file has a transformation scenario that uses as transformation engine Saxon 6.5.5 (for XSLT version 1.0), Saxon 9.6.0.7 PE or Saxon 9.6.0.7 EE (for XSLT version 2.0 / 3.0).
- The edited file has a validation scenario that uses as validation engine Saxon 6.5.5 (for version 1.0), Saxon 9.6.0.7 PE or Saxon 9.6.0.7 EE (for version 2.0 / 3.0).
- The validation engine specified in *Options* page is Saxon 6.5.5 (for version 1.0), Saxon 9.6.0.7 PE or Saxon 9.6.0.7 EE (for version 2.0 / 3.0).

Additionally. the Saxon-CE-specific extension functions and instructions are presented in the Content Completion Assistant's proposals list only if the `http://saxonica.com/ns/interactiveXSLT` namespace is declared.

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

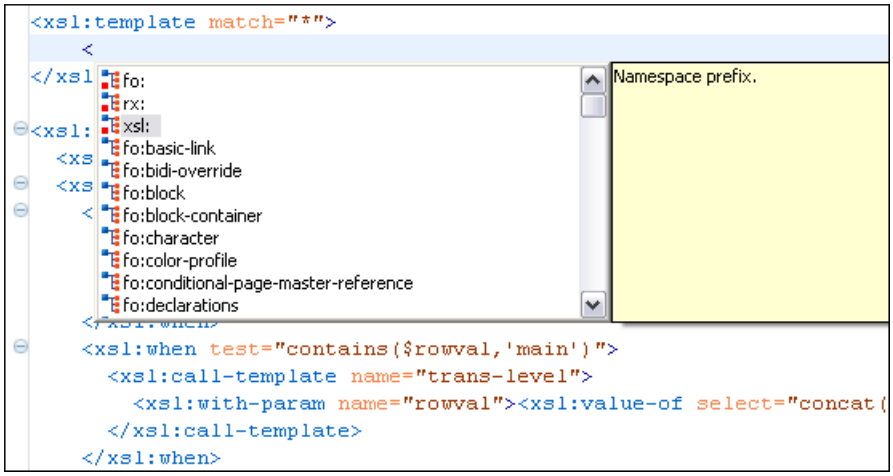


Figure 124: Namespace Prefixes in the Content Completion Window

For the common namespaces like XSL namespace (`http://www.w3.org/1999/XSL/Transform`), XML Schema namespace (`http://www.w3.org/2001/XMLSchema`) or Saxon namespace (`http://icl.com/saxon` for version 1.0, `http://saxon.sf.net/` for version 2.0 / 3.0), Oxygen XML Editor plugin provides an easy mode to declare them by proposing a prefix for these namespaces.

Content Completion in XPath Expressions

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

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

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

Content completion for XPath expressions is started:

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

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

- The context of the current XSLT element.
- The XML document associated with the edited stylesheet in the stylesheet transformation scenario.
- The XSLT version of the stylesheet (1.0, 2.0, or 3.0).

Note: The XSLT 3.0 content completion list of proposals includes specific elements and attributes for the 3.0 version.

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

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

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

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

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

The set of XPath functions depends on the XSLT version declared in the root element `xsl:stylesheet`: 1.0, 2.0 or 3.0.

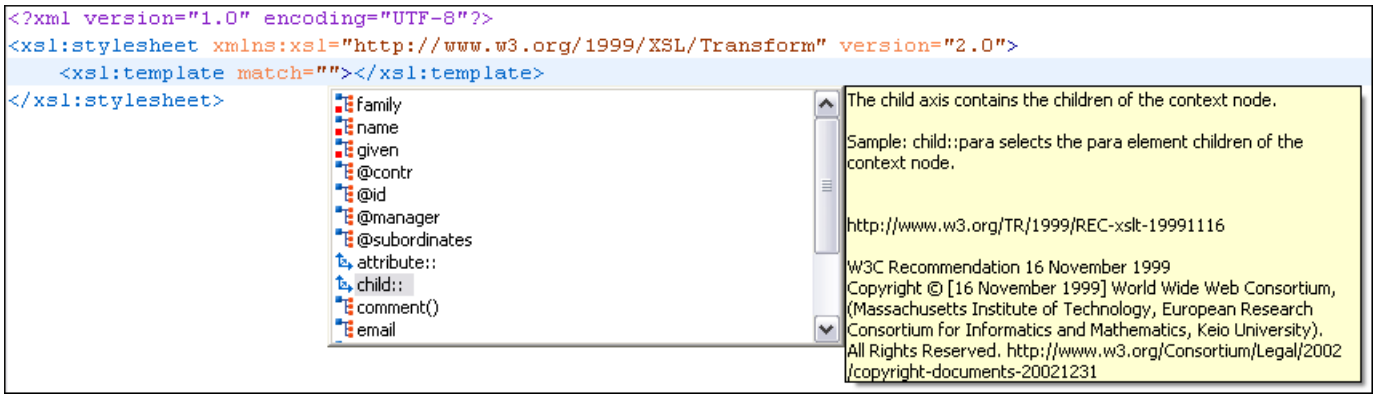


Figure 125: Content Completion in the `match` Attribute

If the cursor is inside the `select` attribute of an `xsl:for-each`, `xsl:apply-templates`, `xsl:value-of` or `xsl:copy-of` element the content completion proposals depend on the path obtained by concatenating the XPath expressions of the parent XSLT elements `xsl:template` and `xsl:for-each` as shown in the following figure:

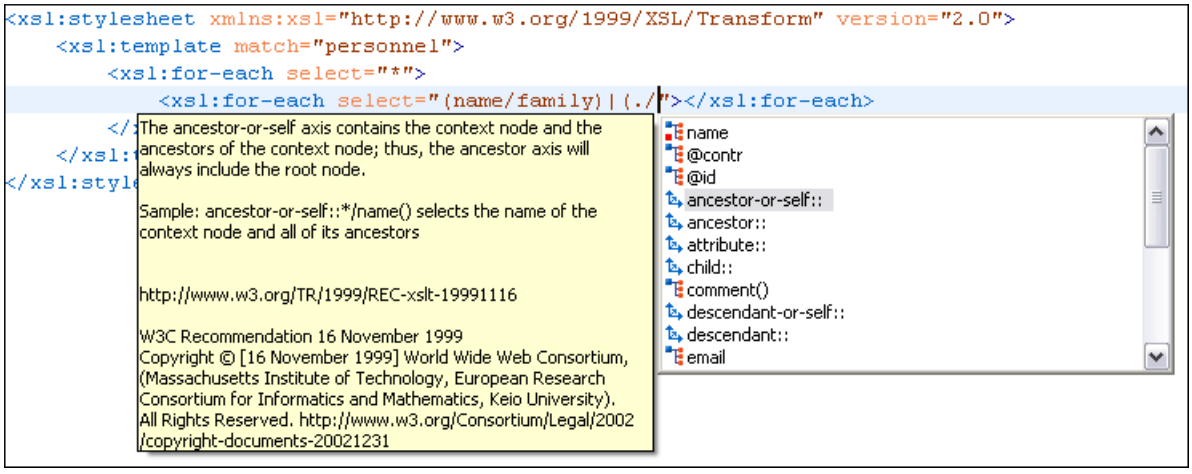


Figure 126: Content Completion in the `select` Attribute

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

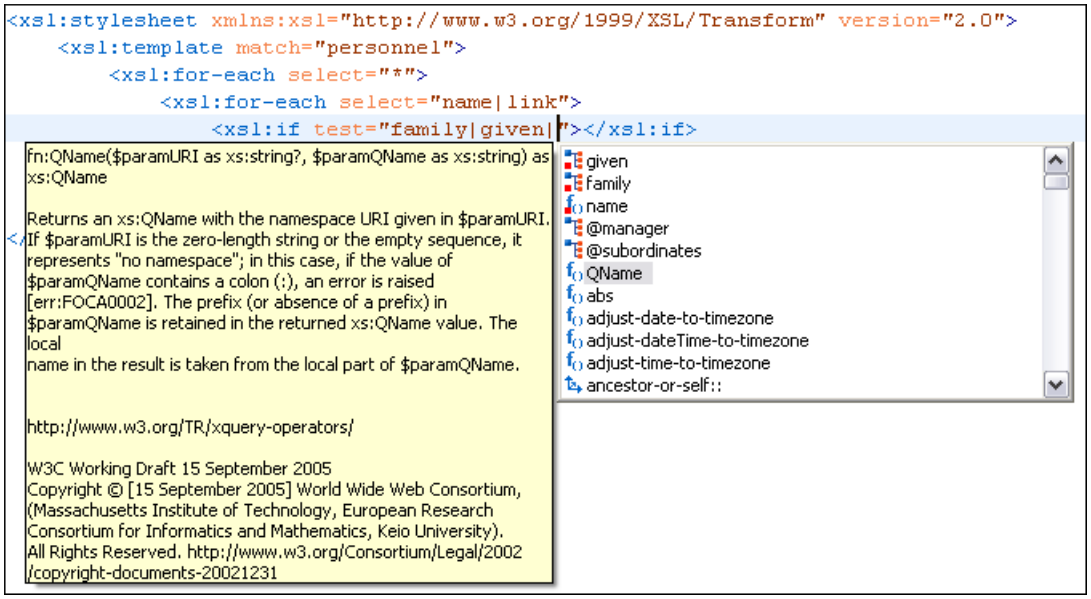


Figure 127: Content Completion in the `test` Attribute

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

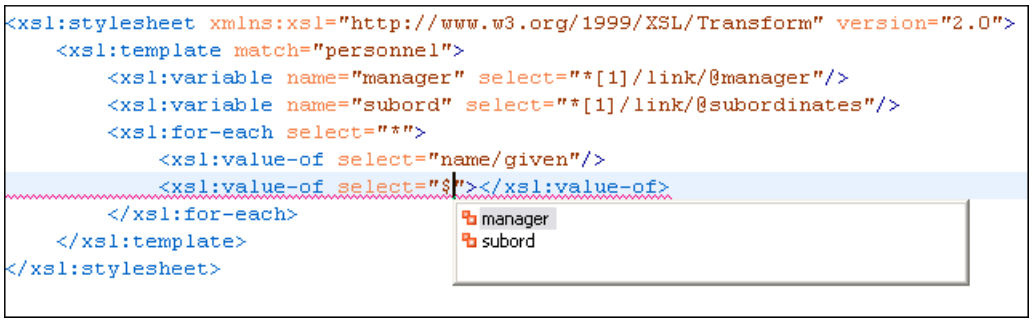


Figure 128: Content Completion in the `test` Attribute

If the { character is the first one in the value of the attribute, the same **Content Completion Assistant** is available also in attribute value templates of non-XSLT elements.

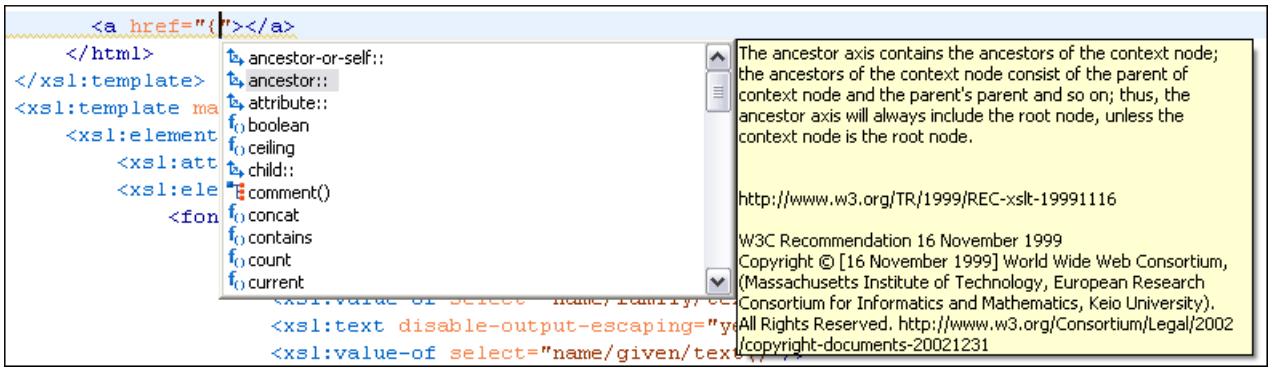


Figure 129: Content Completion in Attribute Value Templates

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

Tooltip Helper for the XPath Functions Arguments

When editing the arguments of an XPath/XSLT function, Oxygen XML Editor plugin tracks the current entered argument by displaying a tooltip containing the function signature. The currently edited argument is highlighted with a bolder font.

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

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

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

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

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

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

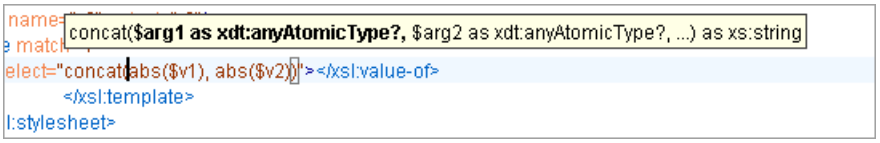


Figure 130: XPath Tooltip Helper - Identify the `concat` Function's First Argument

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

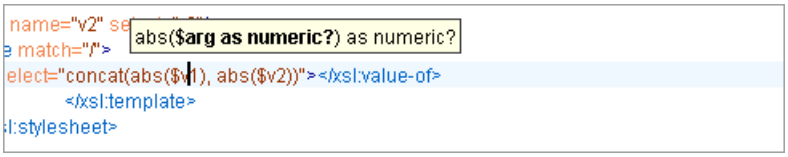


Figure 131: XPath Tooltip Helper - Identify the `abs` Function's Argument

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

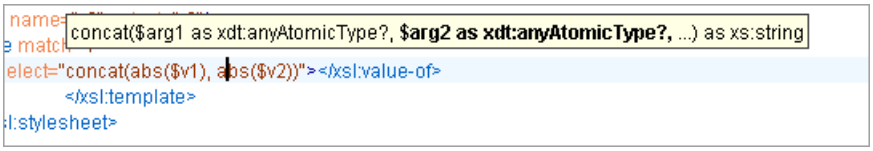


Figure 132: XPath Tooltip Helper - Identify the `concat` Function's Second Argument

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

The XSLT/XQuery Input View

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

The XSLT Input View

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

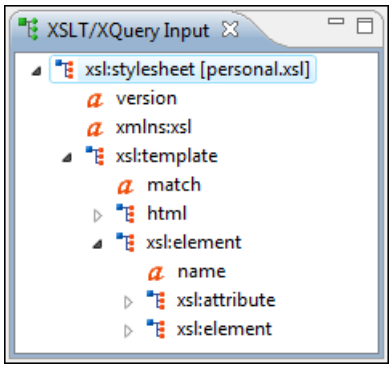


Figure 133: XSLT Input View

For example, for the following XML document:

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

and the following XSLT stylesheet:

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

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

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

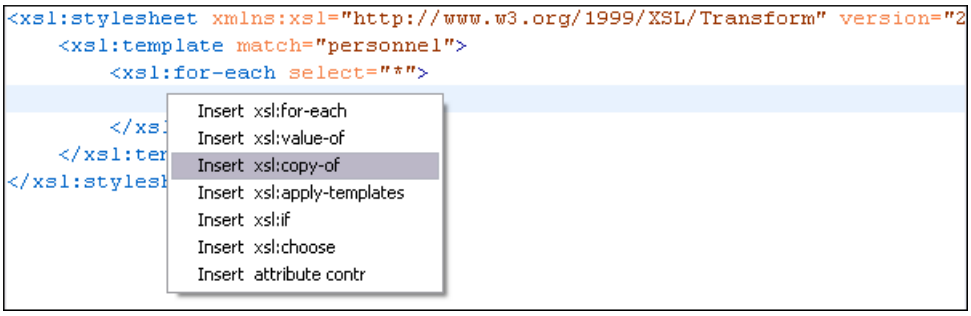


Figure 134: XSLT Input Drag and Drop Pop-up Menu

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

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

Figure 135: XSLT Input Drag and Drop Result

The XSLT Outline View

The XSLT **Outline** view displays the list of all the components (templates, attribute-sets, character-maps, variables, functions, keys, outputs) from both the edited stylesheet and its imports or includes. For XSL and XSD resources, the **Outline** view collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to *Defining Master Files at Project Level*. To enable the **Outline** view, go to **Window** > **Show View** > **Outline**.

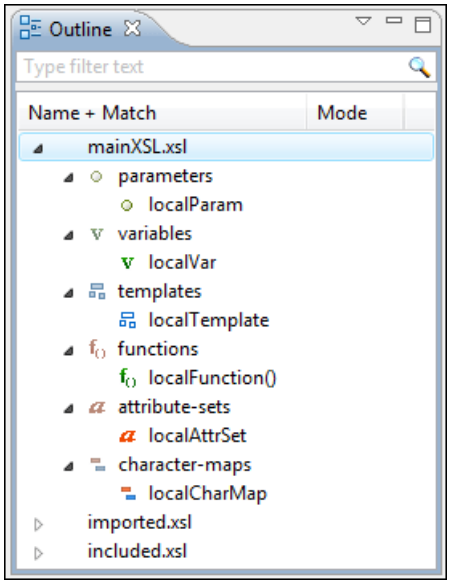


Figure 136: The XSLT Outline View

The following actions are available in the **View Menu** on the **Outline** view action bar:

Filter returns exact matches

The text filter of the **Outline** view returns only exact matches;

Selection update on cursor move

Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the cursor moves or the changes in the XSLT editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

When the **Show components** option is selected, the following actions are available:

Show XML structure

Displays the XML document structure in a tree-like structure.

Show all components

Displays all components that were collected starting from the main file. This option is set by default.

Show only local components

Displays the components defined in the current file only.

Group by location/type

The stylesheet components can be grouped by location and type.

When the **Show XML structure** option is selected, the following actions are available:

Show components

Switches the **Outline** view to the components display mode.

Flat presentation mode of the filtered results

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

Show comments and processing instructions

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

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

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

Configure displayed attributes

Displays the *XML Structured Outline preferences page*.

The following contextual menu actions are also available when the **Show components** option is selected in the **View menu**:

Edit Attributes

Opens a small in-place editor that allow you to edit the attributes of the selected node.

Cut

Cuts the currently selected node.

Copy

Copies the currently selected node.

Delete

Deletes the currently selected node.

Search References Ctrl Shift R (Meta Shift R on OS X)

Searches all references of the item found at current cursor position in the defined scope, if any. See *Finding XSLT References and Declarations* for more details.

Search References in

Searches all references of the item found at current cursor position in the specified scope. See *Finding XSLT References and Declarations* for more details.

Component Dependencies

Allows you to see the dependencies for the current selected component. See *Component Dependencies View* for more details.

Resource Hierarchy

Displays the hierarchy for the currently selected resource.

Resource Dependencies

Displays the dependencies of the currently selected resource.

Rename Component in

Renames the selected component. See *XSLT Refactoring Actions* for more details.

The following contextual menu actions are available in the **Outline** view when the **Show XML structure** option is selected in the **View menu**:

Append Child

Displays a list of elements that you can insert as children of the current element.

Insert Before

Displays a list of elements that you can insert as siblings of the current element, before the current element.

Insert After

Displays a list of elements that you can insert as siblings of the current element, after the current element.

Edit Attributes

Opens a small in-place editor that allow you to edit the attributes of the selected node.

Toggle Comment

Comments/uncomments the currently selected element.

Search references

Searches for the references of the currently selected component.

Search references in

Searches for the references of the currently selected component in the context of a scope that you define.

Component dependencies

Displays the dependencies of the currently selected component.

Rename Component in

Renames the currently selected component in the context of a scope that you define.

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Delete

Deletes the currently selected component.

Expand All

Expands the structure of a component in the **Outline** view.

Collapse All

Collapses the structure of all the component in the **Outline** view.

The stylesheet components information is presented on two columns: the first column presents the `name` and `match` attributes, the second column the `mode` attribute. If you know the component name, match or mode, you can search it in the **Outline** view by typing one of these pieces of information in the filter text field from the top of the view or directly on the tree structure. When you type de component name, match or mode in the text field, you can switch to the tree structure using:

- Keyboard arrow keys
- **Enter** key
- **Tab** key
- **Shift-Tab** key combination

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

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

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

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

The content of the **Outline** view and the editing area are synchronized. When you select a component in the **Outline** view, its definition is highlighted in the editing area.

Oxygen XML Editor plugin allows you to sort the components of the tree in the **Outline** view.

Note: Sorting groups in the **Outline** view is not supported.

Oxygen XML Editor plugin has a predefined order of the groups in the **Outline** view:

- For location, the names of the files are sorted alphabetically. The main file is the one you are editing and it is located at the top of the list.
- For type, the order is: parameters, variables, templates, functions, set attributes, character-map.

Note: When no grouping is available and the table is not sorted, Oxygen XML Editor plugin sorts the components depending on their order in the document. Oxygen XML Editor plugin also takes into account the name of the file that the components are part of.

XSLT Stylesheet Documentation Support

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

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

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

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

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

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

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

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

Generating Documentation for an XSLT Stylesheet

You can use Oxygen XML Editor plugin to generate detailed documentation in HTML format for the elements (top-level elements whose names are in the XSLT namespace) of an XSLT stylesheet. You are able to select what XSLT elements to include in the generated documentation and also the level of details to present for each of them. The elements are hyperlinked. To generate documentation in a *custom output format*, you can edit the XSLT stylesheet used to generate the documentation, or create your own stylesheet.

To open the **XSLT Stylesheet Documentation** dialog box, select **XSLT Stylesheet Documentation** from the **XML Tools** > **Generate Documentation** menu or from the **Generate Stylesheet Documentation** action from the contextual menu of the **Navigator** view.

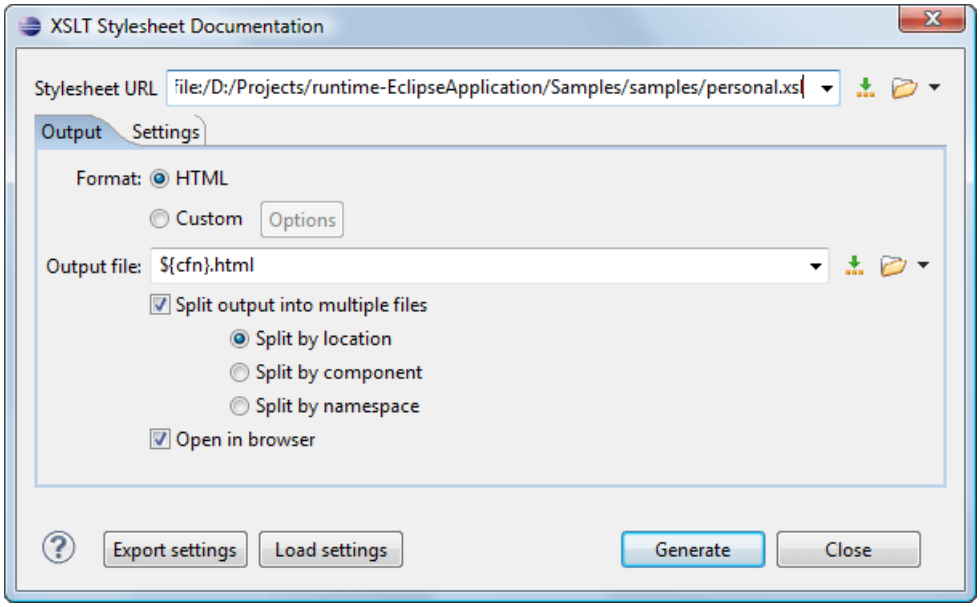


Figure 137: The XSLT Stylesheet Documentation Dialog Box

The **XSL URL** field of the dialog box must contain the full path to the XSL Stylesheet file you want to generate documentation for. The stylesheet may be a local or a remote file. You can specify the path to the stylesheet by entering it in the text field, or by using the **Insert Editor Variables** button or the options in the **Browse** drop-down menu.

The Output Tab

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

- **Format** - Allows you to choose between the following formats:
  - **HTML** - The documentation is generated in *HTML output format*.
  - **Custom** - The documentation is generated in a *custom output format*, allowing you to control the output. Click the **Options** button to open a **Custom format options** dialog box where you can specify a custom stylesheet for creating the output. There is also an option to **Copy additional resources to the output folder** and you can select the path to the additional **Resources** that you want to copy. You can also choose to keep the intermediate XML files created during the documentation process by deselecting the **Delete intermediate XML file** option.
- **Output file** - You can specify the path of the output file by entering it in the text field, or by using the **Insert Editor Variables** button or the options in the **Browse** drop-down menu.
- **Split output into multiple files** - Instructs the application to split the output into multiple files. For large XSLT stylesheets being documented, choosing a different split criterion may generate smaller output files providing a faster documentation browsing. You can choose to split them by namespace, location, or component name.
- **Open in Browser/System Application** - Opens the result in the system application associated with the output file type.

Note: To set the browser or system application that will be used, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**. This will take precedence over the default system application settings.

The Settings Tab

When you generate documentation for an XSLT stylesheet you can choose what XSLT elements to include in the output (templates, functions, global parameters, global variables, attribute sets, character maps, keys, decimal formats, output formats, XSLT elements from referenced stylesheets) and the details to include in the documentation.

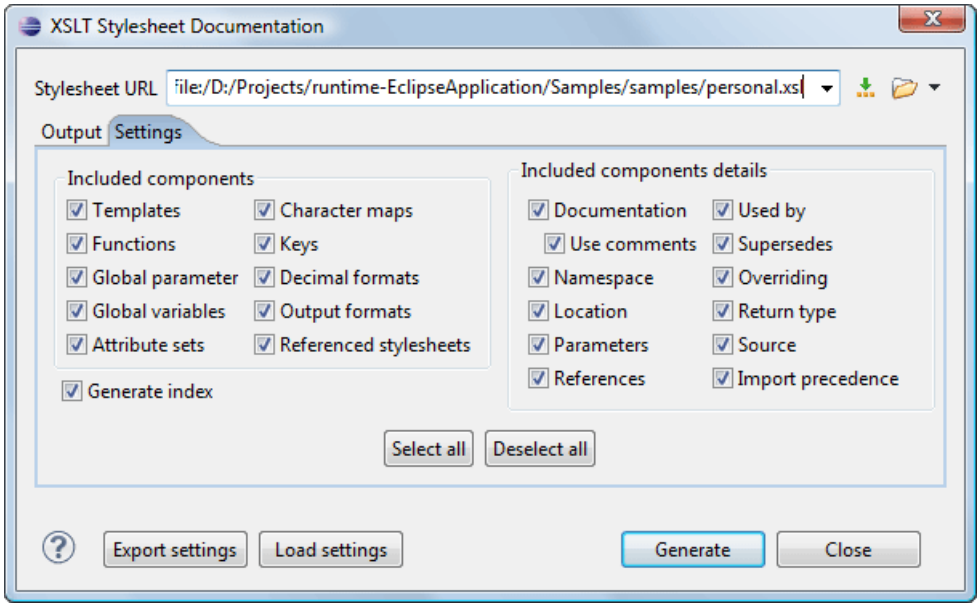


Figure 138: The Settings Tab of the XSLT Stylesheet Documentation Dialog Box

The **Settings** tab allows you to choose whether or not to include the following components: **Templates**, **Functions**, **Global parameters**, **Global variables**, **Attribute sets**, **Character maps**, **Keys**, **Decimal formats**, **Output formats**, **Referenced stylesheets**.

You can choose whether or not to include the following other details:

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

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

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

- **Used by** - Shows the list of all the XSLT elements that reference the current named element.
- **Supersedes** - Shows the list of all the XSLT elements that are superseded the current element.
- **Overriding** - Shows the list of all the XSLT elements that override the current element.
- **Return type** - Shows the return type of the function.
- **Source** - Shows the text stylesheet source for each XSLT element.
- **Import precedence** - Shows the computed import precedence as declared in the XSL transformation specifications.
- **Generate index** - Creates an index with all the XSLT elements included in the documentation.

Export settings - Save the current settings in a settings file for further use (for example, with the exported settings file you can generate the same *documentation from the command-line interface*.)

Load settings - Reloads the settings from the exported file.

Generate - Use this button to generate the XSLT documentation.

Generate XSLT Documentation in HTML Format

The XSLT documentation generated in HTML format is presented in a visual diagram style with various sections, hyperlinks, and options.

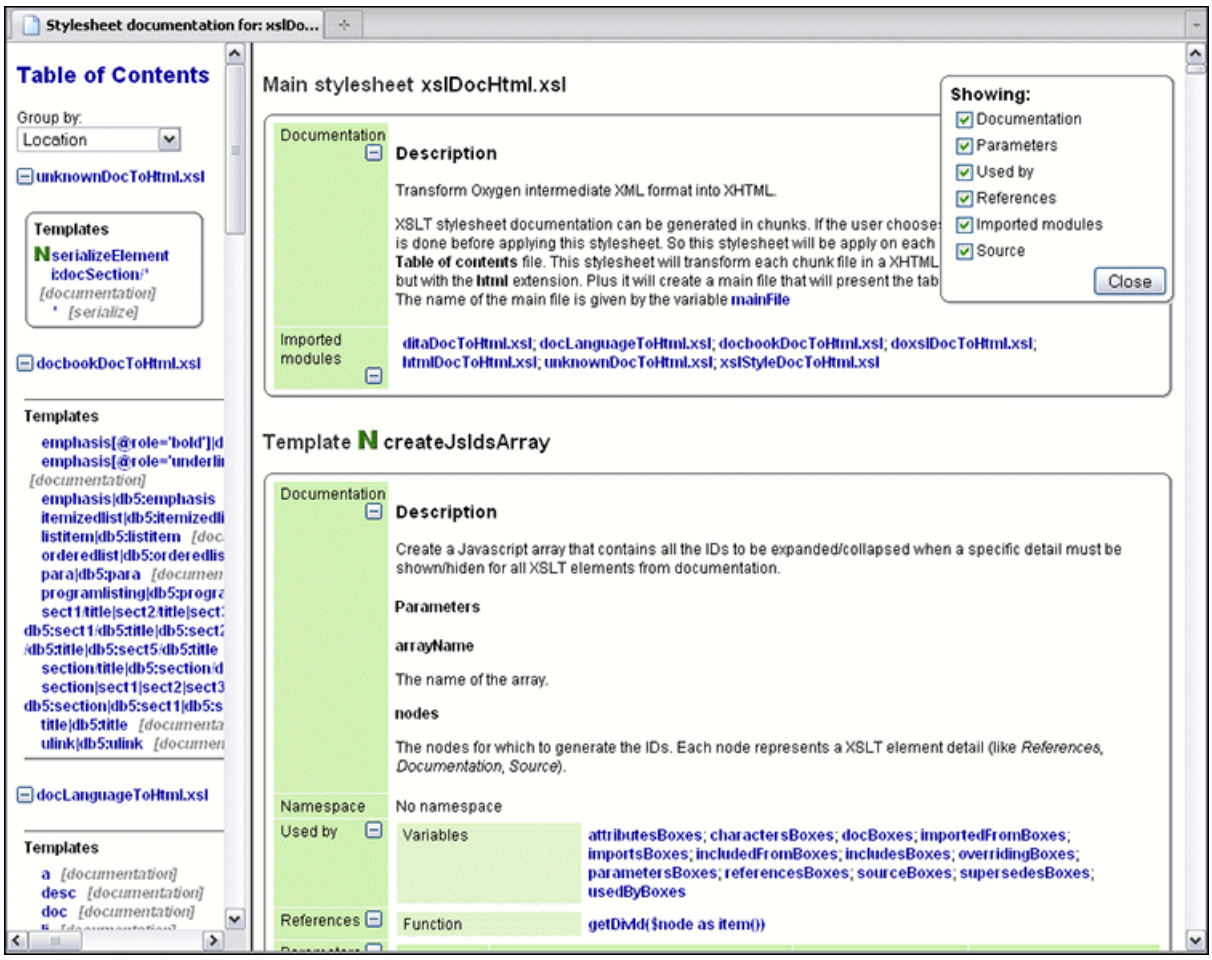


Figure 139: XSLT Stylesheet Documentation Example

The generated documentation includes the following:

- Table of Contents - You can group the contents by namespace, location, or component type. The XSLT elements from each group are sorted alphabetically (named templates are presented first and the `match` ones second).
- Information about main, imported, and included stylesheets. This information consists of:
  - XSLT modules included or imported by the current stylesheet.

- The XSLT stylesheets where the current stylesheet is imported or included.
- The stylesheet location.

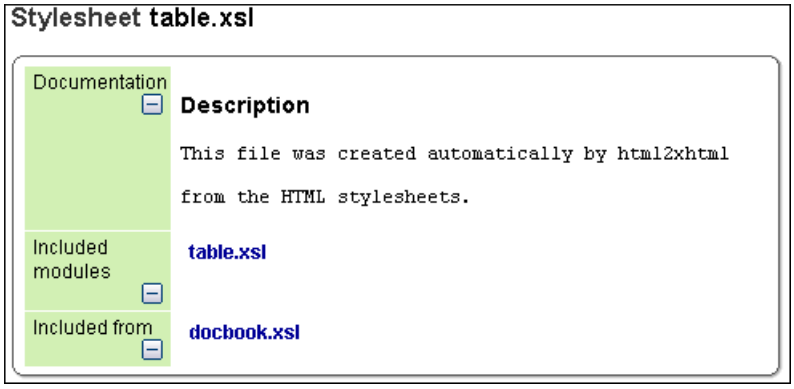


Figure 140: Information About an XSLT Stylesheet

If you choose to split the output into multiple files, the table of contents is displayed in the left frame. The contents are grouped using the same criteria as the split.

After the documentation is generated, you can collapse or expand details for some stylesheet XSLT elements by using the **Showing** options or the **Collapse** or **Expand** buttons.

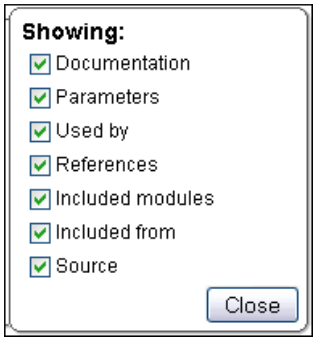


Figure 141: The Showing Options

For each element included in the documentation, the section presents the element type followed by the element name (value of the `name` or `match` attribute for match templates).

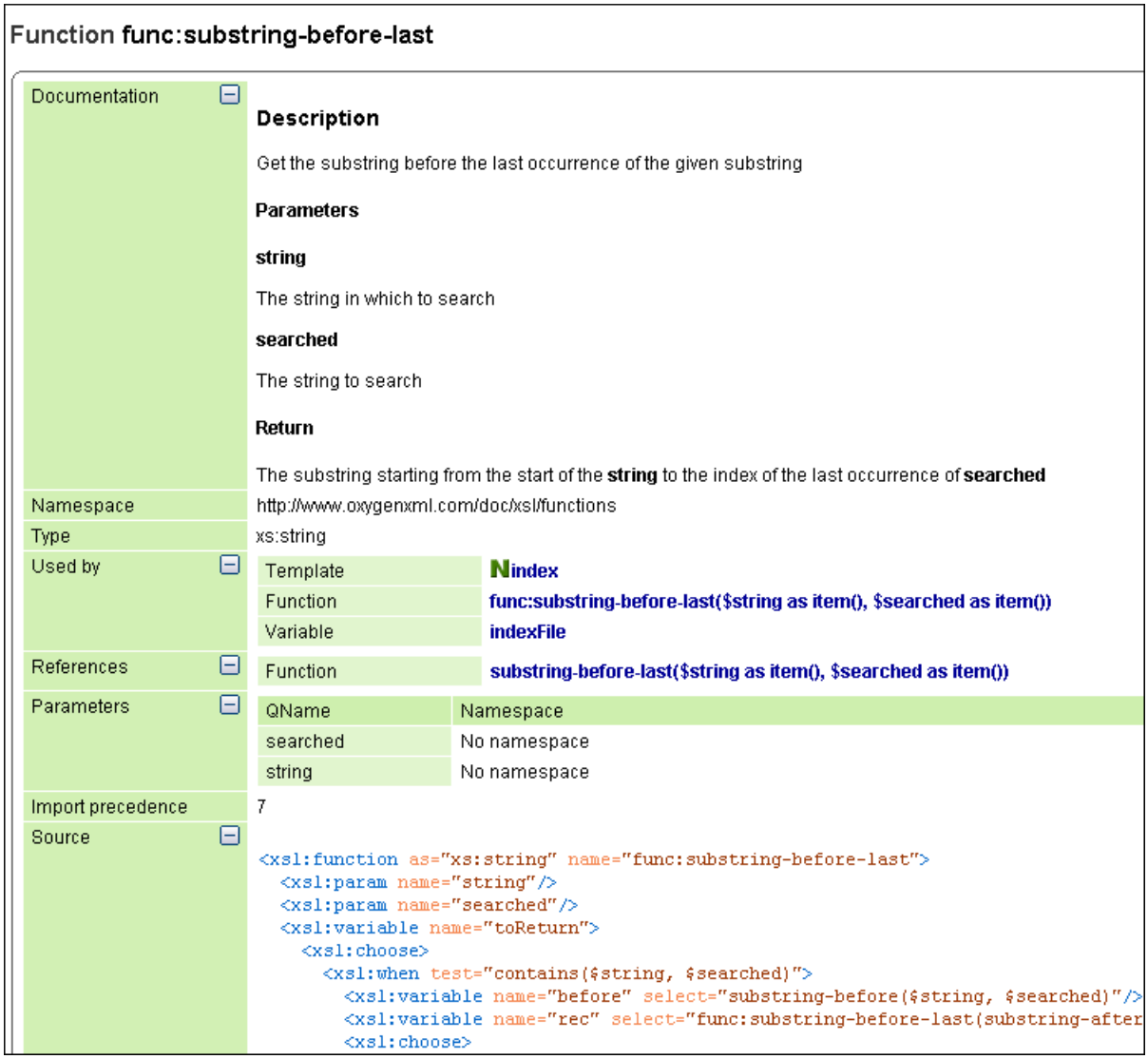


Figure 142: Documentation for an XSLT Element

Generate XSLT Documentation in a Custom Format

XSLT stylesheet documentation can be also generated in a custom format. You can choose the format from the *XSLT Stylesheet Documentation dialog box*. Specify your own stylesheet to transform the intermediary XML generated in the documentation process. You must write your stylesheet based on the schema `xslDocSchema.xsd` from `[OXYGEN_DIR]/frameworks/stylesheet_documentation`. You can create a custom format starting from one of the stylesheets used in the predefined HTML, PDF, and DocBook formats. These stylesheets are available in `[OXYGEN_DIR]/frameworks/stylesheet_documentation/xsl`.

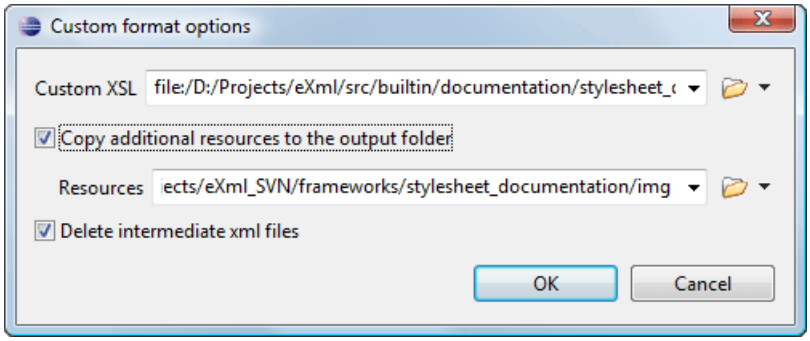


Figure 143: The Custom Format Options Dialog Box

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

Generating XSLT Documentation From the Command-Line Interface

You can export the settings of the **XSLT Stylesheet Documentation** dialog box to an XML file by pressing the **Export settings** button. With the exported settings file, you can generate the same documentation from the command line by running the script `stylesheetDocumentation.bat` (on Windows) / `stylesheetDocumentation.sh` (on OS X / Unix / Linux) located in the Oxygen XML Editor plugin installation folder. The script can be integrated in an external batch process launched from the command-line interface.

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

Example of an XML Configuration File

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

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

Finding XSLT References and Declarations

The following search actions related with XSLT references and declarations are available from the **Search** submenu of the contextual menu:

- **Search References** (Also available from the **XSL** menu) - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined but the currently edited resource is not part of the range of determined resources, a warning dialog box is displayed that allows you to define another search scope.
- **Search References in** - Searches all references of the item found at current cursor position in the file or files that you specify when a scope is defined.
- **Search Declarations** (Also available from the **XSL** menu) - Searches all declarations of the item found at current cursor position in the defined scope, if any. If a scope is defined but the current edited resource is not part of the

range of resources determined by this scope, a warning dialog box is displayed that allows you to define another search scope.

- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when a scope is defined.
- **Search Occurrences in File** - Searches all occurrences of the item at the cursor position in the currently edited file.

The following action is available from the **XSL** menu:

- **Show Definition** - Moves the cursor to the location of the definition of the current item.

  Note: You can also use the **Ctrl Single-Click (Command Single-Click on OS X)** shortcut on a reference to display its definition.

Highlight Component Occurrences

When a component (for example variable or named template) is found at current cursor position, Oxygen XML Editor plugin performs a search over the entire document to find the component declaration and all its references. When found, they are highlighted both in the document and in the stripe bar, at the right side of the document.

Note: Oxygen XML Editor plugin also supports occurrences highlight for template modes.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.

This feature is enabled by default. To configure it, *open the **Preferences** dialog box* and go to **Editor** > **Mark Occurrences**. A search can also be triggered with the **Search** > **Search Occurrences in File (Ctrl Shift U (Meta Shift U on OS X))** contextual menu action. Matches are displayed in separate tabs of the **Results** view.

XSLT Refactoring Actions

Oxygen XML Editor plugin offers a set of actions that allow changing the structure of an XSLT stylesheet without changing the results of running it in an XSLT transformation. Depending on the selected text, the following refactoring actions are available from **Refactoring** submenu from the contextual menu of the current editor:

- **Extract template** - Extracts the selected XSLT instructions sequence into a new template. Opens a dialog box that allows you to specify the name of the new template to be created. The possible changes to perform on the document can be previewed before altering the document. After pressing OK, the template is created and the selection is replaced with a `<xsl:call-template>` instruction referencing the newly created template.

  Note: This action is available only when the selection contains well-formed elements.

  Note: The newly created template is indented and its name is highlighted in the `<xsl:call-template>` element.

- **Move to another stylesheet** - Allows you to move one or more XSLT global components (templates, functions, or parameters) to another stylesheet. Active only when these entire components are selected. Follow these steps:

  1. Choose whether you want to move the selected components to a new stylesheet or an existing one.
  2. Select the destination stylesheet. Press the **Choose** button to select the destination stylesheet file. Oxygen XML Editor plugin will automatically check if the destination stylesheet is already contained by the hierarchy of the current stylesheet. If it is not contained, choose whether or not the destination stylesheet will be referenced (imported or included) from the current stylesheet. The following options are available:
     - **Include** - The current stylesheet will use an `xsl:include` instruction to reference the destination stylesheet.
     - **Import** - The current stylesheet will use an `xsl:import` instruction to reference the destination stylesheet.
     - **None** - There will be created no relation between the current and destination stylesheets.

3. Press the **Move** button to move the components to the destination stylesheet. The moved components are highlighted in the destination stylesheet.

- **Convert attributes to xsl:attributes** - Converts the attributes from the selected element and represents each of them with an `<xsl:attribute>` instruction. For example, from the following element:

```
<person id="Big{test}Boss"/>
```

you obtain:

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

- **Convert xsl:if into xsl:choose/xsl:when** - Converts an `xsl:if` block to an `xsl:when` block surrounded by an `xsl:choose` element. For example, the following block:

```
<xsl:if test="a">
  <!-- XSLT code -->
</xsl:if>
```

is converted to:

```
<xsl:choose>
    <xsl:when test="a">
        <!-- XSLT code -->
    </xsl:when>
    <xsl:otherwise>
        |
    </xsl:otherwise>
</xsl:choose>
```

where the | character is the current cursor position.

- **Extract local variable** - Allows you to create a new local variable by extracting the selected XPath expression. After creating the new local variable before the current element, Oxygen XML Editor plugin allows you to edit in-place the variable's name.

  Note: The action is active on a selection made inside an attribute that contains an XPath expression.

- **Extract global variable** - Allows you to create a new global variable by extracting the selected XPath expression. After creating the new global variable, Oxygen XML Editor plugin allows you to edit in-place the variable's name.

  Note: The action is active on a selection made inside an attribute that contains an XPath expression.

  Note: Oxygen XML Editor plugin checks if the selected expression depends on local variables or parameters that are not available in the global context where the new variable is created.

- **Extract template parameter** - Allows you to create a new template parameter by extracting the selected XPath expression. After creating the new parameter, Oxygen XML Editor plugin allows you to edit in-place its name.

  Note: The action is active on a selection made inside an attribute that contains an XPath expression.

- **Extract global parameter** - Allows you to create a new global parameter by extracting the selected XPath expression. After creating the new parameter, Oxygen XML Editor plugin allows you to edit in-place its name.

  Note: The action is active on a selection made inside an attribute that contains an XPath expression.

  Note: Oxygen XML Editor plugin checks if the selected expression depends on local variables or parameters that are not available in the global context where the new parameter is created.

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the cursor position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
- **Rename Component in** - Opens the **Rename *component_type*** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

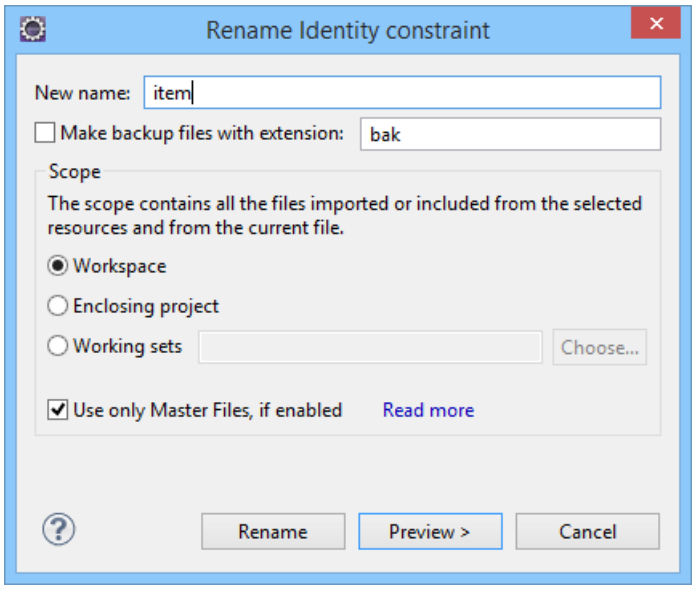


Figure 144: Rename Identity Constraint Dialog Box

Note: These refactoring actions are also proposed by the *Quick Assist support*.

To watch our video demonstration about XSLT refactoring, go to *http://oxygenxml.com/demo/XSL_Refactoring.html*.

XSLT Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a stylesheet. To open this view, go to **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** > **Resource Hierarchy/Dependencies**.

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

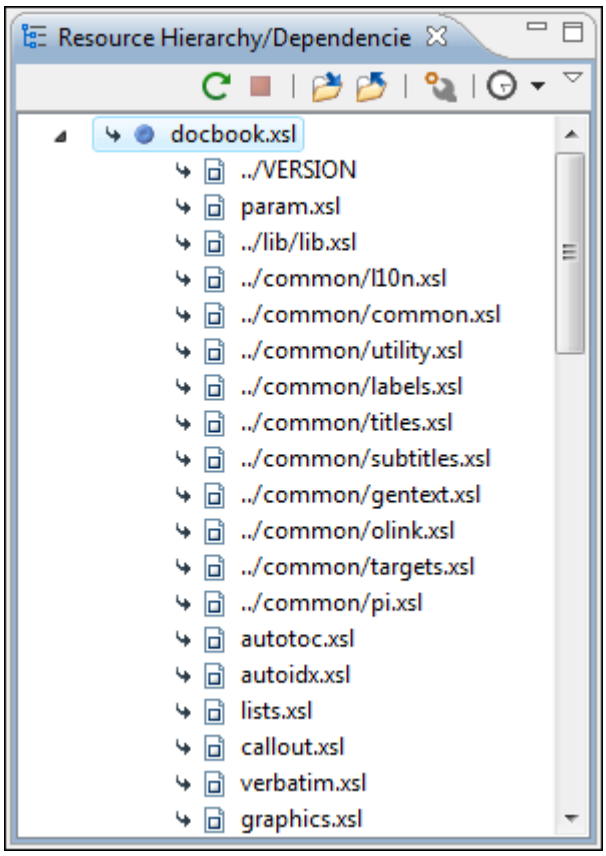


Figure 145: Resource Hierarchy/Dependencies View - Hierarchy for `docbook.xsl`

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

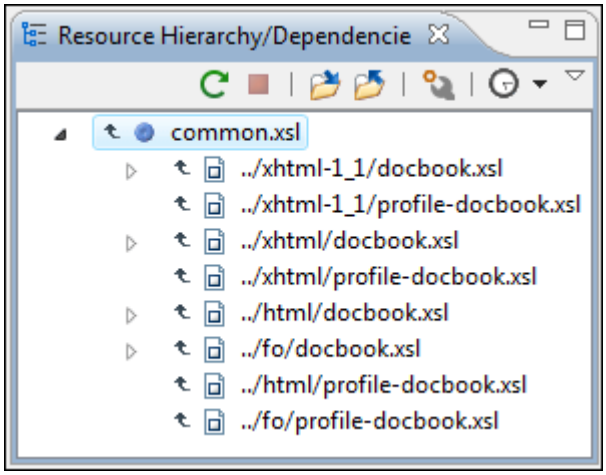


Figure 146: Resource Hierarchy/Dependencies View - Dependencies for common.xsl

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

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

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

History

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

The contextual menu contains the following actions:

Open

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

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the **Master Files** directory*.

Expand All

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

Collapse All

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

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

Moving/Renaming XSLT Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

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

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

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

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.

- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

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

XSLT Component Dependencies View

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

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

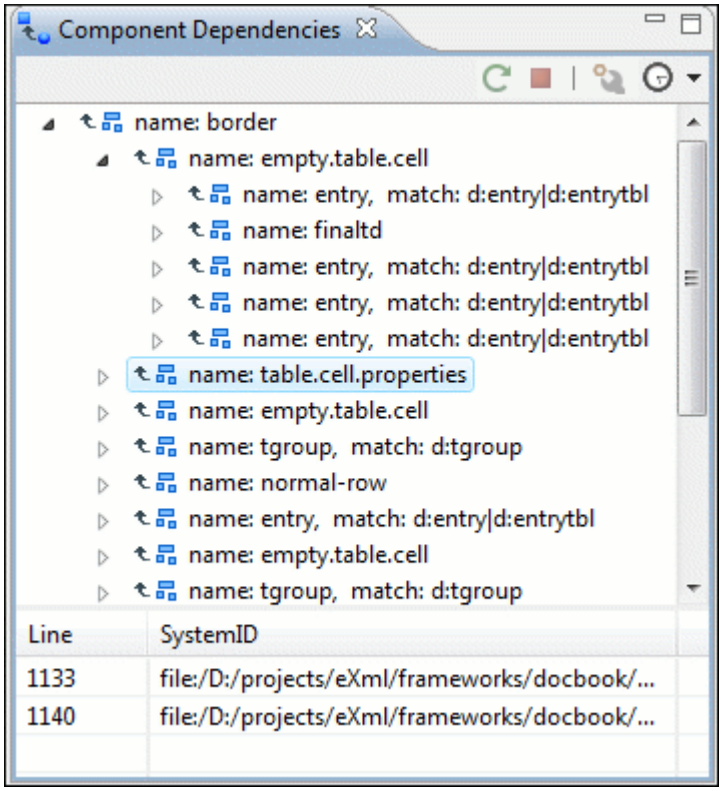


Figure 147: Component Dependencies View - Hierarchy for table.xsl

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

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependencies computing.

Configure

Allows you to configure a search scope to compute the dependencies structure. You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.

History

Allows you to repeat a previous dependencies computation.

The following actions are available on the contextual menu:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Shows the definition of the current selected component in the dependencies tree.

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

XSLT Quick Assist Support

The **Quick Assist** support helps you to rapidly access search and refactoring actions. If one or more actions are available in the current context, they are accessible via a yellow bulb help marker placed on the cursor line, in the editor line number stripe. Also, you can invoke the **Quick Assist** menu if you press **Ctrl + 1** keys (**Meta 1** on Mac OS X) on your keyboard.

Two categories of actions are available in the **Quick Assist** menu:

- Actions available on a selection made inside an attribute that contains an XPath expression:
  - **Extract template** - Extracts the selected XSLT instructions sequence into a new template.
  - **Move to another stylesheet** - Allows you to move one or more XSLT global components (templates, functions, or parameters) to another stylesheet.
  - **Extract local variable** - Allows you to create a new local variable by extracting the selected XPath expression.
  - **Extract global variable** - Allows you to create a new global variable by extracting the selected XPath expression.
  - **Extract template parameter** - Allows you to create a new template parameter by extracting the selected XPath expression.
  - **Extract global parameter** - Allows you to create a new global parameter by extracting the selected XPath expression.

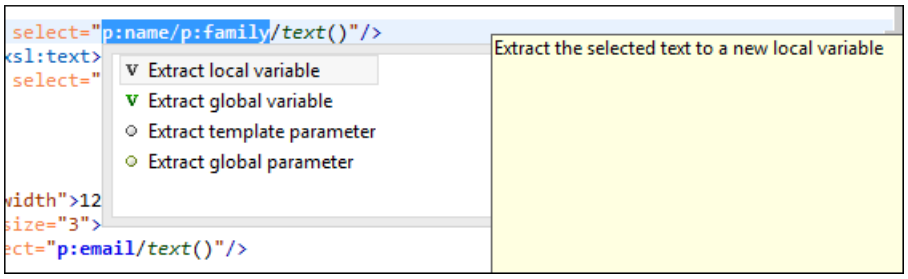


Figure 148: XSLT Quick Assist Support - Refactoring Actions

- actions available when the cursor is positioned over the name of a component:

  Rename Component in

  Renames the component and all its dependencies.

  Search Declarations

  Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

  Search References

  Searches all references of the component in a predefined scope.

  Component Dependencies

  Searches the component dependencies in a predefined scope.

Change Scope

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

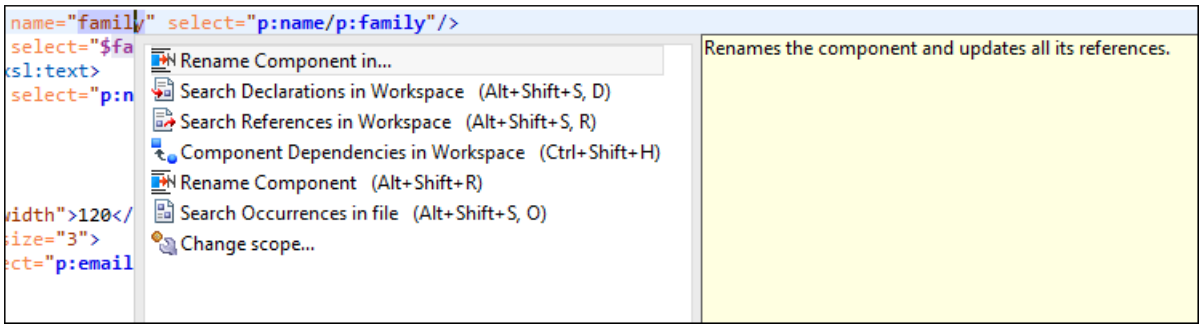


Figure 149: XSLT Quick Assist Support - Component Actions

XSLT Quick Fix Support

The Oxygen XML Editor plugin Quick Fix support helps you resolve various errors that appear in a stylesheet by proposing quick fixes to problems such as missing templates, misspelled template names, missing functions, or references to an undeclared variable or parameter.

To activate this feature, hover over or place the cursor in the highlighted area of text where a validation error or warning occurs. If a Quick Fix is available for that particular error or warning, you can access the Quick Fix proposals with any of the following methods:

- When hovering over the error or warning, the proposals are presented in a tooltip popup window.
- If you place the cursor in the highlighted area where a validation error or warning occurs, a quick fix icon ( ) is displayed in the stripe on the left side of the editor. If you click this icon, Oxygen XML Editor plugin displays the list of available fixes.
- With the cursor placed in the highlighted area of the error or warning, you can also invoke the quick fix menu by pressing **Ctrl 1 (Meta 1 on OS X)** on your keyboard.

Note: The quick fixes are available only when validating an XSLT file with Saxon HE/PE/EE.

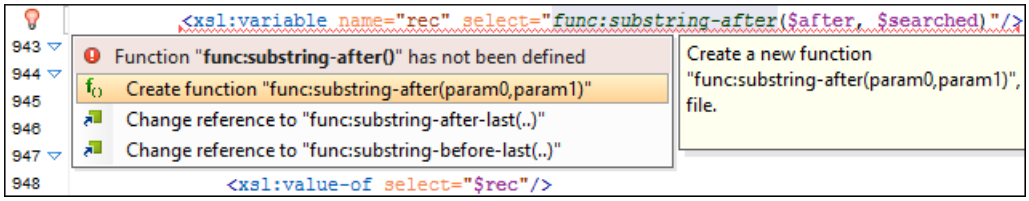


Figure 150: Example of an Undefined XSLT Functions Quick Fix

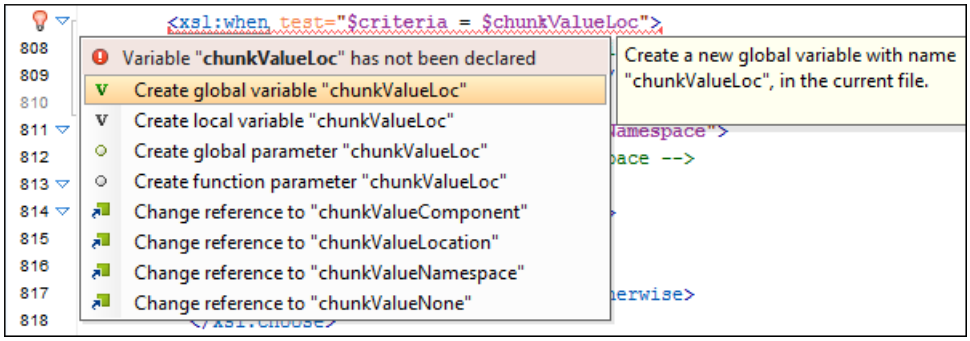


Figure 151: Example of an Undeclared XSLT Variables/Parameters Quick Fix

Oxygen XML Editor plugin provides XSLT quick fixes for the following types of instances:

- **Template does not exist**, when the template name referenced in a `call-template` element does not exist. The following fixes are available:
  - **Create template "templateName"** - creates a template and generates its corresponding parameters. The template name and parameter names and types are collected from the `call-template` element.
  - **Change reference to "newTemplateName"** - changes the name of the missing template referenced in the `call-template` element. The proposed new names are the existing templates with names similar with the missing one.
- **Variable/Parameter not declared**, when a parameter or variable reference cannot be found. The following fixes are available:
  - **Create global variable "varName"** - creates a global variable with the specified name in the current stylesheet. The new variable is added at the beginning of the stylesheet after the last global variable or parameter declaration.
  - **Create global parameter "paramName"** - creates a global parameter with the specified name in the current stylesheet. The new parameter is added at the beginning of the stylesheet after the last global parameter or variable declaration.
  - **Create local variable "varName"** - creates a local variable with the specified name before the current element.
  - **Create template parameter "paramName"** - creates a new parameter with the specified name in the current template. This fix is available if the error is located inside a template.
  - **Create function parameter "paramName"** - creates a new parameter with the specified name in the current function. This fix is available if the error is located inside a function.
  - **Change reference to "varName"** - changes the name of the referenced variable/parameter to an existing local or global variable/parameter, that has a similar name with the current one.
- **Parameter from a called template is not declared**, when a parameter referenced from a `call-template` element is not declared. The following fixes are available:
  - **Create parameter "paramName" in the template "templateName"** - creates a new parameter with the specified name in the referenced template.
  - **Change "paramName" parameter reference to "newParamName"** - changes the parameter reference from the `call-template` element to a parameter that is declared in the called template.
  - **Remove parameter "paramName" from call-template** - removes the parameter with the specified name from the `call-template` element.
- **No value supplied for required parameter**, when a required parameter from a template is not referenced in a `call-template` element. The following quick-fix is available::
  - **Add parameter "paramName" in call-template** - creates a new parameter with the specified name in call-template element.
- **Function "prefix:functionName()" has not been defined**, when a function declaration is not found. The following quick fixes are available:

- **Create function "prefix:functionName(param1, param2)"** - creates a new function with the specified signature, after the current top level element from stylesheet.
- **Change function to "newFunctionName(..)"** - changes the referenced function name to an already defined function. The proposed names are collected from functions with similar names and the same number of parameters.

- **Attribute-set "attrSetName" does not exist**, when the referenced attribute set does not exist. The following quick fixes are available:
  - **Create attribute-set "attrSetName"** - creates a new attribute set with the specified name, after the current top level element from stylesheet.
  - **Change reference to "attrSetName"** - changes the referenced attribute set to an already defined one.
- **Character-map "chacterMap" has not been defined**, when the referenced character map declaration is not found. The following quick fixes are available:
  - **Create character-map "characterMapName"** - creates a new character map with the specified name, after the current top level element from stylesheet.
  - **Change reference to "characterMapName"** - changes the referenced character map to an already defined one.

Linking Between Development and Authoring

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

XSLT Unit Test (XSpec)

XSpec is a behavior driven development (BDD) framework for XSLT and XQuery. XSpec consists of a syntax for describing the behavior of your XSLT or XQuery code, and some code that enables you to test your code against those descriptions.

To create an XSLT Unit Test, go to **File** > **New** > **XSLT Unit Test**. You can also create an XSLT Unit Test from the contextual menu of an XSL file in the **Project** view. Oxygen XML Editor plugin allows you to customize the XSpec document when you create it. In the customization dialog box, you can enter the path to an XSL document or to a master XSL document.

To run an XSLT Unit Test, open the XSPEC file in an editor and click  **Apply Transformation Scenario(s)** on the main toolbar.

Note: The transformation scenario is defined in the XSPEC *document type*.

When you create an XSpec document based on an XSL document, Oxygen XML Editor plugin uses information from the validation and transformation scenarios associated with the XSL file. From the transformation scenario Oxygen XML Editor plugin uses extensions and properties of Saxon 9.6.0.7, improving the Ant scenario associated with the XSpec document.

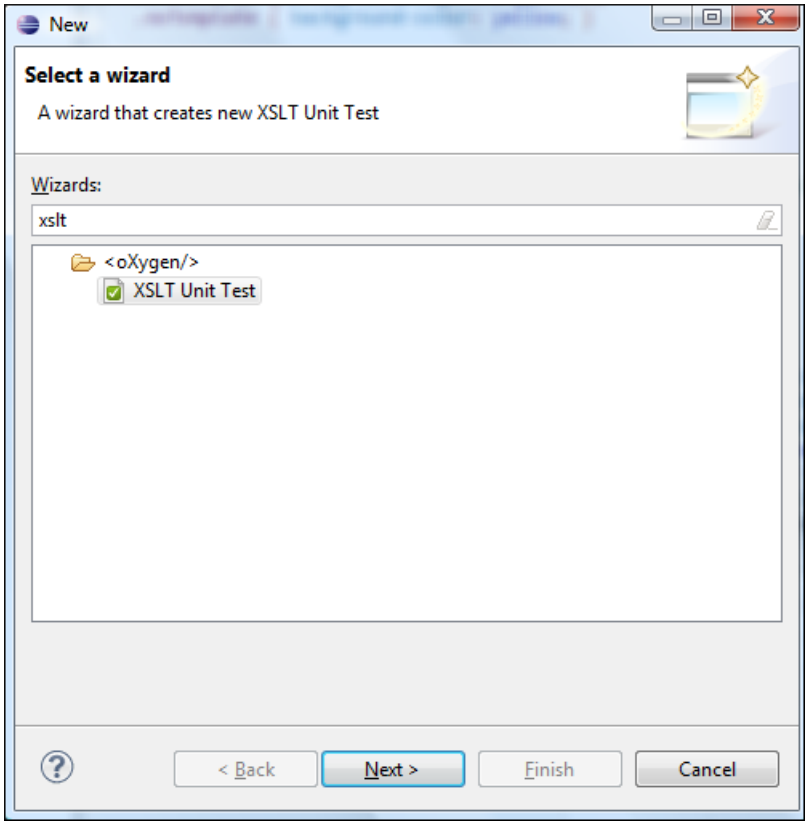


Figure 152: The New XSLT Unit Test Wizard

An XSpec file contains one, or more test scenarios. You can test a stylesheet in one of the following ways:

- Test an entire stylesheet.

  Testing is performed in a certain context. You can define a context as follows:

  - Inline context, building the test based on a string.

    ```
    <x:scenario label="when processing a para element">
        <x:context>
            <para>...</para>
        </x:context>
        ...
    </x:scenario>
    ```

  - Based on an external file, or on a part of an external file extracted with an XPath expression.

    ```
    <x:scenario label="when processing a para element">
        <x:context href="source/test.xml" select="/doc/body/p[1]" />
        ...
    </x:scenario>
    ```

- Test a function.

  ```
  <x:scenario label="when capitalising a string">
    <x:call function="eg:capital-case">
      <x:param select="'an example string'" />
      <x:param select="true()" />
    </x:call>
    ...
  </x:scenario>
  ```

- Test a template with a name.

```
<x:call template="createTable">
    <x:param name="nodes">
      <value>A</value>
      <value>B</value>
    </x:param>
    <x:param name="cols" select="2" />
</x:call>
```

You are able to reference test files between each other, which allows you to define a suite of tests. For further details about test scenarios, go to *https://github.com/expath/xspec/wiki/Writing-Scenarios*.

Editing XML Schemas

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

Two editing modes are provided for working with XML Schema:

- The ***Text*** editing mode.
- The visual ***Design*** editing mode.

Note: Oxygen XML Editor plugin offers support for both XML schema 1.0 and 1.1.

Editing XML Schema in Design Editing Mode

This section explains how to use the graphical diagram editing mode for a W3C XML Schema.

Schema Editing Actions

You can edit an XML schema using drag and drop operations or contextual menu actions.

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

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

Depending on the drop area, different actions are available:

- **move** - Context dependent, the selected component is moved to the destination.
- **reference** - Context dependent, the selected component is referenced from the parent.
- **copy** - If **(Ctrl (Meta on Mac OS))** key is pressed, a copy of the selected component is inserted to the destination.

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

-  - When moving a component.
-  - When referencing a component.
-  - When copying a component.

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

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

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

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

Contextual Menu Actions in the Design Mode

The contextual menu of the **Design** mode includes the following actions:

Show Definition (Ctrl Shift Enter)

Shows the definition for the current selected component. For references, this action is available by clicking the arrow displayed in its bottom right corner.

Open Schema (Ctrl Shift Enter)

Opens the selected schema. This action is available for `xsd:import`, `xsd:include` and `xsd:redefine` elements. If the file you try to open does not exist, a warning message is displayed and you have the possibility to create the file.

Edit Attributes (Alt + Shift + Enter)

Allows you to edit the attributes of the selected component in a small in-place editor that presents the same attributes as in the *Attributes View* and the *Facets View*. The actions that can be performed on attributes in this dialog box are the same actions presented in the two views.

Append child

Offers a list of valid components, depending on the context, and appends your selection as a child of the currently selected component. You can set a name for a named component after it has been added in the diagram.

Insert before

Offers a list of valid components, depending on the context, and inserts your selection before the selected component, as a sibling. You can set a name for a named component after it has been added in the diagram.

Insert after

Offers a list of valid components, depending on the context, and inserts your selection after the selected component, as a sibling. You can set a name for a named component after it has been added in the diagram.

New global

Inserts a global component in the schema diagram. This action does not depend on the current context. If you choose to insert an import you have to specify the URL of the imported file, the target namespace and the import ID. The same information, excluding the target namespace, is requested for an `xsd:include` or `xsd:redefine` element.

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

Edit Schema Namespaces

When performed on the schema root, it allows you to edit the schema target namespace and namespace mappings. You can also invoke the action by double-clicking the target namespace property from **Attributes** view for the schema or by double-clicking the schema component.

Edit Annotations

Allows you to edit the annotation for the selected schema component in the **Edit Annotations** dialog box. You can perform the following operations in the dialog box:

- **Edit all `appinfo/documentation` items for a specific annotation** - all `appinfo/documentation` items for a specific annotation are presented in a table and can be easily edited. Information about an annotation item includes: type (documentation/appinfo), content, source (optional, specify the source of the `documentation/appinfo` element) and `xml:lang`. The content of a `documentation/appinfo` item can be edited in the **Content** area below the table.

- **Insert/Insert before/Remove documentation/appinfo.** The **Add** button allows you to insert a new annotation item (`documentation/appinfo`). You can add a new item before the item selected in table by pressing the **Insert Before** button. Also, you can delete the selected item using the **Remove** button.
- **Move items up/down** - to do this use the **Move up** and **Move down** buttons.
- **Insert/Insert before/Remove annotation** - available for components that allow multiple annotations like schemas or redefines.
- **Specify an ID for the component annotation**. the ID is optional.

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

Note: For imported/included components which do not belong to the currently edited schema, the **Edit Annotations** dialog box presents the annotation as read-only. To edit its annotation, open the schema where the component is defined.

Extract Global Element

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

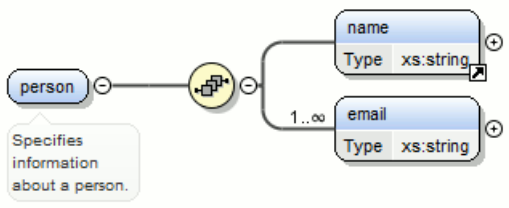
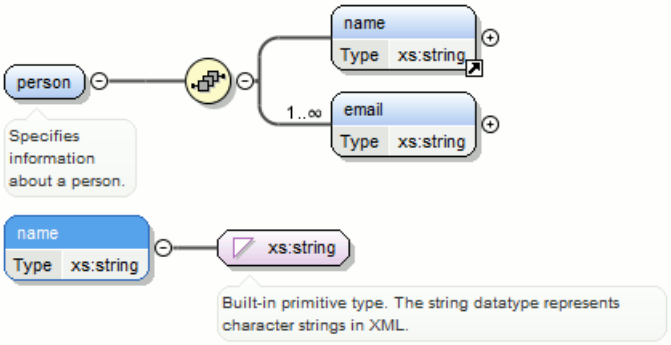


Figure 153: Extracting a Global Element

If you use the **Extract Global Element** action on a `name` element , the result is:



Extract Global Attribute

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

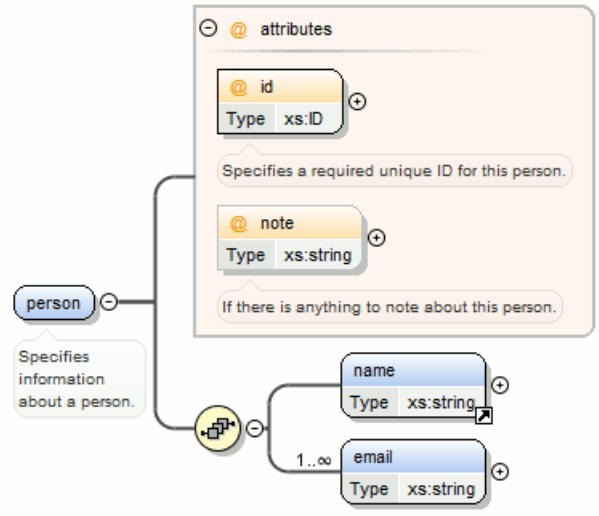
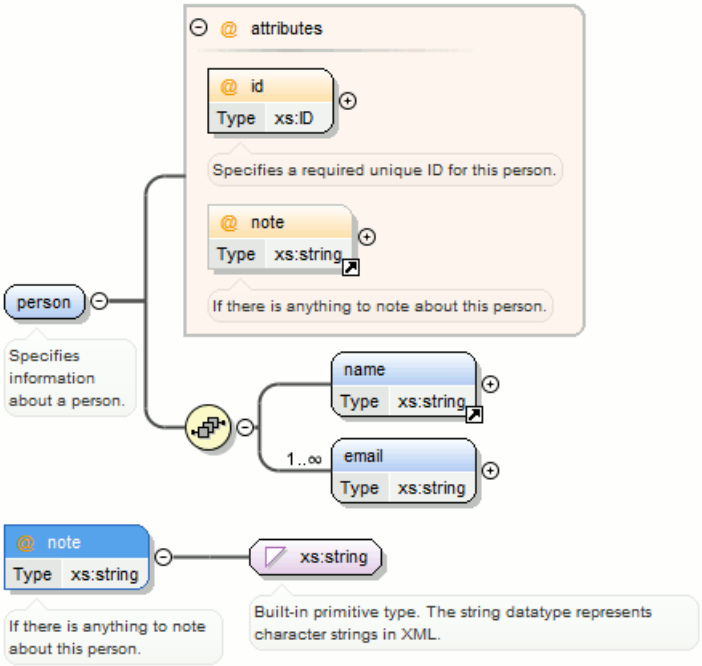


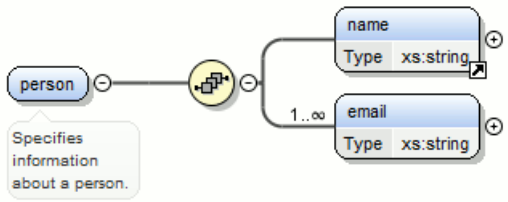
Figure 154: Extracting a Global Attribute

If you use the **Extract Global Attribute** action on a `note` attribute, the result is:



Extract Global Group

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



If you use the **Extract Global Group** action on the `sequence` element, the **Extract Global Component** dialog box is displayed and you can choose a name for the group. If you type `personGroup`, the result is:

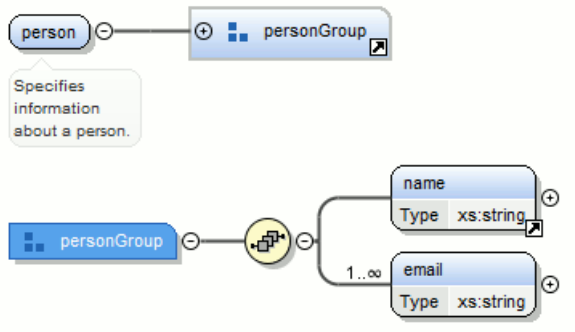


Figure 155: Extracting a Global Group

Extract Global Type

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

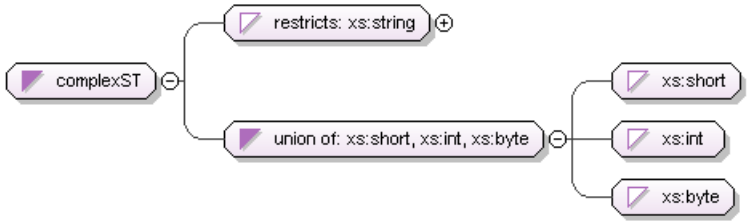


Figure 156: Extracting a Global Simple Type

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

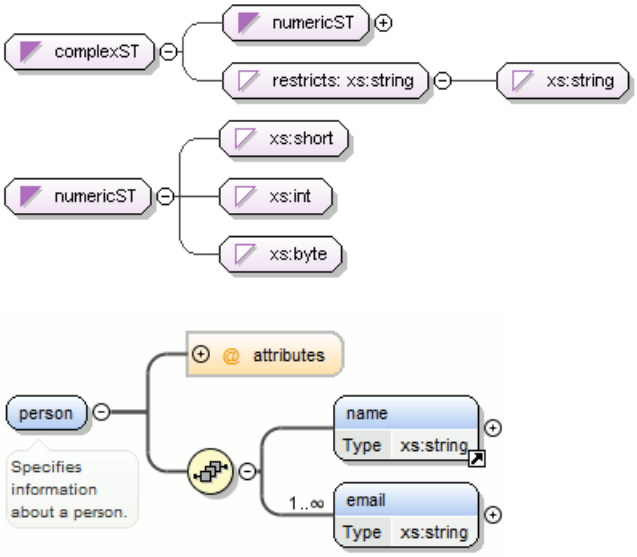
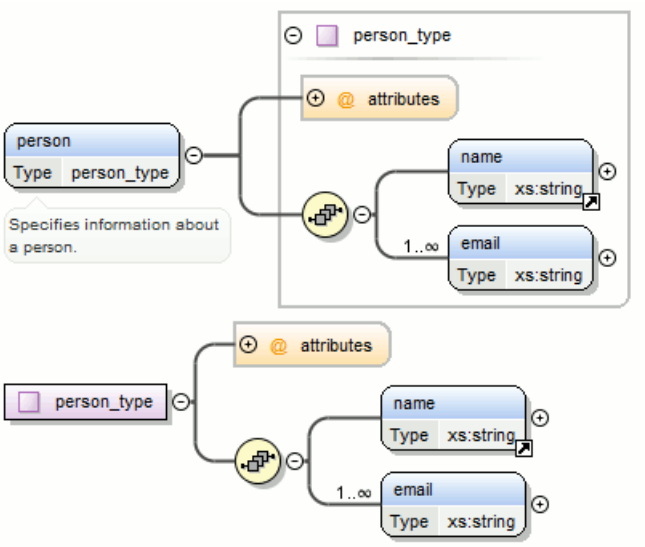


Figure 157: Extracting a Global Complex Type

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



Rename Component in

Rename the selected component.

Cut Ctrl X (Meta X on OS X)

Cut the selected component(s).

Copy Ctrl C (Meta C on OS X)

Copy the selected component(s).

Copy XPath

This action copies an XPath expression that identifies the selected element or attribute in an instance XML document of the edited schema and places it in the clipboard.

Paste Ctrl V (Meta V on OS X)

Paste the component(s) from the clipboard as children of the selected component.

Paste as Reference

Create references to the copied component(s). If not possible a warning message is displayed.

Remove (Delete)

Remove the selected component(s).

Override component

Copies the overridden component in the current XML Schema. This option is available for *xs:override* components.

Redefine component

The referenced component is added in the current XML Schema. This option is available for *xs:redefine* components.

Optional

Can be performed on element/attribute/group references, local attributes, elements, compositors, and element wildcards. The `minOccurs` property is set to 0 and the `use` property for attributes is set to `optional`.

Unbounded

Can be performed on element/attribute/group references, local attributes, elements, compositors, and element wildcards. The `maxOccurs` property is set to `unbounded` and the `use` property for attributes is set to `required`.

Search

Can be performed on local elements or attributes. This action makes a reference to a global element or attribute.

Search References

Searches all references of the item found at current cursor position in the defined scope if any.

Search References in

Searches all references of the item found at current cursor position in the specified scope.

Search Occurrences in File

Searches all occurrences of the item found at current cursor position in the current file.

Component Dependencies

Allows you to see the dependencies for the current selected component.

Resource Hierarchy

Allows you to see the hierarchy for the current selected resource.

Flatten Schema

Recursively adds the components of included Schema files to the main one. It also flattens every imported XML Schema from the hierarchy.

Resource Dependencies

Allows you to see the dependencies for the current selected resource.

Expand All

Recursively expands all sub-components of the selected component.

Collapse All

Recursively collapses all sub-components of the selected component.

Save as Image

Save the diagram as image, in JPEG, BMP, SVG or PNG format.

Generate Sample XML Files

Generate XML files using the current opened schema. The selected component is the XML document root. See more in the *Generate Sample XML Files* section.

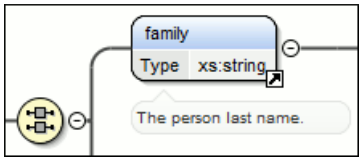
Options

Show the *Schema preferences panel*.

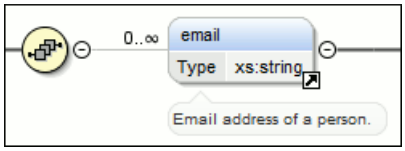
XML Schema Components

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

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

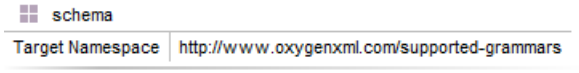


- A thin line to identify a connection with an optional component (in the following image, `email` is an optional element).



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

xs:schema



Defines the root element of a schema. A schema document contains representations for a collection of schema components, such as type definitions and element declarations, that have a common target namespace. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-schema*.

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

xs:schema properties

Property Name	Description	Possible Values
Target Namespace	The schema target namespace.	Any URI
Element Form Default	Determining whether local element declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Attribute Form Default	Determining whether local attribute declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Block Default	Default value of the `block` attribute of `xs:element` and `xs:complexType`.	#all, extension, restriction, substitution, restriction extension, restriction substitution, extension substitution, restriction extension substitution, [Empty].
Final Default	Default value of the `final` attribute of `xs:element` and `xs:complexType`.	#all, restriction, extension, restriction extension, [Empty].
Default Attributes	Specifies a set of attributes that apply to every complex Type in a schema document.	Any.
Xpath Default Namespace	The default namespace used when the XPath expression is evaluated.	##defaultNamespace, ##targetNamespace, ##local.
Version	Schema version	Any token.
ID	The schema id	Any ID.
Component	The edited component name.	Not editable property.
SystemID	The schema system id	Not editable property.

xs:element

Defines an element. An element declaration is an association of a name with a type definition, either simple or complex, an (optional) default value and a (possibly empty) set of identity-constraint definitions. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-element*.

An element by default displays the following properties when rendered in the diagram: *default*, *fixed*, *abstract* and *type*. When referenced or declared locally, the element graphical representation also contains the value for the *minOccurs* and *maxOccurs* properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the element are drawn using dotted lines if the element is optional.

xs:element properties

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

Property Name	Description	Possible Values	Mentions
Substitution Group	Qualified name of the head of the substitution group to which this element belongs.	All declared elements. For XML Schema 1.1 this property supports multiple values.	For global and reference elements
Abstract	Controls whether the element may be used directly in instance XML documents. When set to true, the element may still be used to define content models, but it must be substituted through a substitution group in the instance document.	true/false	For global elements and element references
Form	Defines if the element is "qualified" (i.e. belongs to the target namespace) or "unqualified" (i.e. doesn't belong to any namespace).	unqualified/qualified	Only for local elements
Nillable	When this attribute is set to true, the element can be declared as nil using an `xsi:nil` attribute in the instance documents.	true/false	For global elements and element references
Target Namespace	Specifies the target namespace for local element and attribute declarations. The namespace URI may be different from the schema target namespace. This property is available for local elements only.	Not editable property.	For all elements.
Block	Controls whether the element can be subject to a type or substitution group substitution. '#all' blocks any substitution, 'substitution' blocks any substitution through substitution groups and 'extension'/'restriction' block any substitution (both through `xsi:type` and substitution groups) by elements or types, derived respectively by extension or restriction from the type of the element. Its default value is defined by the `blockDefault` attribute of the parent `xs:schema`.	#all, restriction, extension,substitution, extension restriction, extension substitution, restriction substitution, restriction extension substitution	For global elements and element references

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

xs:attribute

Defines an attribute. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-attribute*.

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

xs:attribute properties

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

Property Name	Description	Possible Value	Mentions
	fixed attributes are mutually exclusive.		
Fixed	When specified, the value of the attribute is fixed and must be equal to this value. The default and fixed attributes are mutually exclusive.	Any string	For all local or global attributes. For references the value is from the referenced attribute.
Use	Possible usage of the attribute. Marking an attribute "prohibited" is useful to exclude attributes during derivations by restriction.	optional, required, prohibited	For local attributes
Form	Specifies if the attribute is qualified (i.e. must have a namespace prefix in the instance XML document) or not. The default value for this attribute is specified by the `attributeFormDefault` attribute of the `xs:schema` document element.	unqualified/qualified	For local attributes.
Inheritable	Specifies if the attribute is inheritable. Inheritable attributes can be used by <alternative> element on descendant elements.	true/false	For all local or global attributes. The default value is false. This property is available for XML Schema 1.1.
Target Namespace	Specifies the target namespace for local attribute declarations. The namespace URI may be different from the schema target namespace.	Any URI	Setting a target namespace for local attribute is useful only when restricts attributes of a complex type that is declared in other schema with different target namespace. This property is available for XML Schema 1.1.
ID	The component id.	Any id	For all attributes.
Component	The edited component name.	Not editable property.	For all attributes.
Namespace	The component namespace.	Not editable property.	For all attributes.
System ID	The component system id.	Not editable property.	For all attributes.

xs:attributeGroup

Defines an attribute group to be used in complex type definitions. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-attributeGroup*.

xs:attributeGroup properties

Property Name	Description	Possible Values	Mentions
Name	Attribute group name. Always required.	Any NCName for global attribute groups, all declared attribute groups for reference.	For all global or referenced attribute groups. If missing, will be displayed as '[attributeGroup]' in diagram.
ID	The component id.	Any id	For all attribute groups.
Component	The edited component name.	Not editable property.	For all attribute groups.
Namespace	The component namespace.	Not editable property.	For all attribute groups.
System ID	The component system id.	Not editable property.	For all attribute groups.

xs:complexType

Defines a top level complex type. Complex Type Definitions provide for: See more data at *http://www.w3.org/TR/xmlschema11-1/#element-complexType*.

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

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

xs:complexType properties

Property Name	Description	Possible Values	Mentions
Name	The name of the complex type. Always required.	Any NCName	Only for global complex types. If missing, will be displayed as '[complexType]' in diagram.
Base Type Definition	The name of the extended/restricted types.	Any from the declared simple or complex types.	For complex types with simple or complex content.
Derivation Method	The derivation method.	restriction/ extension	Only when base type is set. If the base type is a simple type, the derivation method is always extension.
Content	The content of the complex type.	simple/ complex	For complex types which extend/restrict a base type. It is automatically detected.
Content Mixed	Specifies if the complex content model will be mixed.	true/false	For complex contents.
Mixed	Specifies if the complex type content model will be mixed.	true/false	For global and anonymous complex types.

Property Name	Description	Possible Values	Mentions
Abstract	When set to `true`, this complex type cannot be used directly in the instance documents and needs to be substituted using an `xsi:type` attribute.	true/false	For global and anonymous complex types.
Block	Controls whether a substitution (either through a `xsi:type` or substitution groups) can be performed for a complex type, which is an extension or a restriction of the current complex type. This attribute can only block such substitutions (it cannot "unblock" them), which can also be blocked in the element definition. The default value is defined by the `blockDefault` attribute of `xs:schema`.	all, extension, restriction, extension restriction, [Empty]	For global complex types.
Final	Controls whether the complex type can be further derived by extension or restriction to create new complex types.	all, extension, restriction, extension restriction, [Empty]	For global complex types.
Default Attributes Apply	The `schema` element can carry a `defaultAttributes` attribute, which identifies an attribute group. Each `complexType` defined in the schema document then automatically includes that attribute group, unless this is overridden by the `defaultAttributesApply` attribute on the `complexType` element.	true/false	This property is available only for XML Schema 1.1.
ID	The component id.	Any id	For all complex types.
Component	The edited component name.	Not editable property.	For all complex types.
Namespace	The component namespace.	Not editable property.	For all complex types.
System ID	The component system id.	Not editable property.	For all complex types.

xs:simpleType

Defines a simple type. A simple type definition is a set of constraints on strings and information about the values they encode, applicable to the normalized value of an attribute information item or of an element information item with no

element children. Informally, it applies to the values of attributes and the text-only content of elements. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-simpleType*.

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

xs:simpleType properties

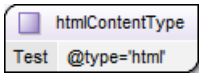
Name	Description	Possible Values	Scope
Name	Simple type name. Always required.	Any NCName.	Only for global simple types. If missing, will be displayed as '[simpleType]' in diagram.
Derivation	The simple type category: restriction, list or union.	restriction,list or union	For all simple types.
Base Type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

Name	Description	Possible Values	Scope
			datatype local definitions in the xs:union element. Both styles can be mixed.
Final	Blocks any further derivations of this datatype (by list, union, derivation or all).	#all, list, restriction, union, list restriction, list union, restriction union. In addition, [Empty] proposal is present for set empty string as value.	Only for global simple types.
ID	The component id.	Any id.	For all simple types
Component	The name of the edited component.	Not editable property.	Only for global and local simple types
Namespace	The component namespace.	Not editable property.	For global simple types.
System ID	The component system id.	Not editable property.	Not present for built-in simple types..

xs:alternative

The *type alternatives* mechanism allows you to specify type substitutions on an element declaration.

Note: `xs:alternative` is available for XML Schema 1.1.



xs:alternative properties

Name	Description	Possible Values
Type	Specifies type substitutions for an element, depending on the value of the attributes.	All declared or built-in types. In addition, the following anonymous types are available: [ST-restriction], [ST-union], [ST-list], [CT-anonymous], [CT-extension SC], [CT-restriction SC], [CT-restriction CC], [CT-extension CC].
Test	Specifies an XPath expression. If the XPath condition is valid, the specified type is selected as the element type. The expressions allowed are limited to a subset of XPath 2.0. Only the attributes of the current element and inheritable attributes from ancestor elements are accessible in the XPath expression. When you edit this property, the content completion list of proposals offers XPath expressions.	An XPath expression.
XPath Default Namespace	The default namespace used when the XPath expression is evaluated.	##defaultNamespace, ##targetNamespace, ##local.
ID	Specifies the component ID.	Any ID.
Component	Specifies the type of XML schema component.	Not editable property.

Name	Description	Possible Values
System ID	Points to the document location of the schema.	Not editable property.

xs:group

Defines a group of elements to be used in complex type definitions. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-group*.

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

xs:group properties

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

xs:include

Adds multiple schemas with the same target namespace to a document. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-include*.

xs:include properties

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

xs:import

Adds multiple schemas with different target namespace to a document. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-import*.

xs:import properties

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

xs:redefine

Redefines simple and complex types, groups, and attribute groups from an external schema. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-redefine*.

xs:redefine properties

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

xs:override

The override construct allows replacements of old components with new ones without any constraint. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-override*.

xs:override properties

Property Name	Description	Possible Values
Schema Location	Redefine schema location.	Any URI
ID	Redefine ID	Any ID

xs:notation

Describes the format of non-XML data within an XML document. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-notation*.

xs:notation properties

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

Property Name	Description	Possible values	Mentions
Namespace	The component namespace.	Not editable property.	For all notations.
System ID	The component system id.	Not editable property.	For all notations.

xs:sequence, xs:choice, xs:all

Figure 158: An `xs:sequence` in diagram

`xs:sequence` specifies that the child elements must appear in a sequence. Each child element can occur from 0 to any number of times. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-sequence*.

Figure 159: An `xs:choice` in diagram

`xs:choice` allows only one of the elements contained in the declaration to be present within the containing element. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-choice*.

Figure 160: An `xs:all` in diagram

`xs:all` specifies that the child elements can appear in any order. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-all*.

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

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

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

xs:any

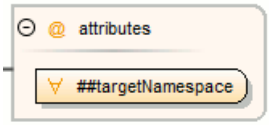
Enables the author to extend the XML document with elements not specified by the schema. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-any*.

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

xs:any properties

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

xs:anyAttribute



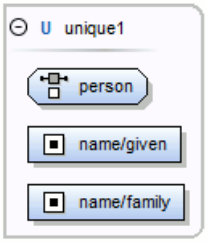
Enables the author to extend the XML document with attributes not specified by the schema. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-anyAttribute*.

xs:anyAttribute properties

Property Name	Description	Possible Value
Namespace	The list of allowed namespaces. The namespace attribute expects a list of namespace URIs. In this list, two values have a specific meaning: '##targetNamespace' stands for the target namespace, and '##local' stands for local attributes (without namespaces).	##any, ##other, ##targetNamespace, ##local or anyURI
Process Contents	Type of validation required on the elements allowed for this wildcard.	skip, lax, strict
ID	The component id.	Any ID.

Property Name	Description	Possible Value
Component	The name of the edited component.	Not editable property.
System ID	The component system id.	Not editable property.

xs:unique

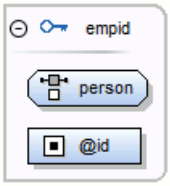


Defines that an element or an attribute value must be unique within the scope. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-unique*.

xs:unique properties

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

xs:key

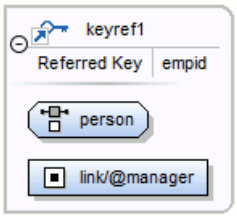


Specifies an attribute or element value as a key (unique, non-nullable and always present) within the containing element in an instance document. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-key*.

xs:key properties

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

xs:keyRef



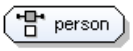
Specifies that an attribute or element value corresponds to that of the specified key or unique element. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-keyref*.

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

xs:keyRef properties

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

xs:selector



Specifies an XPath expression that selects a set of elements for an identity constraint. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-selector*.

xs:selector properties

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

xs:field

Specifies an XPath expression that specifies the value used to define an identity constraint. See more info at *http://www.w3.org/TR/xmlschema11-1/#element-field*.

xs:field properties

Property Name	Description	Possible Values
XPath	Relative XPath expression identifying the field(s) composing the key, key reference, or unique constraint.	An XPath expression.

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

xs:assert

Assertions provide a flexible way to control the occurrence and values of elements and attributes available in an XML Schema.

Note: `xs:assert` is available for XML Schema 1.1.

xs:assert properties

Property Name	Description	Possible Values
Test	Specifies an XPath expression. If the XPath condition is valid, the specified type is selected as the element type. The expressions allowed are limited to a subset of XPath 2.0. Only the attributes of the current element and inheritable attributes from ancestor elements are accessible in the XPath expression. When you edit this property, the content completion list of proposals offers XPath expressions.	An XPath expression.
XPath Default Namespace	The default namespace used when the XPath expression is evaluated.	##defaultNamespace, ##targetNamespace, ##local.
ID	Specifies the component ID.	Any ID.
Component	The edited component name.	Not editable property.
System ID	The component system id.	Not editable property.

xs:openContent

The `openContent` element enables instance documents to contain extension elements interleaved among the elements declared by the schema. You can declare open content for your elements at one place - within the complexType definition, or at the schema level.

For further details about the `openContent` component, go to *http://www.w3.org/TR/xmlschema11-1/#element-openContent*.

xs:openContent properties

Property Name	Description	Possible Value
Mode	Specifies where the extension elements can be inserted.	The value can be: "interleave", "suffix" or "none". The default value is "interleave".
ID	The component id.	Any ID.
Component	The edited component name.	Not editable property.

Property Name	Description	Possible Value
System ID	The component system id.	Not editable property.

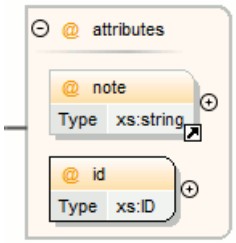
Note: This component is available for XML Schema 1.1 only. To change the version of the XML Schema, *open the **Preferences** dialog box* and go to **XML** > **XML Parser** > **XML Schema**.

Constructs Used to Group Schema Components

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

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

Attributes

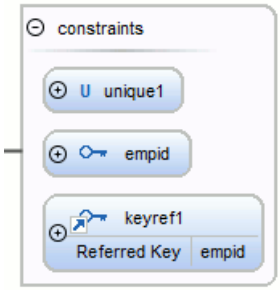


Groups all attributes and attribute groups belonging to a complex type.

Attributes properties

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

Constraints

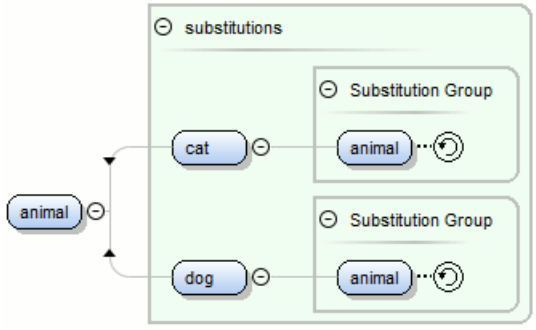


Groups all constraints (`xs:key`, `xs:keyRef` or `xs:unique`) belonging to an element.

Attributes properties

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

Substitutions



Groups all elements which can substitute the current element.

Attributes properties

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

Schema Validation

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

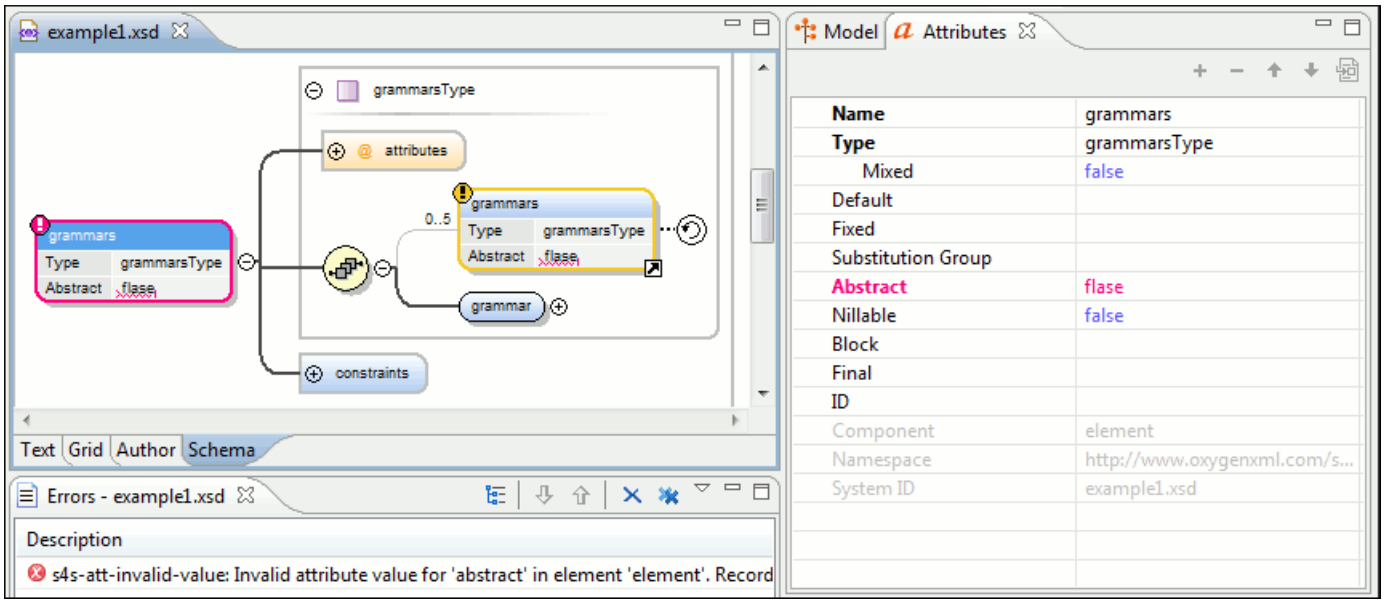


Figure 161: XML Schema Validation

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

- In the *Attributes View*.
- In the diagram by surrounding the component that has the error with a red border.

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

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

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

If you validate the entire schema using the **Validate** action from the **XML** menu or from the **Validation** toolbar drop-down menu, all validation errors will be presented in the **Errors** tab. To resolve an error, just click it (or double-click for errors located in other schemas) and the corresponding schema component will be displayed as the diagram root so that you can easily correct the error.

Important: If the schema imports only the namespace of other schema without specifying the schema location and a *catalog is set-up* that maps the namespace to a certain location both the validation and the diagram will correctly identify the imported schema.

Tip: If the validation action finds that the schema contains unresolved references, the application will suggest the use of validation scenarios, but only if the current edited schema is a XML Schema module.

Edit Schema Namespaces

You can use the **XML Schema Namespaces** dialog box to easily set a target namespace and define namespace mappings for a newly created XML Schema. In the **Design** mode these namespaces can be modified anytime by choosing **Edit Schema Namespaces** from the contextual menu. Also you can do that by double-clicking the schema root in the diagram.

The **XML Schema Namespaces** dialog box allows you to edit the following information:

- **Target namespace** - The target namespace of the schema.
- **Prefixes** - The dialog box displays a table with namespaces and the mapped prefixes. You can add a new prefix mapping or remove an already existing one.

Editing XML Schema in Text Editing Mode

This page is used for editing the XML Schema in a text mode. It offers powerful content completion support, a synchronized Outline view, and multiple *refactoring actions*. The Outline view has two display modes: the *standard outline* mode and the *components* mode.

A diagram of the XML Schema can be presented side by side with the text. To activate the diagram presentation, enable the *Show Full Model XML Schema diagram* check box from the *Diagram* preferences page.

Content Completion in XML Schema

The intelligent **Content Completion Assistant** available in Oxygen XML Editor plugin enables rapid, in-line identification and insertion of elements, attributes and attribute values valid in the editing context. All available proposals are listed in a pop-up menu displayed at the current cursor position.

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

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

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

When active, it displays a list of context-sensitive proposals valid at the current cursor position. Elements are selected 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, do one of the following:

- Press Enter or Tab key on your keyboard to insert both the start and end tags. The cursor is positioned inside the start tag, in a position suitable for inserting attributes.
- Press **Ctrl Enter (Meta Enter on OS X)** on your keyboard. Oxygen XML Editor plugin inserts both the start and end tags and positions the cursor between the tags, so you can start typing content.

Depending on the *selected schema version*, Oxygen XML Editor plugin populates the proposals list with information taken either from XML Schema 1.0 or 1.1.

Oxygen XML Editor plugin helps you to easily reference a component by providing the list of proposals (complex types, simple types, elements, attributes, groups, attribute groups, or notations) valid in the current context. The components are collected from the current file or from the imported/included schemas.

When editing `xs:annotation/xs:appinfo` elements of an XML Schema, the Content Completion Assistant proposes elements and attributes from a custom schema (by default ISO Schematron). This feature can be configured from the *XSD Content Completion* preferences page.

Highlight Component Occurrences

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

This feature is on by default. To configured it, *open the **Preferences** dialog box* and go to **Editor** > **Mark Occurrences**. A search can also be triggered with the **Search** > **Search Occurrences in File ()** contextual menu action. All matches are displayed in a separate tab of the **Results** view.

Editing XML Schema in the Master Files Context

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

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

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

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

- Correct validation of a module in the context of a larger schema structure.
- **Content Completion Assistant** displays all the referable components valid in the current context. This include components defined in modules other than the currently edited one.
- The **Outline** displays the components collected from the entire schema structure.

Searching and Refactoring Actions in XML Schemas

Search Actions

The following search actions can be applied on `attribute`, `attributeGroup`, `element`, `group`, `key`, `unique`, `keyref`, `notation`, `simple`, or `complex` types and are available from the **Search** submenu in the contextual menu of the current editor:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
- **Search Occurrences in File** - Searches all occurrences of the item at the cursor position in the currently edited file.

The following action is available from the **XSL** menu:

- **Show Definition** - Moves the cursor to the definition of the referenced XML Schema item.

  Note: You can also use the **Ctrl Single-Click (Command Single-Click on OS X)** shortcut on a reference to display its definition.

Refactoring Actions

The following refactoring actions can be applied on `attribute`, `attributeGroup`, `element`, `group`, `key`, `unique`, `keyref`, `notation`, `simple`, or `complex` types and are available from the **Refactoring** submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the cursor position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
- **Rename Component in** - Opens the **Rename *component_type*** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

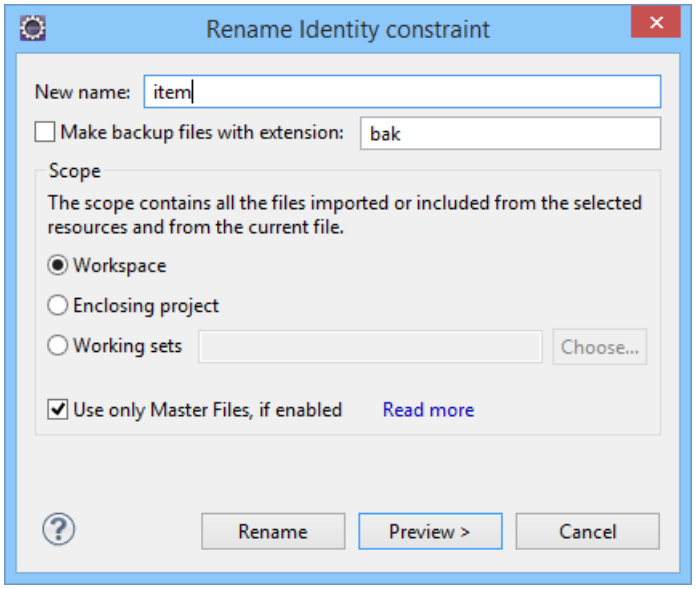


Figure 162: Rename Identity Constraint Dialog Box

XML Schema Outline View

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

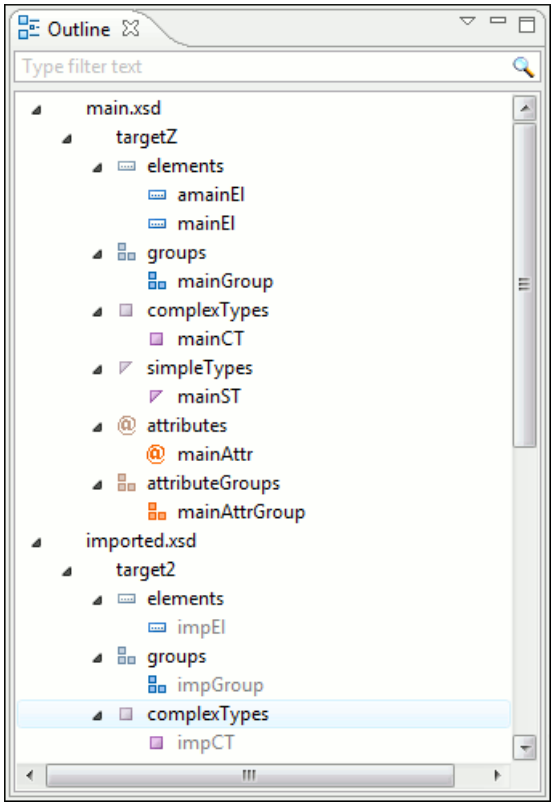


Figure 163: The Outline View for XML Schema

The **Outline** view provides the following options in the **View Menu** on the **Outline** view action bar:

Filter returns exact matches

The text filter of the **Outline** view returns only exact matches;

Selection update on cursor move

Allows a synchronization between **Outline** view and schema diagram. The selected view from the diagram is also selected in the **Outline** view.

Sort

Allows you to sort alphabetically the schema components.

Show all components

Displays all components that were collected starting from the main files. Components that are not referable from the current file are marked with an orange underline. To reference them, add an import directive with the `componentNS` namespace.

Show referable components

Displays all components (collected starting from the main files) that can be referenced from the current file. This option is set by default.

Show only local components

Displays the components defined in the current file only.

Group by location/namespace/type

These three operations allow you to group the components by location, namespace, or type. When grouping by namespace, the main schema target namespace is the first presented in the **Outline** view.

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

Remove (Delete)

Removes the selected item from the diagram.

 **Search References**

Searches all references of the item found at current cursor position in the defined scope, if any.

Search References in

Searches all references of the item found at current cursor position in the specified scope.

 **Component Dependencies**

Allows you to see the dependencies for the current selected component.

Resource Hierarchy

Allows you to see the hierarchy for the current selected resource.

Resource Dependencies

Allows you to see the dependencies for the current selected resource.

 **Rename Component in**

Renames the selected component.

 **Generate Sample XML Files**

Generate XML files using the current opened schema. The selected component is the XML document root.

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

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

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

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

The content of the **Outline** view and the editing area are synchronized. When you select a component in the **Outline** view, its definition is highlighted in the editing area.

Component Dependencies View for XML Schema

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

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

- Select the desired schema component in the editor.
- Choose the **Component Dependencies** action from the contextual menu.

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

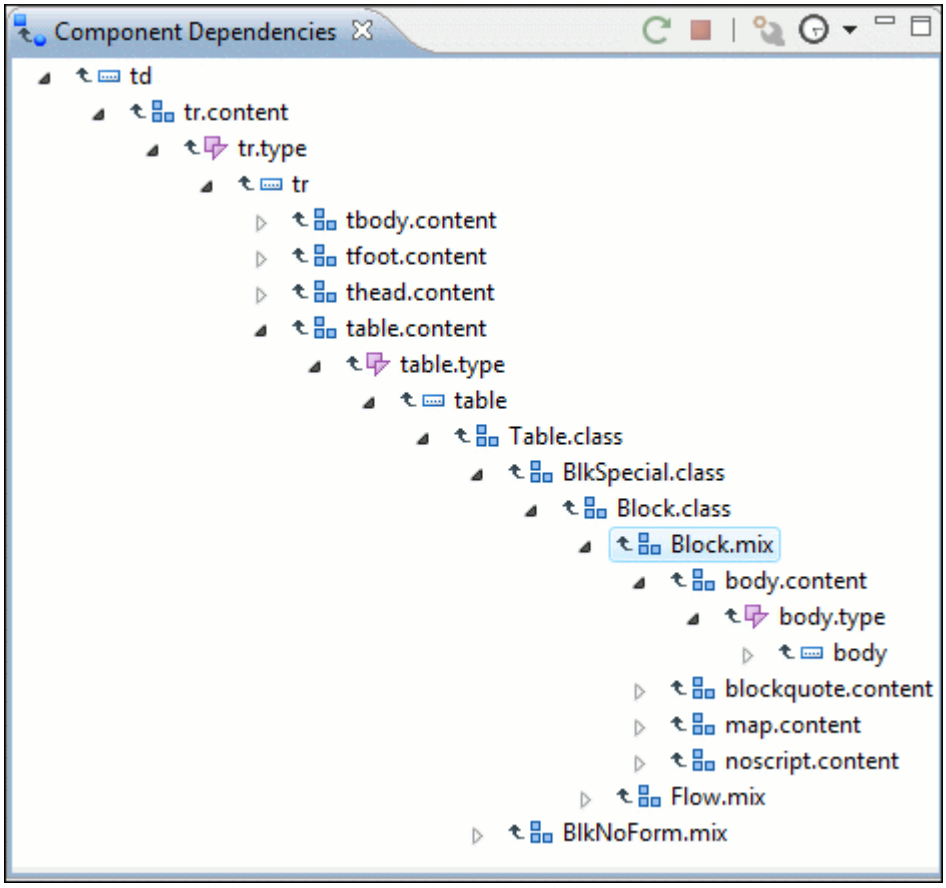


Figure 164: Component Dependencies View - Hierarchy for `xhtml11.xsd`

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

Refresh

Refreshes the dependencies structure.

Stop

Stop the dependencies computing.

Configure

Allows you to configure a search scope to compute the dependencies structure.

History

Contains a list of previously executed dependencies computations.

The contextual menu contains the following actions:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Shows the definition of the currently selected component in the dependencies tree.

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

XML Schema Quick Assist Support

The *Quick Assist* support improves the development work flow, offering fast access to the most commonly used actions when you edit XML Schema documents.

The *Quick Assist* feature is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the line number stripe on the left side of the editor. Also, you can invoke the quick assist menu by using the **Ctrl + 1** keys (**Meta 1** on Mac OS X) keyboard shortcuts.

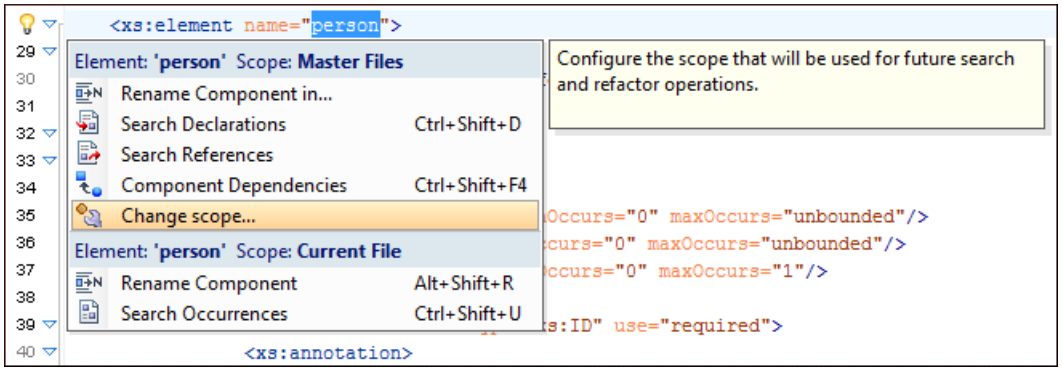


Figure 165: Quick Assist Support

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

Rename Component in

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

To watch our video demonstration about improving schema development using the **Quick Assist** action set, go to *http://oxygenxml.com/demo/Quick_Assist.html*.

XML Schema Resource Hierarchy / Dependencies View

The **Resource Hierarchy / Dependencies** view allows you to easily see the hierarchy / dependencies for an XML Schema, a *Relax NG schema* or an *XSLT stylesheet*. To open this view, go to **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** > **Resource Hierarchy/Dependencies**.

The **Resource Hierarchy / Dependencies** is useful when you want to start from an XML Schema (XSD) file and build and review the hierarchy of all the other XSD files that are imported, included or redefined in the given XSD file. The view is also able to build the tree structure, that is the structure of all other XSD files that import, include or redefine the given XSD file. The scope of the search is configurable (the current project, a set of local folders, etc.)

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

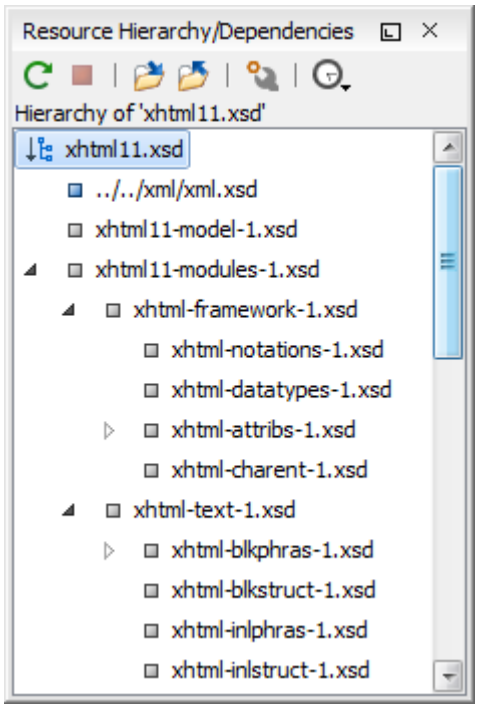


Figure 166: Resource Hierarchy/Dependencies View - Hierarchy for `xhtml11.xsd`

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

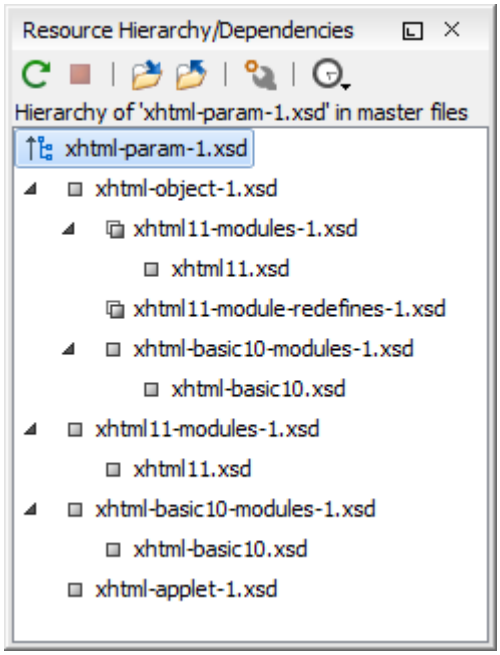


Figure 167: Resource Hierarchy/Dependencies View - Dependencies for `xhtml-param-1.xsd`

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

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

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

History

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

The contextual menu contains the following actions:

Open

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

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

Add to Master Files

Adds the currently selected resource in *the **Master Files** directory*.

Expand All

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

Collapse All

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

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

Note: The **Move resource** or **Rename resource** actions give you the option to *update the references to the resource*.

Moving/Renaming XML Schema Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

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

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

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

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

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

Generate Sample XML Files

Oxygen XML Editor plugin offers support to generate sample XML files both from XML schema 1.0 and XML schema 1.1, depending on the XML schema version set in **Preferences**.

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

The **Generate Sample XML Files** dialog box contains the following tabs:

- ***Schema***
- ***Options***
- ***Advanced***

To watch our video demonstration about generating sample XML files, go to *http://oxygenxml.com/demo/Generate_Sample_XML_Files.html*.

The Schema Tab

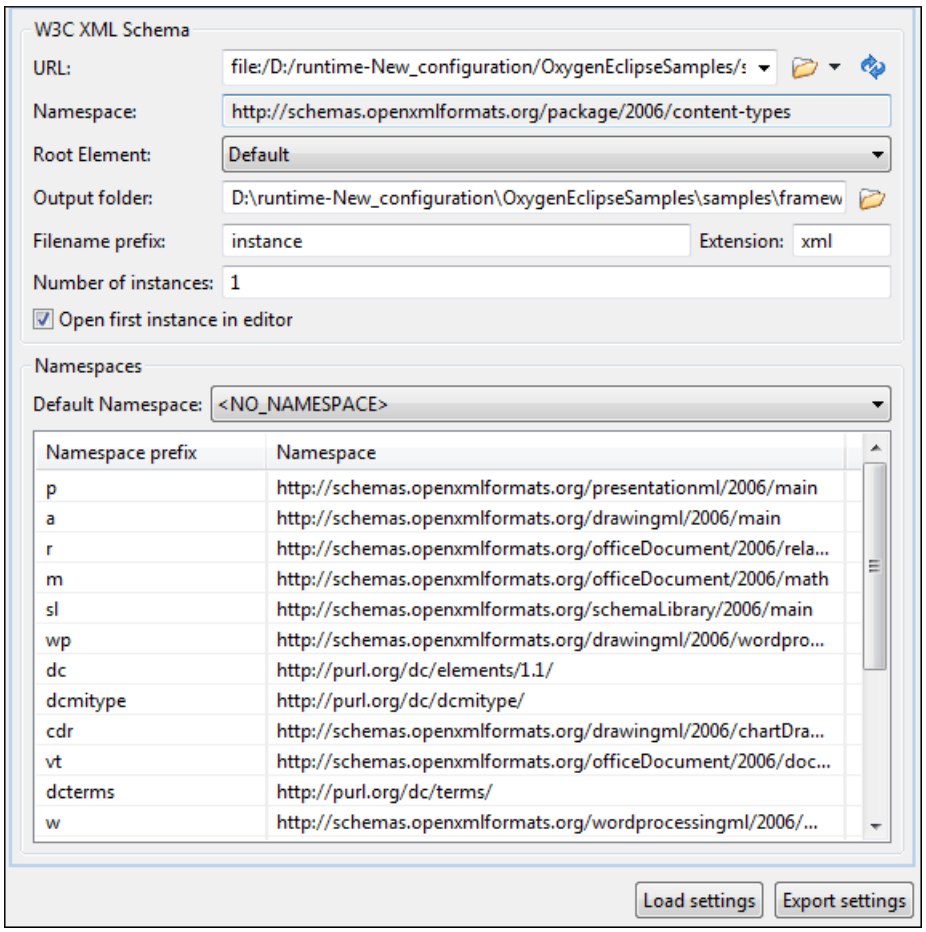


Figure 168: The Generate Sample XML Files Dialog Box

Complete the dialog box as follows:

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

The Options Tab

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

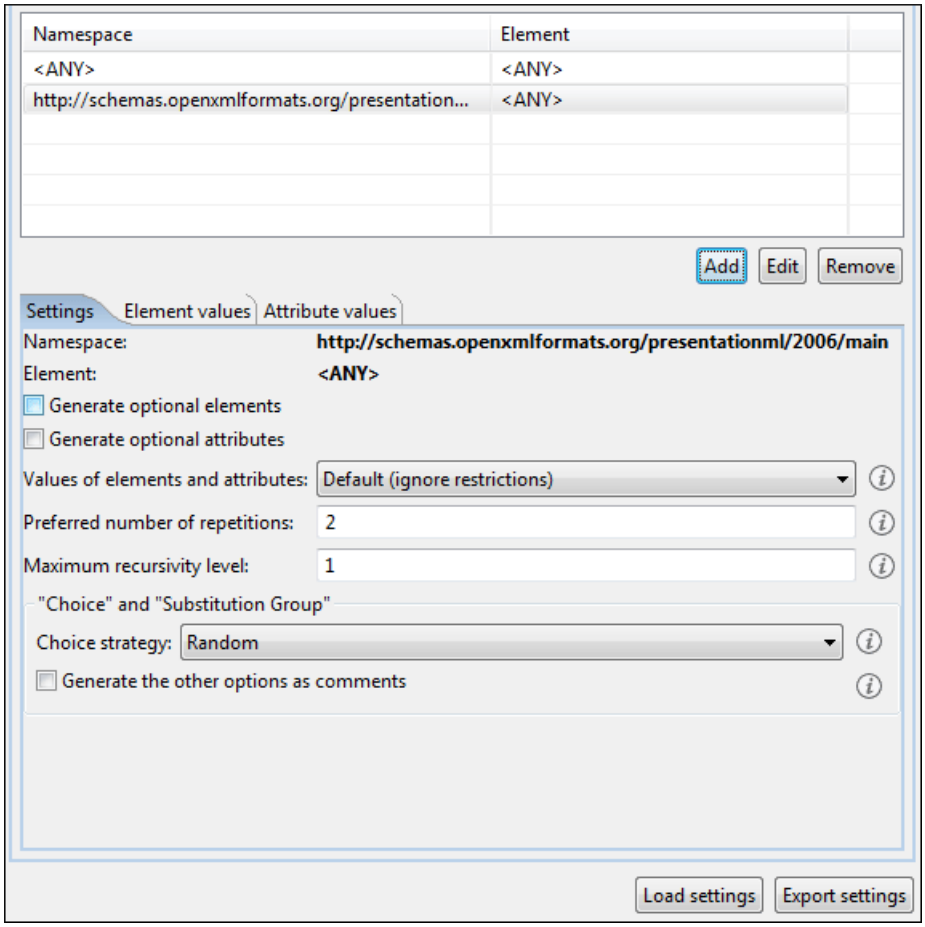


Figure 169: The Generate Sample XML Files Dialog Box

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

Important:

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

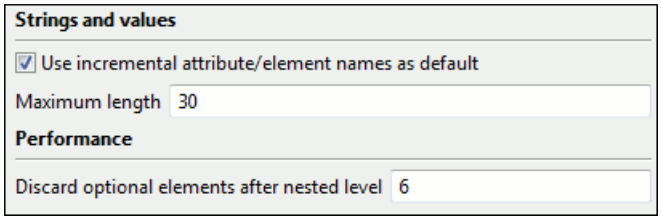
- At least one of the restrictions is a `regexp`.
- The value generated after applying the regexp does not match the restrictions imposed by one of the facets.

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

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

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

The Advanced Tab



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

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

Generating Documentation for an XML Schema

Oxygen XML Editor plugin can generate detailed documentation for the components of an XML Schema in HTML, PDF, DocBook, or other custom formats. You can select the components and the level of detail. The components are hyperlinked in both HTML and DocBook documents.

Note: You can generate documentation for both XML Schema version 1.0 and 1.1.

To generate documentation for an XML Schema document, select **XML Schema Documentation** from the **XML Tools** > **Generate Documentation** menu or from the **Generate XML Schema Documentation** action from the contextual menu of the **Navigator** view.

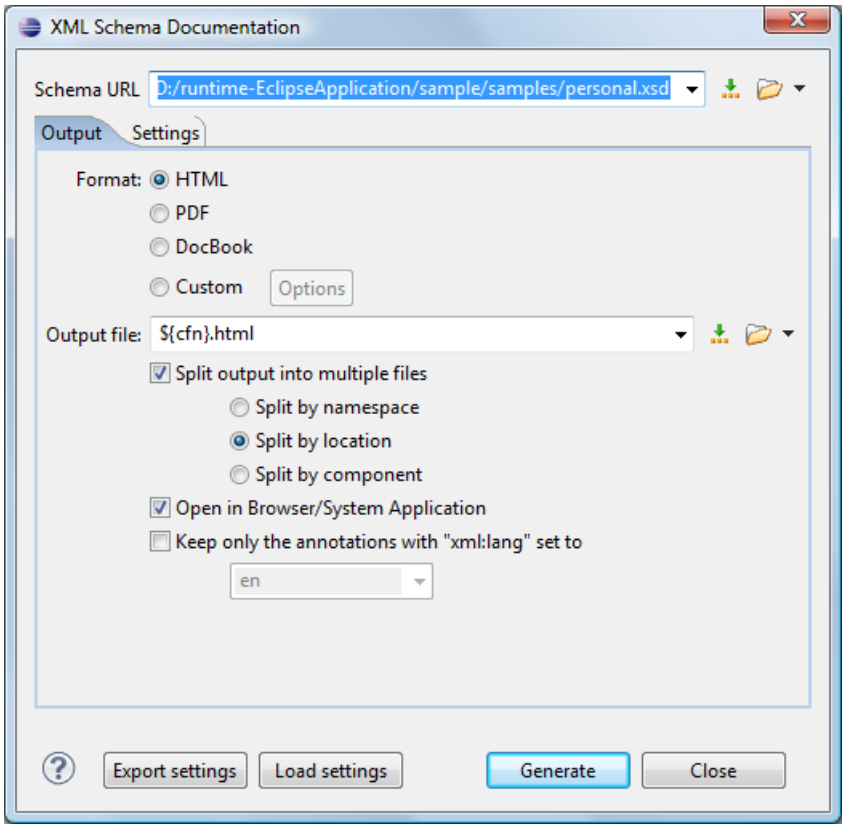


Figure 170: The XML Schema Documentation Dialog Box

The **Schema URL** field of the dialog box must contain the full path to the XML Schema (XSD) file for which you want to generate documentation. The schema may be a local or a remote file. You can specify the path to the schema by entering it in the text field, or by using the **Insert Editor Variables** button or the options in the **Browse** drop-down menu.

The Output Tab

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

- **Format** - Allows you to choose between the following formats:
  - **HTML** - The documentation is generated in *HTML output format*.
  - **PDF** - The documentation is generated in *PDF output format*.
  - **DocBook** - The documentation is generated in *DocBook output format*.
  - **Custom** - The documentation is generated in a *custom output format*, allowing you to control the output. Click the **Options** button to open a **Custom format options** dialog box where you can specify a custom stylesheet for creating the output. There is also an option to **Copy additional resources to the output folder** and you can

select the path to the additional **Resources** that you want to copy. You can also choose to keep the intermediate XML files created during the documentation process by deselecting the **Delete intermediate XML file** option.

- **Output file** - You can specify the path of the output file by entering it in the text field, or by using the **Insert Editor Variables** button or the options in the **Browse** drop-down menu.
- **Split output into multiple files** - Instructs the application to split the output into multiple files. You can choose to split them by namespace, location, or component name.
- **Open in Browser/System Application** - Opens the result in the system application associated with the output file type.

  Note: To set the browser or system application that will be used, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**. This will take precedence over the default system application settings.

- **Keep only the annotations with xml:lang set to** - The generated output will contain only the annotations with the `xml:lang` attribute set to the selected language. If you choose a primary language code (for example, **en** for English), this includes all its possible variations (**en-us**, **en-uk**, etc.).

The Settings Tab

When you generate documentation for an XML schema you can choose what components to include in the output and the details to be included in the documentation.

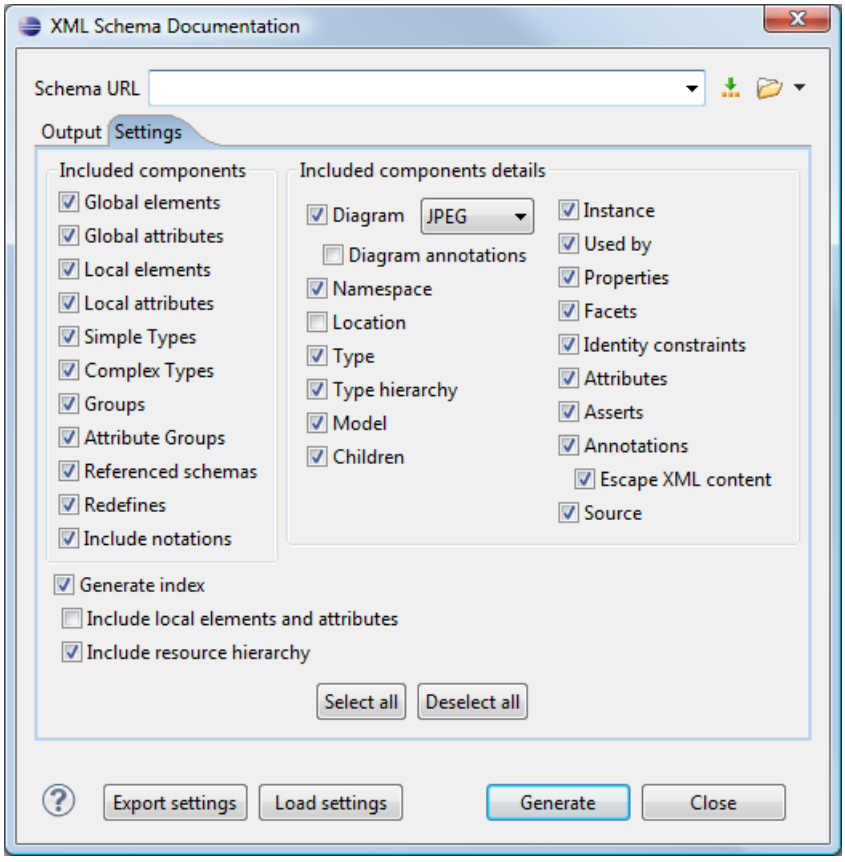


Figure 171: The Settings Tab of the XML Schema Documentation Dialog Box

The **Settings** tab allows you to choose whether or not to include the following components: **Global elements**, **Global attributes**, **Local elements**, **Local attributes**, **Simple Types**, **Complex Types**, **Groups**, **Attribute Groups**, **Redefines**, **Referenced schemas**, **Include notations**.

You can choose whether or not to include the following other details:

- **Diagram** - Displays the diagram for each component. You can choose the image format (JPEG, PNG, SVG) to use for the diagram section. The generated diagrams are dependent on the options from the *Schema Design Properties* page.
  - **Diagram annotations** - This option controls whether the annotations of the components presented in the diagram sections are included.
- **Namespace** - Displays the namespace for each component.
- **Location** - Displays the schema location for each component.
- **Type** - Displays the component type if it is not an anonymous one.
- **Type hierarchy** - Displays the types hierarchy.
- **Model** - Displays the model (sequence, choice, all) presented in BNF form. Different separator characters are used depending on the information item used:
  - `xs:all` - its children will be separated by space characters.
  - `xs:sequence` - its children will be separated by comma characters.
  - `xs:choice` - its children will be separated by | characters.
- **Children** - Displays the list of component's children.
- **Instance** - Displays an XML instance generated based on each schema element.
- **Used by** - Displays the list of all the components that reference the current one. The list is sorted by component type and name.
- **Properties** - Displays some of the component's properties.
- **Facets** - Displays the facets for each simple type
- **Identity constraints** - Displays the identity constraints for each element. For each constraint there are presented the name, type (unique, key, keyref), reference attribute, selector and field(s).
- **Attributes** - Displays the attributes for the component. For each attribute there are presented the name, type, fixed or default value, usage and annotation.
- **Asserts** - Displays the **assert** elements defined in a complex type. The test, XPath default namespace, and annotation are presented for each assert.
- **Annotations** - Displays the annotations for the component. If you choose **Escape XML Content**, the XML tags are present in the annotations.
- **Source** - Displays the text schema source for each component.
- **Generate index** - Displays an index with the components included in the documentation.
  - **Include local elements and attributes** - If checked, local elements and attributes are included in the documentation index.
  - **Include resource hierarchy** - Specifies whether the resource hierarchy for an XML Schema documentation is generated.

Export settings - Save the current settings in a settings file for further use (for example, with the exported settings file you can generate the same *documentation from the command line interface*.)

Load settings - Reloads the settings from the exported file.

Generate - Use this button to generate the XML Schema documentation.

Output Formats for Generating XML Schema Documentation

XML Schema documentation can be generated in HTML, PDF, DocBook, or a custom format. You can choose the format from the *Schema Documentation* dialog box. For the PDF and DocBook formats, the option to split the output in multiple files is not available.

HTML Output Format

The XML Schema documentation generated in HTML format contains images corresponding to the same schema definitions as the ones displayed by *the schema diagram editor.* These images are divided in clickable areas that are linked to the definitions of the names of types or elements. The documentation of a definition includes a **Used By** section

with links to the other definitions that reference it. If the **Escape XML Content** option is unchecked, the HTML or XHTML tags used inside the `xs:documentation` elements of the input XML Schema for formatting the documentation text (for example `<b>`, `<i>`, `<u>`, `<ul>`, `<li>`, etc.) are rendered in the generated HTML documentation.

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

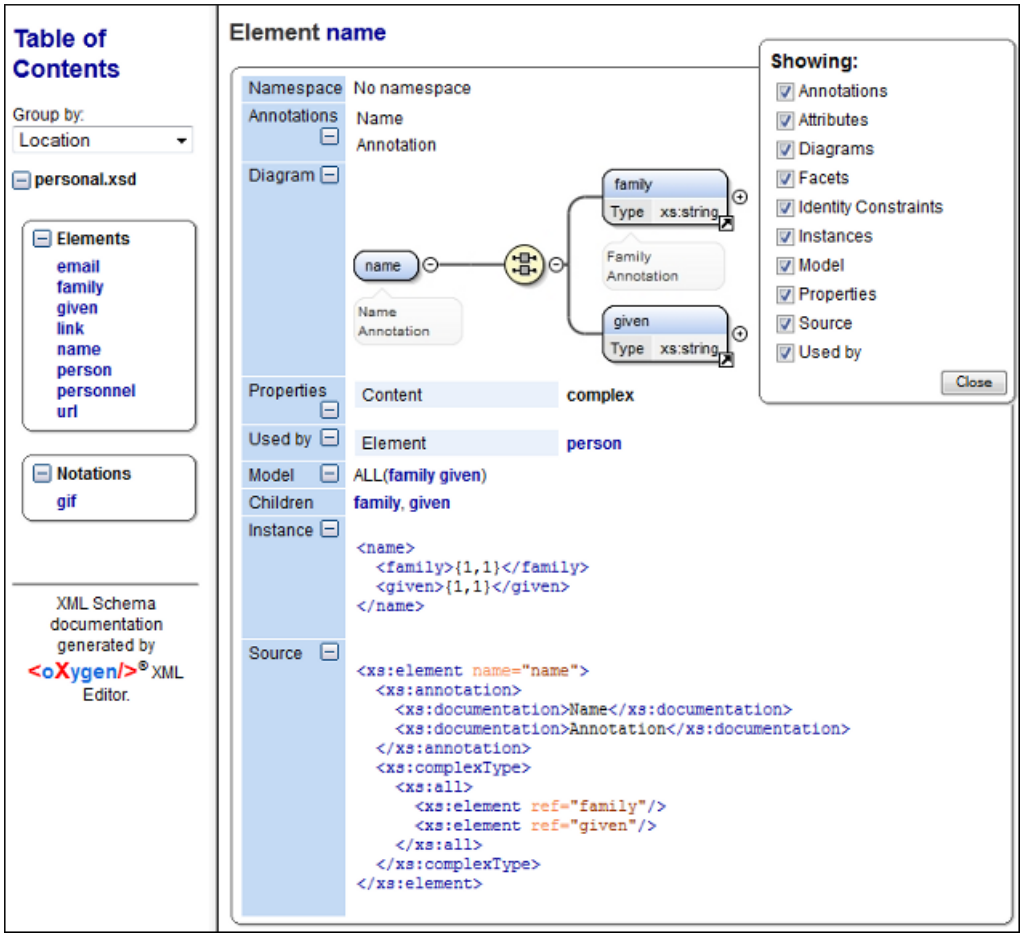


Figure 172: XML Schema Documentation Example

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

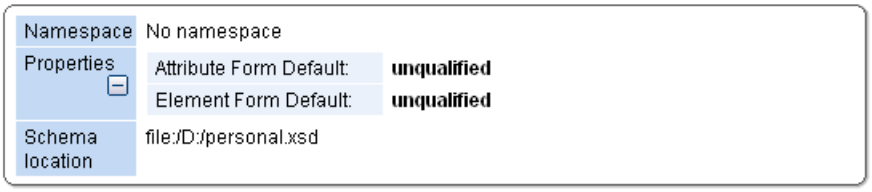


Figure 173: Information About a Schema

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

After the documentation is generated, you can collapse or expand details for some schema components by using the **Showing** options or the **Collapse** or **Expand** buttons.

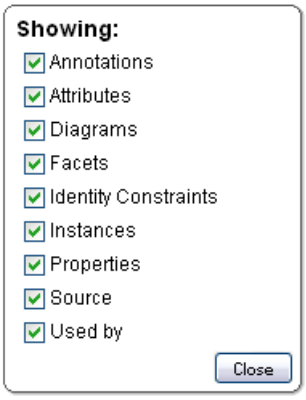


Figure 174: The Showing Options

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

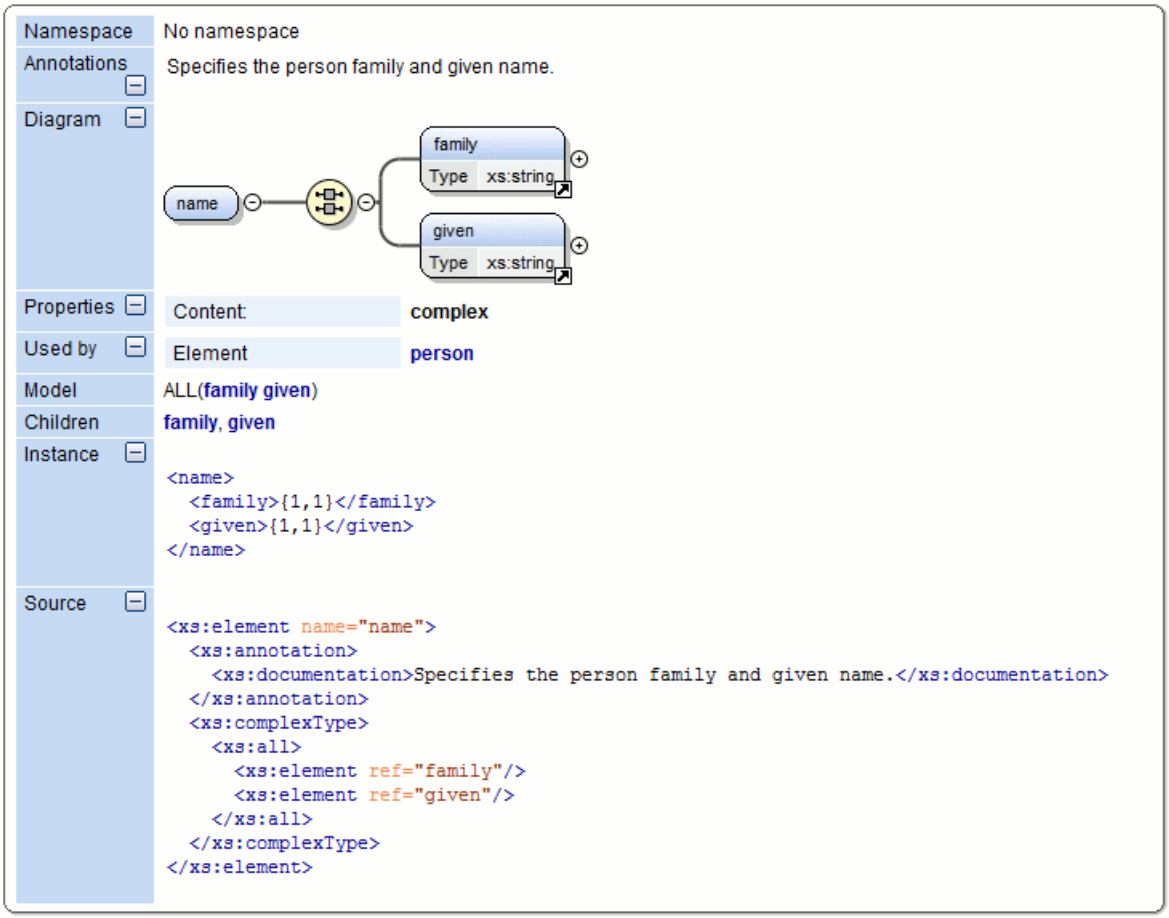
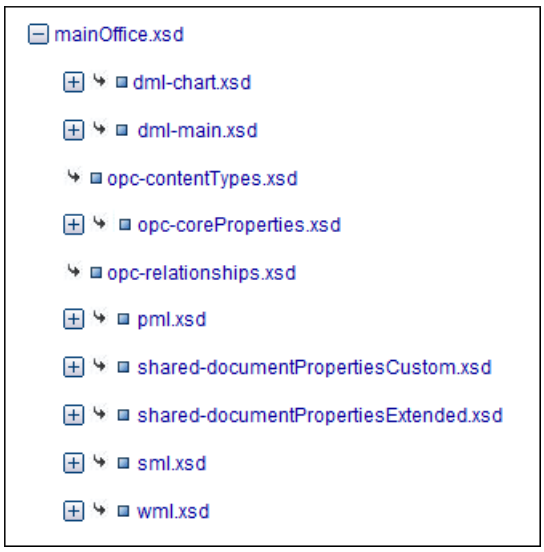


Figure 175: Documentation for a Schema Component

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



PDF Output Format

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

For information about customizing the PDF output, see the *Customizing the PDF Output of Generated XML Schema Documentation* on page 319 topic.

DocBook Output Format

If you generate the documentation in DocBook output format, the documentation is generated as a DocBook XML file. You can then apply a predefined transformation scenario (such as, *DocBook PDF* or *DocBook HTML*) on the output file, or configure your own transformation scenario for it to convert it into whatever format you desire.

Custom Output Format

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

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

Customizing the PDF Output of Generated XML Schema Documentation

To customize the PDF output of the generated XML Schema documentation, use the following procedure:

1. Customize the `[OXYGEN_DIR]/frameworks/schema_documentation/xsl/xsdDocDocbook.xsl` stylesheet to include the content that you want to add in the PDF output. Add the content in the XSLT template with the `match="schemaDoc"` attribute between the `info` and `xsl:apply-templates` elements, as commented in the following example:

```
<info>
    <pubdate><xsl:value-of select="format-date(current-date(),'[Mn] [D], [Y]', 'en', (), ())"/></pubdate>
</info>
   <!-- Add the XSLT template content with the match="schemaDoc" attribute here -->
<xsl:apply-templates select="schemaHierarchy"/>
```

Note: The content that you insert here has to be wrapped in DocBook markup. For details about working with DocBook see the following video demonstration: *http://www.oxygenxml.com/demo/DocBook_Editing_in_Author.html*.

2. Create an intermediary file that holds the content of your XML Schema documentation.
   a. Go to **Tools** > **Generate Documentation** > **XML Schema Documentation**.
   b. Select **Custom** for the output format and click the **Options** button.
   c. In the **Custom format options** dialog box, do the following:
      a. Enter the customized stylesheet in the **Custom XSL** field (`[OXYGEN_DIR]/frameworks/schema_documentation/xsl/xsdDocDocbook.xsl`).
      b. Enable the **Copy additional resources to the output folder** option, and leave the default selection in the **Resources** field.
      c. Click **OK**.
   d. When you return to the **XML Schema Documentation** dialog box, just press the **Generate** button to generate a DocBook XML file with an intermediary form of the Schema documentation.
3. Create the final PDF document.
   a. Use the  **Configure Transformation Scenario(s)** action from the toolbar or the **XML** menu, click **New**, and select **XML transformation with XSLT**.
   b. In the **New Scenario** dialog box, go to the **XSL URL** field and choose the `[OXYGEN_DIR]/frameworks/docbook/oxygen/xsdDocDocbookCustomizationFO.xsl` file.
   c. Go to the **FO Processor** tab and enable the **Perform FO Processing** and **XSLT result as input** options.
   d. Go to the **Output** tab and select the directory where you want to store the XML Schema documentation output and click **OK**.
   e. Click **Apply Associated**.

Generating XML Schema Documentation From the Command-Line Interface

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

- `schemaDocumentation.bat` on Windows.
- `schemaDocumentation.sh` on OS X / Unix / Linux.

The script is located in the Oxygen XML Editor plugin installation folder. The script can be integrated in an external batch process launched from the command-line interface. The script accepts a variety command line arguments that allow you to configure the documentation.

The accepted syntax and arguments are as follows:

```
schemaDocumentation schemaFile [ [-cfg:configFile] | [ [-out:outputFile]
[-format:<value>] [-xsl:<xslFile>] [-split:<value>] [-openInBrowser:<value>] ] |
--help | -help | --h | -h ]
```

schemaFile

The XML Schema file.

-cfg:configfile

The exported configuration file. It contains the output file, output format options, split method, and some advanced options regarding the included components and components details. If an external configuration file is specified, all other supplied arguments except for the XML Schema file will be ignored.

-out:outputFile

The file where the generated documentation will be saved. By default, it is the name of the schema file with an *html* extension.

-format:<value>

The output format type used when generating the documentation. Possible values are as follows:

- `html` - To generate documentation in HTML format.

- `pdf` - To generate documentation in PDF format.
- `docbook` - To generate documentation in DocBook format.
- `custom` - To generate documentation in a custom format.

-xsl:<xslFile>

The XSL file to be applied on the intermediate XML format. If there is no XSL file provided, the result will be in the HTML format.

-split:<value>

The split method used when generating the documentation. Splitting is recommended for large schemas. Possible values are as follows:

- `none` (default value) - To generate one single output file.
- `namespace` - To generate an output file for every namespace in the schema.
- `component` - To generate an output file for every component in the schema.
- `location` - To generate an output file for every schema location.

-openInBrowser:<value>

Opens the result of the transformation in a browser or system application. Possible values are `true` or `false` (default value).

--help | -help | --h | -h

Displays the available options.

Example of the script in a Windows command line:

```
schemaDocumentation example.xsd -out:schemaDocumentation.html -format:custom -xsl:example.xsl
 -split:namespace
```

Convert Database Structure to XML Schema

Oxygen XML Editor plugin includes a tool that allows you to create an XML Schema from the structure of a database. To convert a database structure to an XML Schema, use the following procedure:

1. Select the **Convert DB Structure to XML Schema** action from the **Tools** menu.
   The **Convert DB Structure to XML Schema** dialog box is opened and your current database connections are displayed in the **Connections** section.
2. If the database source is not listed, click the **Configure Database Sources** button to open the ***Data Sources*** *preferences page* where you can configure data sources and connections.
3. In the **Format for generated schema** section, select one of the following formats:
   - **Flat schema** - A flat structure that resembles a tree-like view of the database without references to elements.
   - **Hierarchical schema** - Display the table dependencies visually, in a type of tree view where dependent tables are shown as indented child elements in the content model. Select this option if you want to configure the database columns of the tables to be converted.
4. Click **Connect**
   The database structure is listed in the **Select database tables** section according to the format you chose.
5. Select the database tables that you want to be included in the XML Schema.
6. If you selected **Hierarchical schema** for the format, you can configure the database columns.
   a) Select the database column you want to configure.
   b) In the **Criterion** section you can choose to convert the selected database column as an **Element**, **Attribute**, or to be **Skipped** in the resulting XML Schema.
   c) You can also change the name of the selected database column by changing it in the **Name** text field.
7. Click **Generate XML Schema**.
   The database structure is converted to an XML Schema and it is opened for viewing and editing.

Flatten an XML Schema

You can organize an XML schema on several levels linked by `xs:include` and `xs:import` statements. In some cases, working on such a schema as on a single file is more convenient.

The **Flatten Schema** operation allows you to flatten an entire hierarchy of XML schemas. Starting with the main XML schema, Oxygen XML Editor plugin calculates its hierarchy by processing the `xs:include` and `xs:import` statements. This action is available from the contextual menu of the editor.

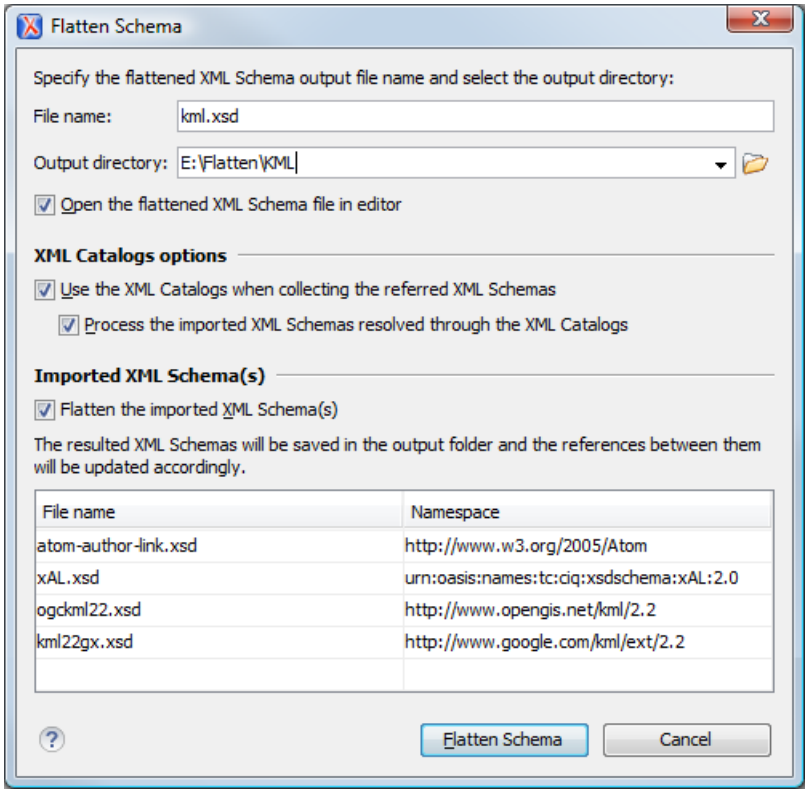


Figure 176: Flatten Schema Dialog Box

For the main schema file and for each imported schema, a new flattened schema is generated in the output folder. These schemas have the same name as the original ones.

Note: If necessary, the operation renames the resulted schemas to avoid duplicated file names.

Note: You can choose the output folder and the name of each generated schema file.

A flattened XML schema is obtained by recursively adding the components of the included schemas into the main one. This means Oxygen XML Editor plugin replaces the `xs:include`, `xs:redefine`, and `xs:override` elements with the ones coming from the included files.

The following options are available in the **Flatten Schema** dialog box:

- **Open the flattened XML Schema file in editor** - opens the main flattened schema in the editing area after the operation completes
- **Use the XML Catalogs when collecting the referenced XML Schemas** - enables resolving the imported and included schemas through the available XML Catalogs;

  Note: Changing this option triggers the recalculation of the dependencies graph for the main schema.

- **Process the imported XML Schemas resolved through the XML Catalogs** - specifies whether the imported schemas that were resolved through an XML Catalog are flattened
- **Flatten the imported XML Schema(s)** - specifies whether the imported schemas are flattened.

Note: For the schemas skipped by the flatten operation, no files are created in the output folder and the corresponding import statements remain unchanged.

To flatten a schema from the command line, use the following command:

- `flattenSchema.bat` on Windows
- `sh flattenSchemaMac.sh` on OS X
- `sh flattenSchema.sh` on Unix/Linux

The command line accepts the following parameters:

- `-in:inputSchemaURL` - the input schema URL
- `-outDir:outputDirectory` - the directory where the flattened schemas should be saved
- `-flattenImports:<boolean_value>` - controls if the imported XML Schemas should be flattened or not. The default value `true`
- `-useCatalogs:<boolean_value>` - controls if the references to other XML Schemas should be resolved through the available XML Catalogs. The default value `false`
- `-flattenCatalogResolvedImports:<boolean_value>` - controls if the imported schemas that were resolved through the XML Catalogs should be flattened or not

  Note: This option is used only when `-useCatalogs` is set to `true`. The default value is `true`.

- `-verbose` - provides information about the current flatten XML Schema operation
- `--help | -help | --h | -h` - prints the available parameters for the operation

Command Line Example for Windows

```
flattenSchema.bat -in:http://www.w3.org/MarkUp/SCHEMA/xhtml11.xsd
-outDir:mySchemas/flattened/xhtml -flattenImports:true -useCatalogs:true
-flattenCatalogResolvedImports:true -verbose
```

Command Line Example for OS X

```
sh flattenSchemaMac.sh -in:http://www.w3.org/MarkUp/SCHEMA/xhtml11.xsd
-outDir:mySchemas/flattened/xhtml -flattenImports:true -useCatalogs:true
-flattenCatalogResolvedImports:true -verbose
```

Command Line Example for Unix/Linux

```
sh flattenSchema.sh -in:http://www.w3.org/MarkUp/SCHEMA/xhtml11.xsd
-outDir:mySchemas/flattened/xhtml -flattenImports:true -useCatalogs:true
-flattenCatalogResolvedImports:true -verbose
```

XML Schema Regular Expressions Builder

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

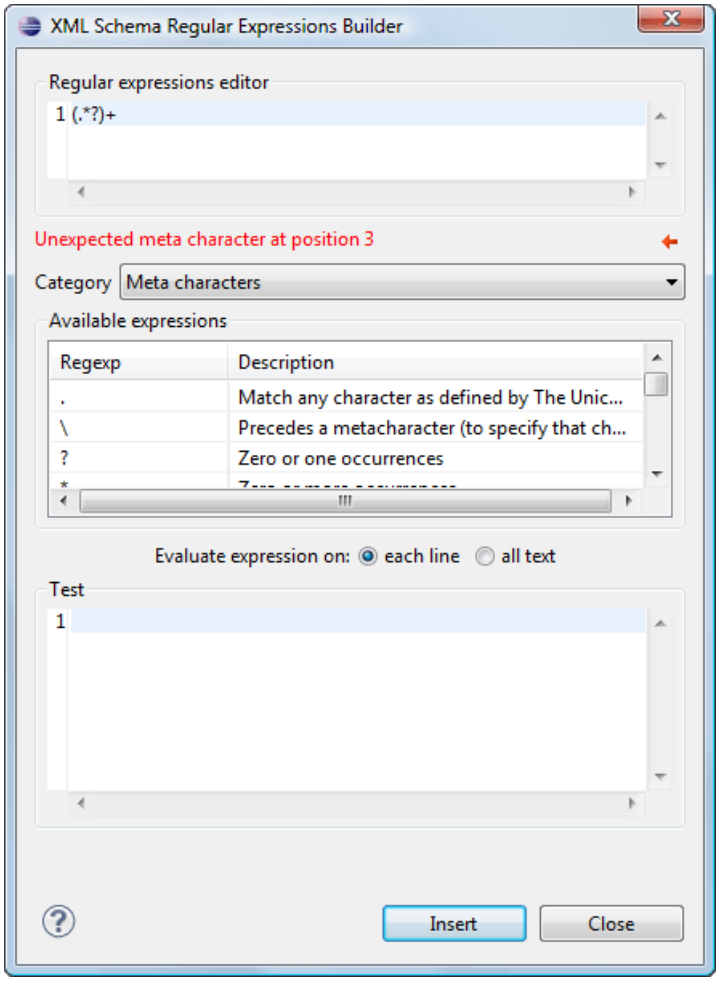


Figure 177: XML Schema Regular Expressions Builder Dialog Box

The dialog box contains the following sections:

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

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

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

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

XML Schema 1.1

Oxygen XML Editor plugin offers full support for XML Schema 1.1, including:

- XML Documents Validation and Content Completion Based on XML Schema 1.1.
- XML Schema 1.1 Validation and Content Completion.
- Editing XML Schema 1.1 files in the Schema **Design** mode.
- The Flatten Schema action.
- Resource Hierarchy/Dependencies and Refactoring Actions.
- Master Files.
- Generating Documentation for XML Schema 1.1.
- Support for generating XML instances based on XML Schema.
- Support for validating XML documents with an NVDL schema that contains an XML Schema 1.1 validation step.

Note: To enable XML Schema 1.1 validation in NVDL, you need to pass the following option to the validation engine to specify the schema version: `http://www.thaiopensource.com/validate/xsd-version` (the possible values are `1.0` or `1.1`.

Note: To enable the full XPath expression in assertions and type alternatives, you need to set the `http://www.thaiopensource.com/validate/full-xpath` option.

XML Schema 1.1 is a superset of XML Schema 1.0, that offers lots of new powerful capabilities.

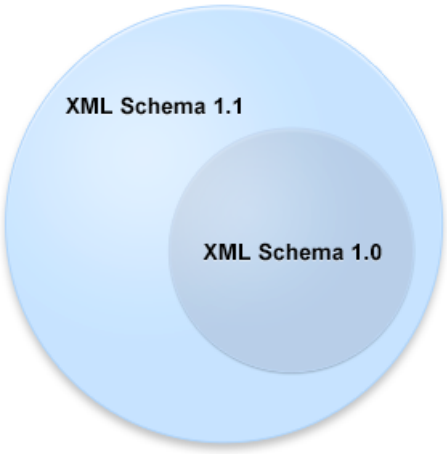


Figure 178: XML Schema 1.1

The significant new features in XSD 1.1 are:

- **Assertions** - support to define assertions against the document content using XPath 2.0 expressions (an idea borrowed from Schematron).

- **Conditional type assignment** - the ability to select the type against which an element is validated based on the values of the attribute of the element.
- **Open content** - content models are able to use the `openContent` element to specify content models with *open content*. These content models allow elements not explicitly mentioned in the content model to appear in the document instance. It is as if wildcards were automatically inserted at appropriate points within the content model. A schema document wide default may be set, which causes all content models to be open unless specified otherwise.

To see the complete list with changes since version 1.0, go to *http://www.w3.org/TR/xmlschema11-1/#ch_specs*.

XML Schema 1.1 is intended to be mostly compatible with XML Schema 1.0 and to have approximately the same scope. It also addresses bug fixes and brings improvements that are consistent with the constraints on scope and compatibility.

Note: An XML document conforming to a 1.0 schema can be validated using a 1.1 validator, but an XML document conforming to a 1.1 schema may not validate using a 1.0 validator.

If you are constrained to use XML Schema 1.0 (for example if you develop schemas for a server that uses an XML Schema 1.0 validator which cannot be updated), change the default XML Schema version to 1.0. If you keep the default XML Schema version set to 1.1, the content completion window presents XML Schema 1.1 elements that you can insert accidentally in an 1.0 XML Schema. So even if you make a document invalid conforming with XML Schema 1.0, the validation process does not signal any issues.

To change the default XML Schema version, *open the **Preferences** dialog box* and go to **XML** > **XML Parser** > **XML Schema**.

To watch our video demonstration about the XML Schema 1.1 support, go to *http://oxygenxml.com/demo/XML_Schema_11.html*.

Setting the XML Schema Version

Oxygen XML Editor plugin lets you set the version of the XML Schema you are editing either in the **XML Schema** preferences page, or through the versioning attributes. If you want to use the versioning attributes, set the `minVersion` and `maxVersion` attributes, from the *http://www.w3.org/2007/XMLSchema-versioning* namespace, on the `schema` root element.

Note: The versioning attributes take priority over the XML Schema version defined in the preferences page.

Table 6: Using the `minVersion` and `maxVersion` Attributes to Set the XML Schema Version

Versioning Attributes	XML Schema Version
`minVersion` = "1.0" `maxVerion` = "1.1"	1.0
`minVersion` = "1.1"	1.1
`minVersion` = "1.0" `maxVerion` = greater than "1.1"	the XML Schema version defined in the **XML Schema** preferences page.
Not set in the XML Schema document.	the XML Schema version defined in the **XML Schema** preferences page.

To change the XML Schema version of the current document, use the **Change XML Schema version** action from the contextual menu. This is available both in the **Text** mode, and in the **Design** mode and opens the **Change XML Schema version** dialog box. The following options are available:

- **XML Schema 1.0** - Inserts the `minVersion` and `maxVersion` attributes on the `schema` element and gives them the values "1.0" and "1.1" respectively. Also, the namespace declaration (*xmlns:vc=http://www.w3.org/2007/XMLSchema-versioning*) is inserted automatically if it does not exist.
- **XML Schema 1.1** - Inserts the `minVersion` attribute on the `schema` element and gives it the value "1.1". Also, removes the `maxVersion` attribute if it exists and adds the versioning namespace (*xmlns:vc=http://www.w3.org/2007/XMLSchema-versioning*) if it is not declared.

- **Default XML Schema version** - Removes the `minVersion` and `maxVersion` attributes from the `schema` element. The default schema version, defined in the **XML Schema** preferences page, is used.

Note: The **Change XML Schema version** action is also available in the informative panel presented at the top of the edited XML Schema. If you close this panel, it will no longer appear until you restore Oxygen XML Editor plugin to its default options.

Oxygen XML Editor plugin automatically uses the version set through the versioning attributes, or the default version if the versioning attributes are not declared, when proposing content completion elements and validating an XML Schema. Also, the XML instance validation against an XML Schema is aware of the versioning attributes defined in the XML Schema.

When you generate sample XML files from an XML Schema, Oxygen XML Editor plugin takes into account the `minVersion` and `maxVersion` attributes defined in the XML Schema.

Linking Between Development and Authoring

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

Editing XQuery Documents

This section explains the features of the XQuery editor and how to use them.

XQuery Outline View

The XQuery document structure is presented in the XQuery **Outline** view. The outline tree presents the list of all the components (namespaces, imports, variables, and functions) from both the edited XQuery file and its imports. It allows quick access to components and can be opened from **Window** > **Show View** > **Outline**.

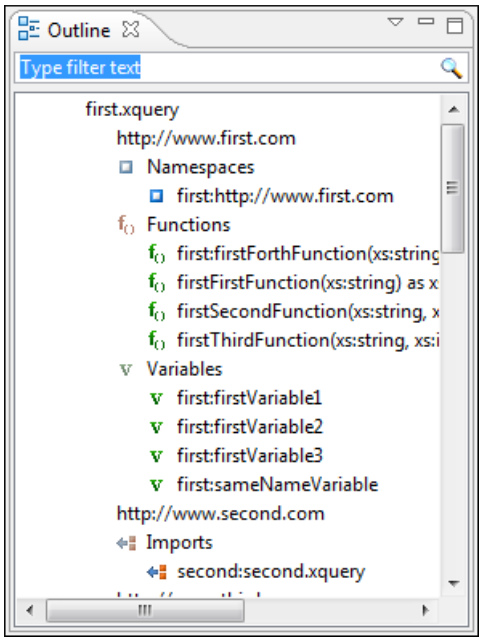


Figure 179: XQuery Outline View

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

Selection update on cursor move

Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the cursor moves or the changes performed in the XQuery editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

Sort

Allows you to alphabetically sort the XQuery components.

Show all components

Displays all collected components starting from the current file. This option is set by default.

Show only local components

Displays the components defined in the current file only.

Group by location/namespace/type

Allows you to group the components by location, namespace, and type. When grouping by namespace, the main XQuery module namespace is presented first in the **Outline** view.

If you know the component name, you can search it in the **Outline** view by typing its name in the filter text field from the top of the view or directly on the tree structure. When you type the component name in the filter text field you can switch to the tree structure using the arrow keys of the keyboard, **(Enter)**, **(Tab)**, **(Shift-Tab)**. To switch from tree structure to the filter text field, you can use **(Tab)**, **(Shift-Tab)**.

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

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

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

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

Folding in XQuery Documents

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

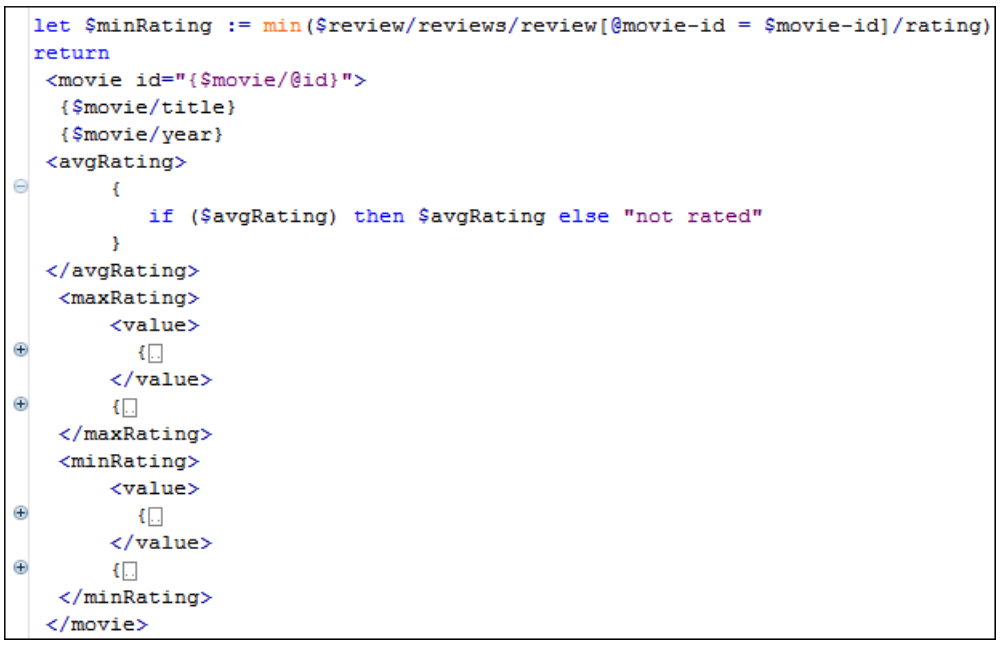


Figure 180: Folding in XQuery Documents

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

Formatting and Indenting XQuery Documents

Editing XQuery documents may lead to large chunks of content that are not easily readable by human audience. Also, each developer may have a particular way of writing XQuery code. Oxygen XML Editor plugin assists you in maintaining a consistent code writing style with the **Format and Indent** action that is available in the **Document** > **Source** menu and also on the toolbar..

The **Format and Indent** action achieves this by performing the following steps:

- Manages whitespaces, by collapsing or inserting space characters where needed.
- Formats complex expressions on multiple, more readable lines by properly indenting each of them. The amount of whitespaces that form an indent unit is controlled through one of the **Indent with tabs** and **Indent size** options from the ***Format Preferences*** *page*.

Note: These operations can be performed only if your XQuery document conforms with W3C XQuery 1.0, XQuery Update Facility 1.0, and XQuery 3.0 specifications. If the *Format and Indent* operation fails, the document is left unaltered and an error message is presented in the **Results** view.

Generating HTML Documentation for an XQuery Document

To generate HTML documentation for an XQuery document, use the **XQuery Documentation** dialog box. It is opened with the **XQuery Documentation** action that is available from the **XML Tools** > **Generate Documentation** menu or from the **Generate XQuery Documentation** action from the contextual menu of the **Navigator** view.

The dialog box allows you to configure a set of parameters for the process of generating the HTML documentation.

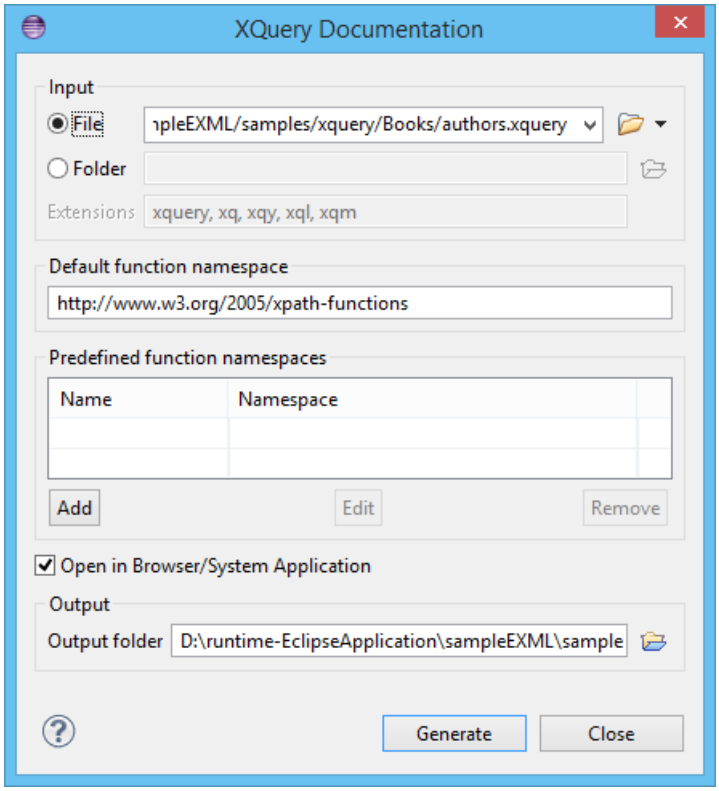


Figure 181: The XQuery Documentation Dialog Box

The following options are available:

- **Input** - The full path to the XQuery file must be specified in one of the two fields in this section:
  - **URLFile** - The URL of the file in which you want to generate the documentation.
  - **Folder** - The directory that contains the files for which you want to generate the documentation. You can also specify the XQuery file extensions to be searched for in the specified directory.
- **Default function namespace** - Optional URI for the default namespace for the submitted XQuery.
- **Predefined function namespaces** - Optional, engine-dependent, predefined namespaces that the submitted XQuery refers to. They allow the conversion to generate annotation information to support the presentation component hypertext linking (only if the predefined modules have been loaded into the local xqDoc XML repository).
- **Open in Browser/System Application** - Select this option if you want the result to be opened in the system application associated with that file type.

  Note: To set the browser or system application that will be used, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**. This will take precedence over the default system application settings.

- **Output** - Allows you to specify where the generated documentation is saved on disk.

Editing WSDL Documents

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

Oxygen XML Editor plugin provides a special type of editor dedicated to WSDL documents. The WSDL editor offers support to check whether a WSDL document is valid, a specialized Content Completion Assistant, a component oriented **Outline** view, searching and refactoring operations, and support to generate documentation.

Both WSDL version 1.1 and 2.0 are supported and SOAP versions 1.1 and 1.2. That means that in the location where a SOAP extension can be inserted the **Content Completion Assistant** offers elements from both SOAP 1.1 and SOAP 1.2. Validation of SOAP requests is executed first against a SOAP 1.1 schema and then against a SOAP 1.2 schema. In addition to validation against the XSD schemas, Oxygen XML Editor plugin also checks if the WSDL file conforms with the WSDL specification (available only for WSDL 1.1 and SOAP 1.1).

In the following example you can see how the errors are reported.

Figure 182: Validating a WSDL file

To watch our video demonstration about the WSDL editing support in Oxygen XML Editor plugin, go to *http://www.oxygenxml.com/demo/Create_New_WSDL.html*.

WSDL Outline View

The WSDL **Outline** view displays the list of all the components (services, bindings, port types and so on) of the currently open WSDL document along with the components of its imports.

If you use the ***Master Files** support*, the **Outline** view collects the components of a WSDL document starting from the master files of the current document.

To enable the **Outline** view, go to **Window** > **Show View** > **Outline**.

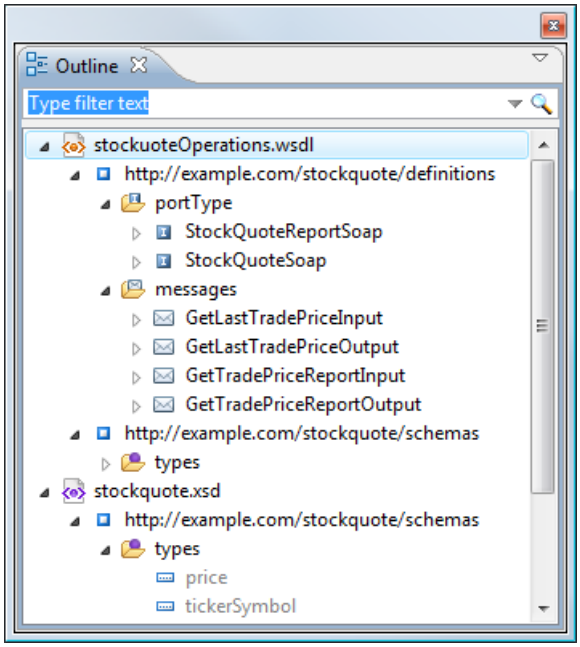


Figure 183: The WSDL Outline View

The **Outline** view can display both the components of the current document and its XML structure, organized in a tree-like fashion. You can switch between the display modes by using the **Show XML structure** and **Show components** actions in the **View menu** on the **Outline** view action bar. The following actions are available:

Filter returns exact matches

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

Selection update on cursor move

Controls the synchronization between the **Outline** view and the current document. The selection in the **Outline** view can be synchronized with the cursor moves or the changes in the WSDL editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the current document.

When the **Show components** option is selected, the following actions are available:

Show XML structure

Displays the XML structure of the current document in a tree-like manner.

Sort

Sorts the components in the **Outline** view alphabetically.

Show all components

Displays all the components that were collected starting from current document or from the main document, if it is defined.

Show referable components

Displays all the components that you can reference from the current document.

Show only local components

Displays the components defined in the current file only.

Group by location

Groups the WSDL components by their location.

Group by type

Groups the WSDL components by their type.

Group by namespace

Groups the WSDL components by their namespace.

Note: By default all the three grouping criteria are active.

When the **Show XML structure** option is selected, the following actions are available:

Show components

Switches the **Outline** view to the components display mode.

Flat presentation mode of the filtered results

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

Show comments and processing instructions

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

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

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

Configure displayed attributes

Displays the *XML Structured Outline preferences page*.

The following contextual menu actions are available in the **Outline** view when the **Show components** option is selected in the **View menu**:

Edit Attributes

Opens a dialog box that allows you to edit the attributes of the currently selected component.

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Delete

Deletes the currently selected component.

Search references

Searches for the references of the currently selected component.

Search references in

Searches for the references of the currently selected component in the context of a scope that you define.

Component dependencies

Displays the dependencies of the currently selected component.

Resource Hierarchy

Displays the hierarchy for the currently selected resource.

Resource Dependencies

Displays the dependencies of the currently selected resource.

Rename Component in

Renames the currently selected component in the context of a scope that you define.

The following contextual menu actions are available in the **Outline** view when the **Show XML structure** option is selected in the **View menu**:

Append Child

Displays a list of elements that you can insert as children of the current element.

Insert Before

Displays a list of elements that you can insert as siblings of the current element, before the current element.

Insert After

Displays a list of elements that you can insert as siblings of the current element, after the current element.

Edit Attributes

Opens a dialog box that allows you to edit the attributes of the currently selected component.

Toggle Comment

Comments/uncomments the currently selected element.

Search references

Searches for the references of the currently selected component.

Search references in

Searches for the references of the currently selected component in the context of a scope that you define.

Component dependencies

Displays the dependencies of the currently selected component.

Rename Component in

Renames the currently selected component in the context of a scope that you define.

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Delete

Deletes the currently selected component.

Expand All

Expands the structure of a component in the **Outline** view.

Collapse All

Collapses the structure of all the component in the **Outline** view.

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

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

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

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

The content of the **Outline** view and the editing area are synchronized. When you select a component in the **Outline** view, its definition is highlighted in the editing area.

Content Completion in WSDL Documents

The **Content Completion Assistant** is a powerful feature that enhances the editing of WSDL documents. It helps you define WSDL components by proposing context-sensitive element names. Another important capability of the **Content Completion Assistant** is to propose references to the defined components when you edit attribute values. For example, when you edit the `type` attribute of a `binding` element, the **Content Completion Assistant** proposes all the defined port types. Each proposal that the **Content Completion Assistant** offers is accompanied by a documentation hint.

Note: XML schema specific elements and attributes are offered when the current editing context is the internal XML schema of a WSDL document.

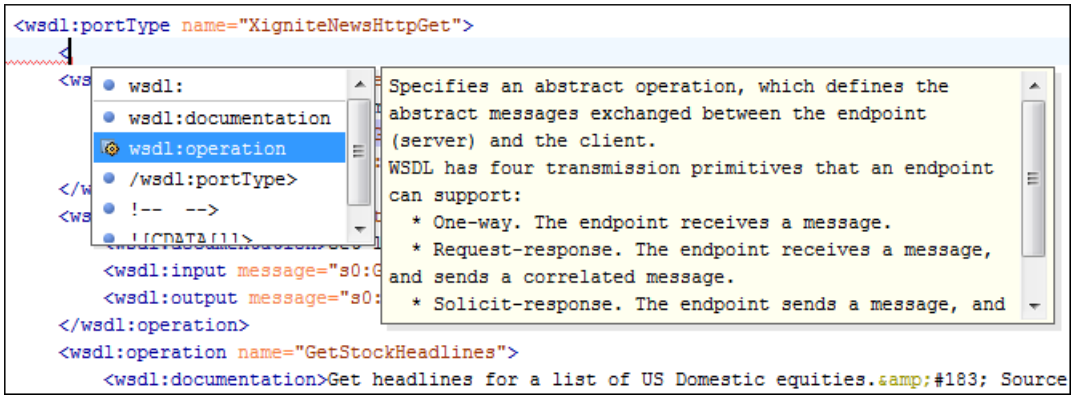


Figure 184: WSDL Content Completion Window

Note: The **Content Completion Assistant** collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to *Defining Master Files at Project Level*.

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

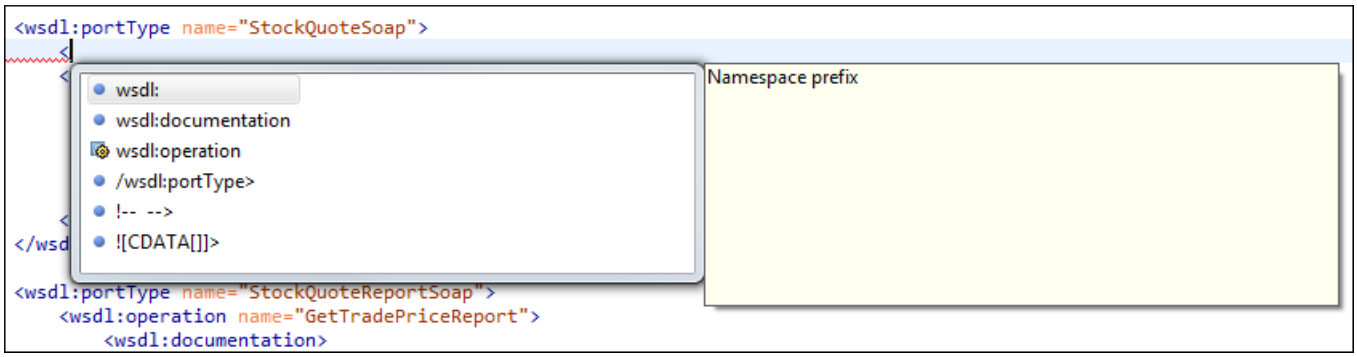


Figure 185: Namespace Prefixes in the Content Completion Window

For the common namespaces, like XML Schema namespace (*http://www.w3.org/2001/XMLSchema*) or SOAP namespace (*http://schemas.xmlsoap.org/wsdl/soap/*), Oxygen XML Editor plugin provides an easy mode to declare them by proposing a prefix for these namespaces.

Editing WSDL Documents in the Master Files Context

Smaller interrelated modules that define a complex WSDL structure cannot be correctly edited or validated individually, due to their interdependency with other modules. Oxygen XML Editor plugin provides the support for defining the main module (or modules), allowing you to edit any of the imported/included files in the context of the larger WSDL structure.

You cat set a main WSDL document either using the *master files support from the **Navigator** view*, or using a validation scenario.

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

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

- Correct validation of a module in the context of a larger WSDL structure.
- **Content Completion Assistant** displays all components valid in the current context.
- The **Outline** displays the components collected from the entire WSDL structure.

Note: When you edit an XML schema document that has a WSDL document set as master, the validation operation is performed over the master WSDL document.

To watch our video demonstration about editing WSDL documents in the master files context, go to *http://oxygenxml.com/demo/WSDL_Working_Modules.html*.

Searching and Refactoring Operations in WSDL Documents

Search Actions

The following search actions are available from the **Search** submenu in the contextual menu of the current editor:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
- **Search Occurrences in File** - Searches all occurrences of the item at the cursor position in the currently edited file.

The following action is available from the **WSDL** menu:

- **Show Definition** - Takes you to the location of the definition of the current item.

  Note: You can also use the **Ctrl Single-Click (Command Single-Click on OS X)** shortcut on a reference to display its definition.

Refactoring Actions

The following refactoring actions are available from the **Refactoring** submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the cursor position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
- **Rename Component in** - Opens the **Rename *component_type*** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

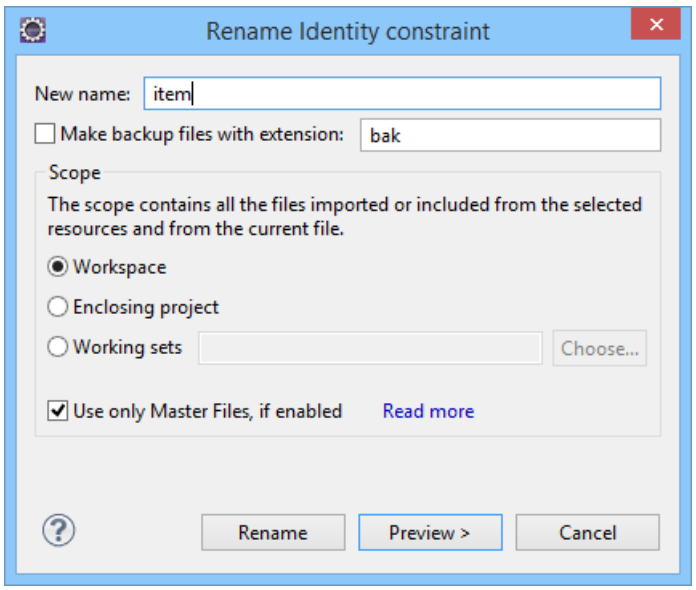


Figure 186: Rename Identity Constraint Dialog Box

Searching and Refactoring Operations Scope in WSDL Documents

The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the **Change scope** operation, available in the Quick Fix action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project or to one or multiple working sets. The **Use only Master Files, if enabled** check-box allows you to restrict the scope of the search and refactor operations to the resources from the **Master Files** directory. Click **read more** for details about the *Master Files support*.

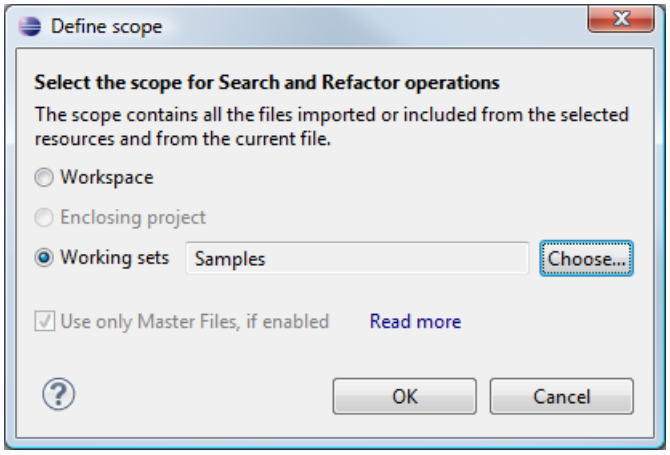


Figure 187: Change Scope Dialog Box

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.

WSDL Resource Hierarchy/Dependencies View in WSDL Documents

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a WSDL resource. To open this view, go to **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** > **Resource Hierarchy/Dependencies**.

Note: The hierarchy of a WSDL resource includes the hierarchy of imported XML Schema resources. The dependencies of an XML Schema resource present the WSDL documents that import the schema.

To view the hierarchy of a WSDL document, select the document in the project view and choose **Resource Hierarchy** from the contextual menu.

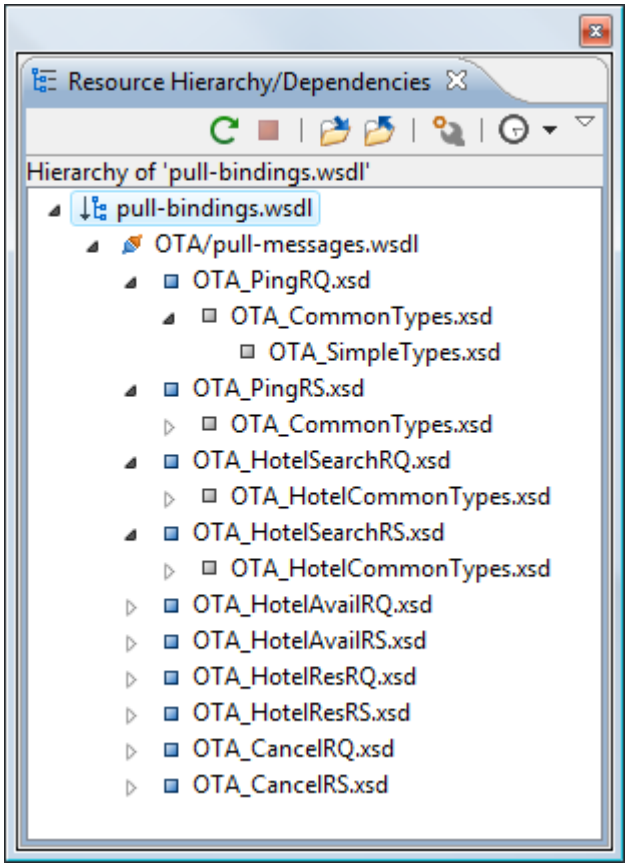


Figure 188: Resource Hierarchy/Dependencies View

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

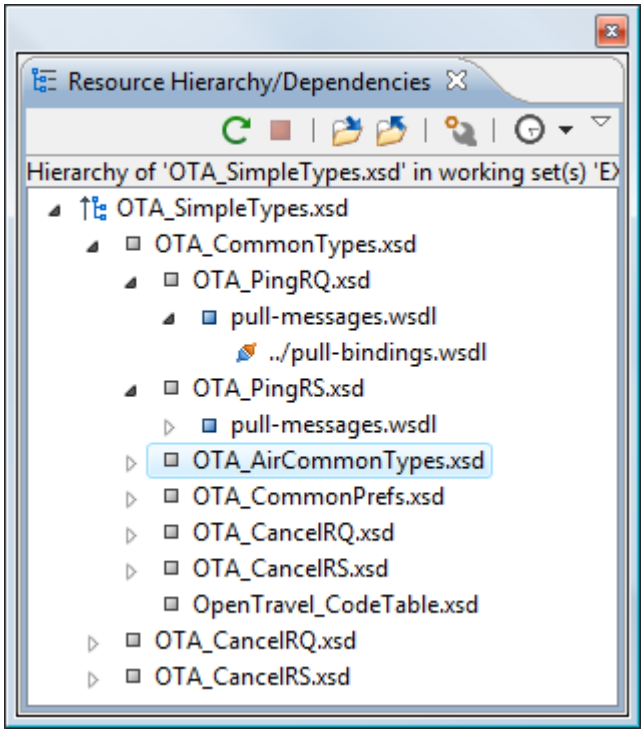


Figure 189: Resource Hierarchy/Dependencies View

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

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

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

History

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

The contextual menu contains the following actions:

Open

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

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

 **Add to Master Files**

Adds the currently selected resource in *the **Master Files** directory*.

Expand All

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

Collapse All

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

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

Note: The **Move resource** or **Rename resource** actions give you the option to *update the references to the resource*.

Moving/Renaming WSDL Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

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

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

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

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

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

Component Dependencies View in WSDL Documents

The **Component Dependencies** view allows you to view the dependencies for a selected WSDL component. To open the **Component Dependencies** view, go to **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** > **Component Dependencies**.

To view the dependencies of an WSDL component, select the desired component in the editor and choose the **Component Dependencies** action from the contextual menu. This action is available for all WSDL components (`messages`, `port types`, `operations`, `bindings` and so on).

Note: If you search for dependencies of XML Schema elements, the **Component Dependencies** view presents the references from WSDL documents.

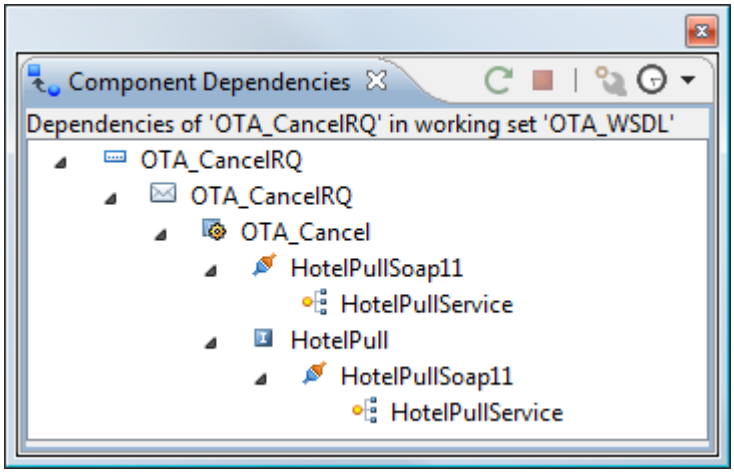


Figure 190: Component Dependencies View

The following action are available in the toolbar of the **Component Dependencies** view:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependencies computing.

Configure

Allows you to configure a *search scope* to compute the dependencies structure. You can decide to use the defined scope for future operations automatically, by checking the corresponding check box.

History

Allows you to repeat a previous dependencies computation.

The following actions are available in the contextual menu of the **Component Dependencies** view:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Displays the definition of the current selected component in the dependencies tree.

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

Highlight Component Occurrences in WSDL Documents

When you position your mouse cursor over a component in a WSDL document, Oxygen XML Editor plugin searches for the component declaration and all its references and highlights them automatically.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected.

To change the default behavior of **Highlight Component Occurrences**, *open the **Preferences** dialog box* and go to **Editor** > **Mark Occurrences**. You can also trigger a search using the **Search** > **Search Occurrences in File ()** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.

Quick Assist Support in WSDL Documents

The *Quick Assist* feature is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the line number stripe on the left side of the editor. Also, you can invoke the quick assist menu by using the **Ctrl + 1** keys (**Meta 1** on Mac OS X) keyboard shortcuts.

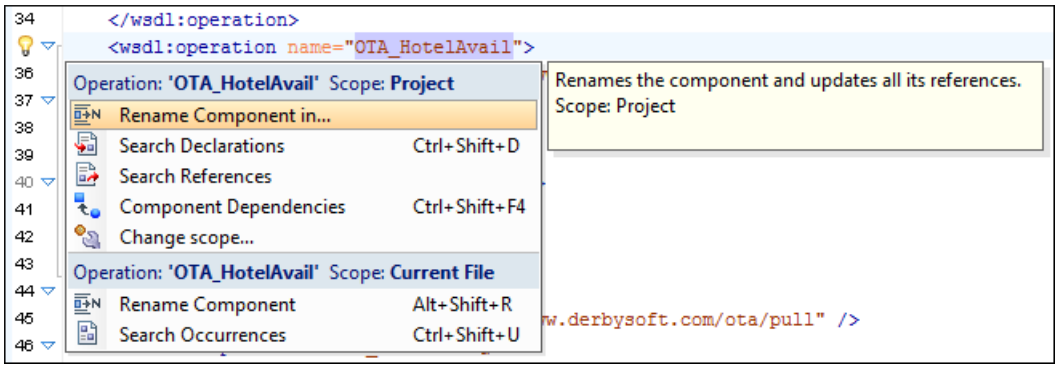


Figure 191: WSDL Quick Assist Support

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

Rename Component in

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

Generating Documentation for WSDL Documents

You can use Oxygen XML Editor plugin to generate detailed documentation for the components of a WSDL document in HTML format. You can select the WSDL components to include in your output and the level of details to present for each of them. Also, the components are hyperlinked. You can also generate the documentation in a *custom output format* by using a custom stylesheet.

Note: The WSDL documentation includes the XML Schema components that belong to the internal or imported XML schemas.

To generate documentation for a WSDL document, select **WSDL Documentation** from the **XML Tools** > **Generate Documentation** menu or from the **Generate WSDL Documentation** action from the contextual menu of the **Navigator** view.

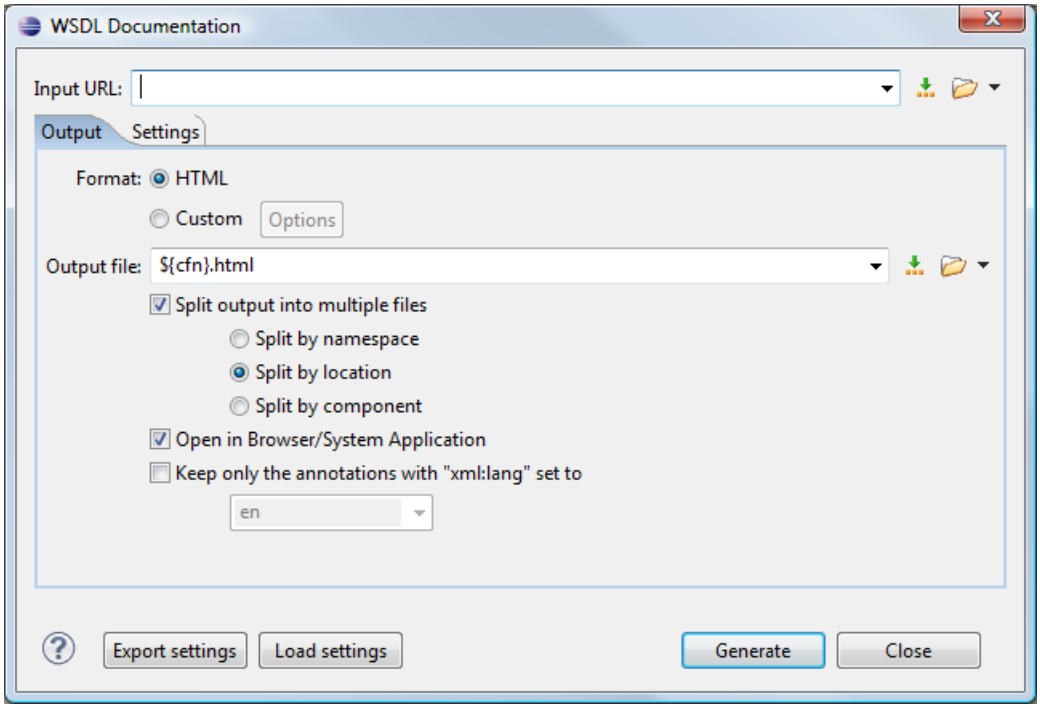


Figure 192: The WSDL Documentation Dialog Box

The **Input URL** field of the dialog box must contain the full path to the WSDL document that you want to generate documentation for. The WSDL document may be a local or a remote file. You can specify the path to the WSDL file by entering it in the text field, or by using the **Insert Editor Variables** button or the options in the **Browse** drop-down menu.

The Output Tab

The following options are available in the **Output** tab:

- **Format** - Allows you to choose between the following formats:
  - **HTML** - The documentation is generated in *HTML output format*.
  - **Custom** - The documentation is generated in a *custom output format*, allowing you to control the output. Click the **Options** button to open a **Custom format options** dialog box where you can specify a custom stylesheet for creating the output. There is also an option to **Copy additional resources to the output folder** and you can select the path to the additional **Resources** that you want to copy. You can also choose to keep the intermediate XML files created during the documentation process by deselecting the **Delete intermediate XML file** option.
- **Output file** - You can specify the path of the output file by entering it in the text field, or by using the **Insert Editor Variables** button or the options in the **Browse** drop-down menu.
- **Split output into multiple files** - Instructs the application to split the output into multiple files. For large WSDL documents, choosing a different split criterion may generate smaller output files providing a faster documentation browsing. You can choose to split them by namespace, location, or component name.
- **Open in Browser/System Application** - Opens the result in the system application associated with the output file type.

  Note: To set the browser or system application that will be used, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**. This will take precedence over the default system application settings.

- **Keep only the annotations with xml:lang set to** - The generated output will contain only the annotations with the `xml:lang` attribute set to the selected language. If you choose a primary language code (for example, **en** for English), this includes all its possible variations (**en-us**, **en-uk**, etc.).

The Setting Tab

When you generate documentation for a WSDL document, you can choose what components to include in the output and the details to be included in the documentation.

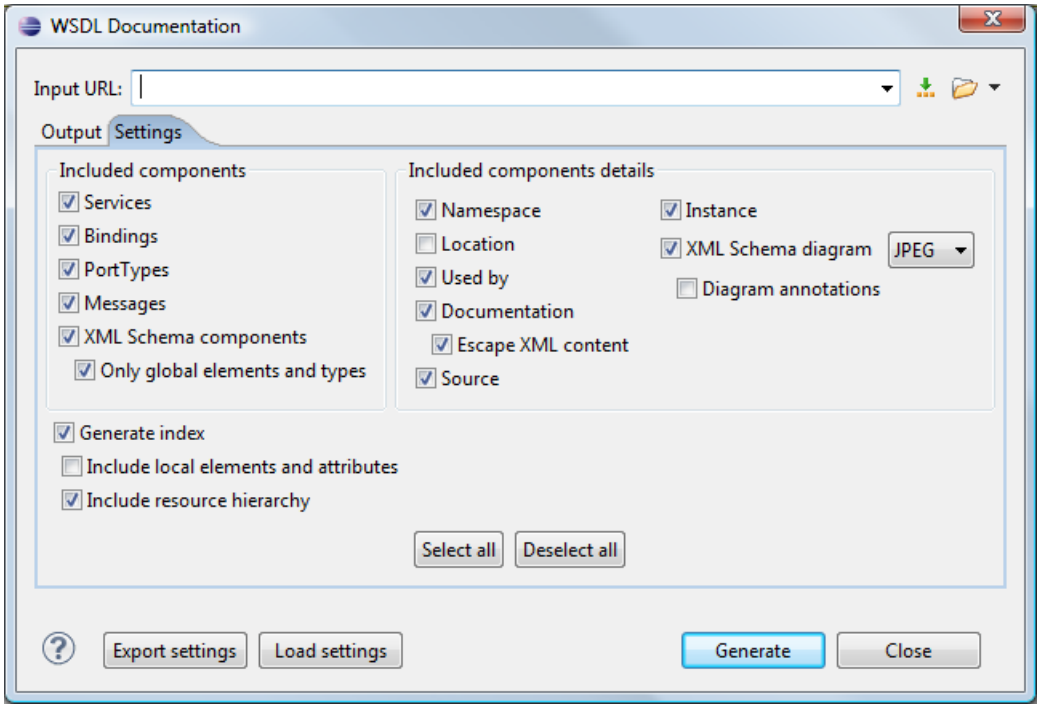


Figure 193: The Settings Tab of the WSDL Documentation Dialog Box

The **Settings** tab allows you to choose whether or not to include the following:

- **Components**
  - **Services** - Specifies whether the generated documentation includes the WSDL services.
  - **Bindings** - Specifies whether the generated documentation includes the WSDL bindings.
  - **Port Types** - Specifies whether the generated documentation includes the WSDL port types.
  - **Messages** - Specifies whether the generated documentation includes the WSDL messages.
  - **XML Schema Components** - Specifies whether the generated documentation includes the XML Schema components.
    - **Only global elements and types** - Specifies whether the generated documentation includes only global elements and types.
- **Component Details**
  - **Namespace** - Presents the namespace information for WSDL or XML Schema components.
  - **Location** - Presents the location information for each WSDL or XML Schema component.
  - **Used by** - Presents the list of components that reference the current one.
  - **Documentation** - Presents the component documentation. If you choose **Escape XML Content**, the XML tags are presented in the documentation.
  - **Source** - Presents the XML fragment that defines the current component.
  - **Instance** - Generates a sample XML instance for the current component.

    Note: This option applies to the XML Schema components only.

  - **XML Schema Diagram** - Displays the diagram for each XML Schema component. You can choose the image format (JPEG, PNG, SVG) to use for the diagram section.

    - **Diagram annotations** - Specifies whether the annotations of the components presented in the diagram sections are included.

- **Generate index** - Displays an index with the components included in the documentation.
    - **Include local elements and attributes** - If checked, local elements and attributes are included in the documentation index.
    - **Include resource hierarchy** - Specifies whether the resource hierarchy for an XML Schema documentation is generated.

Export settings - Save the current settings in a settings file for further use (for example, with the exported settings file you can generate the same *documentation from the command-line interface*.)

Load settings - Reloads the settings from the exported file.

Generate - Use this button to generate the WSDL documentation.

Generating WSDL Documentation in HTML Format

The WSDL documentation generated in HTML format is presented in a visual diagram style with various sections, hyperlinks, and options.

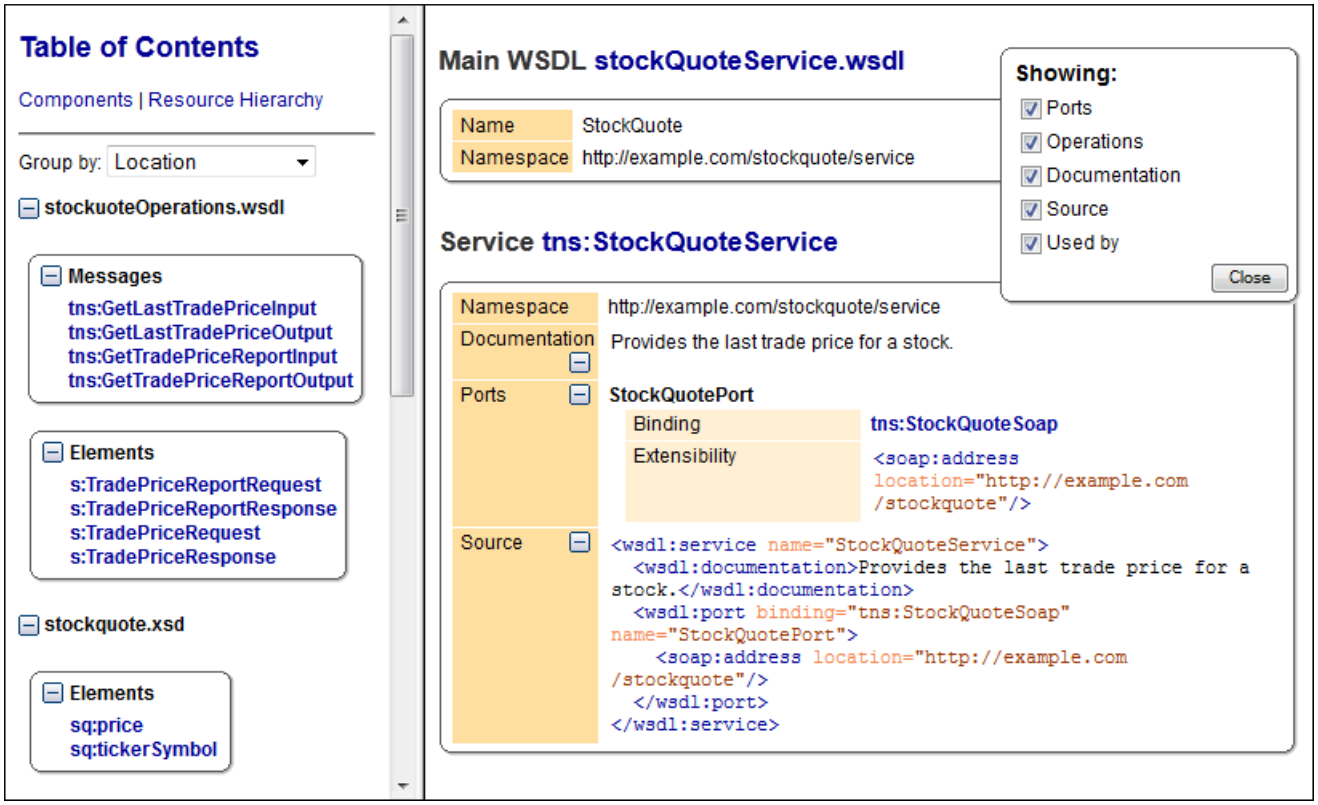


Figure 194: WSDL Documentation in HTML Format

The documentation of each component is presented in a separate section. The title of the section is composed of the component type and the component name. The component information (namespace, documentation, etc.) is presented in a tabular form.

If you choose to split the output into multiple files, the table of contents is displayed in the left frame and is divided in two tabs: **Components** and **Resource Hierarchy**.

The **Components** tab allows you to group the contents by namespace, location, or component type. The WSDL components from each group are sorted alphabetically. The **Resource Hierarchy** tab displays the dependencies between

WSDL and XML Schema modules in a tree like fashion. The root of the tree is the WSDL document that you generate documentation for.

After the documentation is generated, you can collapse or expand details for some WSDL components by using the **Showing** options or the **Collapse** or **Expand** buttons.

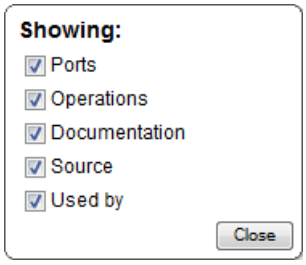


Figure 195: The Showing Options

Generating WSDL Documentation in a Custom Format

To obtain the default HTML documentation output from a WSDL document, Oxygen XML Editor plugin uses an intermediary XML document to which it applies an XSLT stylesheet. To create a custom output from your WSDL document, edit the `wsdlDocHtml.xsl` XSLT stylesheet or create your own.

Note: The `wsdlDocHtml.xsl` stylesheet that is used to obtain the HTML documentation is located in the `[OXYGEN_DIR]/frameworks/wsdl_documentation/xsl` folder.

Note: The intermediary XML document complies with the `wsdlDocSchema.xsd` XML Schema. This schema is located in the `[OXYGEN_DIR]/frameworks/wsdl_documentation` folder.

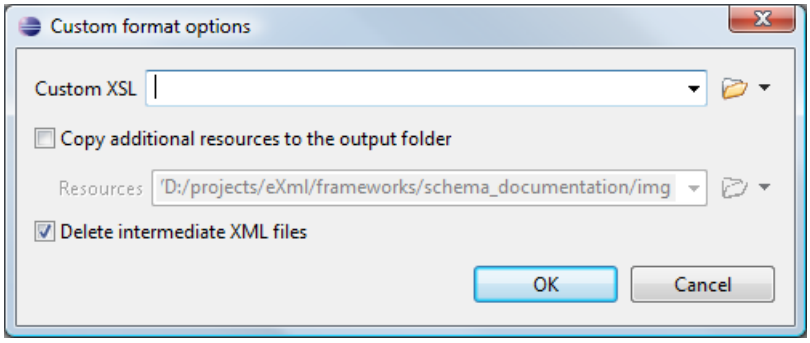


Figure 196: The Custom Format Options Dialog Box

When using a custom format, you can also copy additional resources into the output folder or choose to keep the intermediate XML files created during the documentation process.

Generating WSDL Documentation from the Command-Line Interface

To generate documentation for a WSDL document from the command line, open the **WSDL Documentation** dialog box and click **Export settings**. Using the exported settings file you can generate the same documentation from the command line by running the following scripts:

- `wsdlDocumentation.bat` on Windows.
- `wsdlDocumentation.sh` on Unix / Linux.
- `wsdlDocumentationMac.sh` on Mac OS X.

The scripts are located in the installation folder of Oxygen XML Editor plugin. You can integrate the scripts in an external batch process launched from the command-line interface.

WSDL SOAP Analyzer

After you edit and validate your Web service descriptor against a mix of the XML Schemas for WSDL and SOAP, it is easy to check if the defined SOAP messages are accepted by the remote Web Services server using Oxygen XML Editor plugin's **WSDL SOAP Analyzer** integrated tool (Available .

Composing a SOAP Request

WSDL SOAP Analyzer is a tool that helps you test if the messages defined in a Web Service Descriptor (WSDL) are accepted by a Web Services server.

Oxygen XML Editor plugin provides two ways of testing, one for the currently edited WSDL document and another for the remote WSDL documents that are published on a web server. To open the **WSDL SOAP Analyzer** tool for the currently edited WSDL document do one of the following:

- Click the **WSDL SOAP Analyzer** toolbar button.
- Use the **WSDL SOAP Analyzer** action from the **WSDL** menu.
- Go to **Open with** > **WSDL Editor** in the contextual menu of the **Navigator** view.

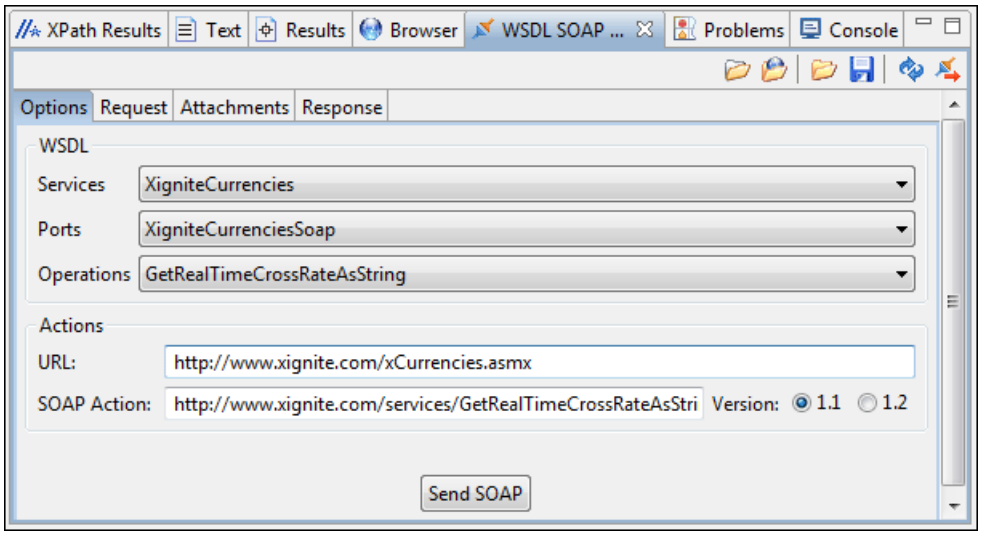


Figure 197: WSDL SOAP Analyzer View

This tool contains a SOAP analyzer and sender for Web Services Description Language file types. The analyzer fields are as follows:

- **Services** - The list of services defined by the WSDL file.
- **Ports** - The ports for the selected service.
- **Operations** - The list of available operations for the selected service.
- **Action URL** - The script that serves the operation.
- **SOAP Action** - Identifies the action performed by the script.
- **Version** - Choose between 1.1 and 1.2. The SOAP version is selected automatically depending on the selected port.
- **Request Editor** - It allows you to compose the web service request. When an action is selected, Oxygen XML Editor plugin tries to generate as much content as possible for the SOAP request. The envelope of the SOAP request has the correct namespace for the selected SOAP version, that is *http://schemas.xmlsoap.org/soap/envelope/* for SOAP 1.1 or *http://www.w3.org/2003/05/soap-envelope* for SOAP 1.2. Usually you just have to change few values in order for the request to be valid. The content completion assistant is available for this editor and is driven by the schema that defines the type of the current message. While selecting different operations, Oxygen XML Editor plugin remembers the modified request for each one. You can press the **Regenerate** button in order to overwrite your modifications for the current request with the initial generated content.
- **Attachments List** - You can define a list of file URLs to be attached to the request.

- **Response Area** - Initially it displays an auto generated server sample response so you can have an idea about how the response looks like. After pressing the **Send** button, it presents the message received from the server in response to the Web Service request. It may show also error messages. If the response message contains attachments, Oxygen XML Editor plugin prompts you to save them, then tries to open them with the associated system application.
- **Errors List** - There may be situations in which the WSDL file is respecting the WSDL XML Schema, but it fails to be valid for example in the case of a message that is defined by means of an element that is not found in the types section of the WSDL. In such a case, the errors are listed here. This list is presented only when there are errors.
- **Send Button** - Executes the request. A status dialog box is displayed when Oxygen XML Editor plugin is connecting to the server.

The testing of a WSDL file is straight-forward: click the WSDL analysis button, then select the service, the port, and the operation. The editor generates the skeleton for the SOAP request. You can edit the request, eventually attach files to it and send it to the server. Watch the server response in the response area. You can find more details in the *Testing Remote WSDL Files* section.

Note: SOAP requests and responses are automatically validated in the **WSDL SOAP Analyzer** using the XML Schemas specified in the WSDL file.

Once defined, a request derived from a Web Service descriptor can be saved with the **Save** button to a Web Service SOAP Call (WSSC) file for later reuse. In this way, you save time in configuring the URLs and parameters.

You can open the result of a Web Service call in an editor panel using the **Open** button.

Testing Remote WSDL Files

To open and test a remote WSDL file the steps are the following:

1. Go to **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** > **WSDL SOAP Analyzer**.
2. Press the **Choose WSDL** button and enter the URL of the remote WSDL file.
   You enter the URL:
   - by typing
   - by browsing the local file system
   - by browsing a remote file system
   - by browsing *a UDDI Registry*
3. Press the **OK** button.
   This will open the **WSDL SOAP Analyzer** tool. In the **Saved SOAP Request** tab you can open directly a previously saved Web Service SOAP Call (WSSC) file thus skipping the analysis phase.

The UDDI Registry Browser

Pressing the button in the **WSDL File Opener** dialog box (menu **Tools** > **WSDL SOAP Analyzer**) opens the **UDDI Registry Browser** dialog box.

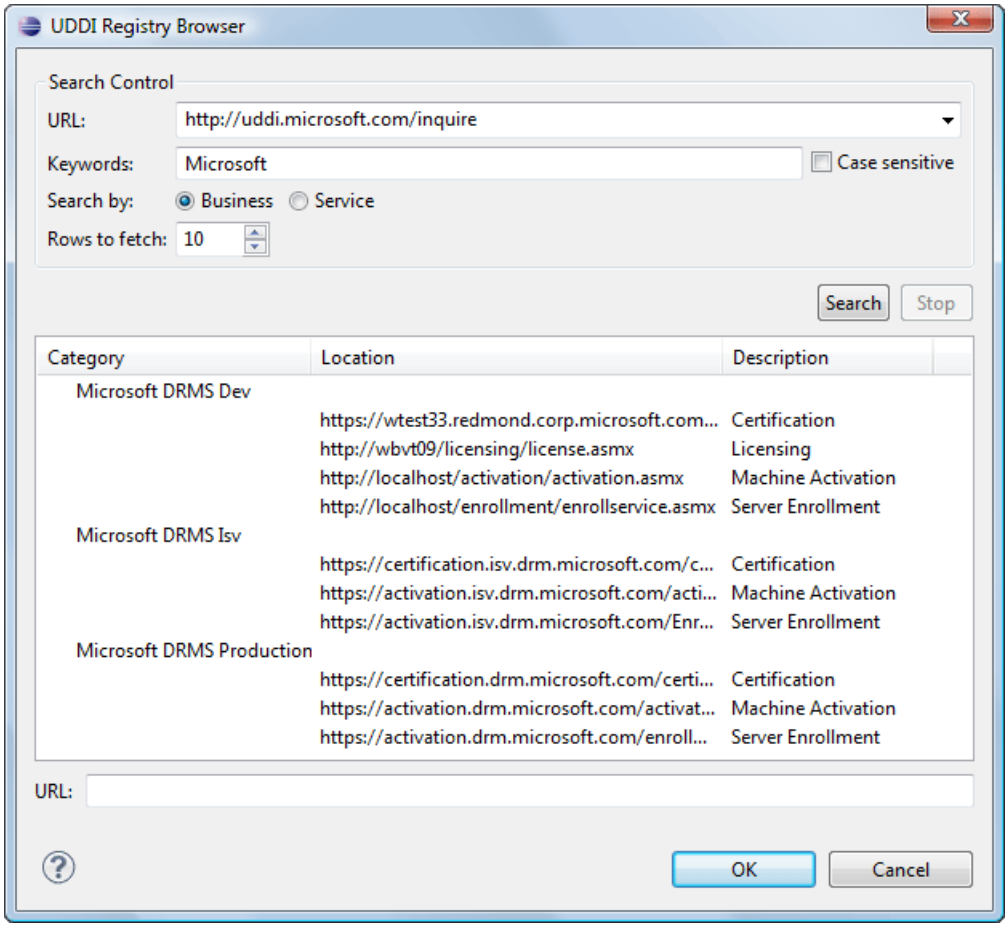


Figure 198: UDDI Registry Browser Dialog Box

The fields of the dialog box are as follows:

- **URL** - Type the URL of an UDDI registry or choose one from the default list.
- **Keywords** - Enter the string you want to be used when searching the selected UDDI registry for available Web services.
- **Rows to fetch** - The maximum number of rows to be displayed in the result list.
- **Search by** - You can choose to search either by company or by provided service.
- **Case sensitive** - When checked, the search takes into account the keyword case.
- **Search** - The WSDL files that matched the search criteria are added in the result list.

When you select a WSDL from the list and click the **OK** button, the **UDDI Registry Browser** dialog box is closed and you are returned to the **WSDL File Opener** dialog box.

Editing CSS Stylesheets

This section explains the features of the editor for CSS stylesheets and how these features should be used.

Validating CSS Stylesheets

Oxygen XML Editor plugin includes a built-in *CSS Validator*, integrated with general validation support. This makes the *usual validation features* for presenting errors also available for CSS stylesheets.

The CSS properties accepted by the validator are those included in the current CSS profile that is selected in *the CSS validation preferences*. The **CSS 3 with Oxygen extensions** profile includes all the CSS 3 standard properties plus 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

Lists the steps required for specifying custom CSS properties.

To specify custom CSS properties, follow these steps:

1. Create a file named `CustomProperties.xml` that has the following structure:

```
<?xml 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 anywhere on your desktop. From the contextual menu, select **Open Command Window Here**.
4. In the command line, run the `jar cvf custom_props.jar builtin/` command.
   The `custom_props.jar` file is created.
5. Go to `[OXYGEN_DIR]/lib` and create the `endorsed` folder. Copy the `custom_props.jar` file to `[OXYGEN_DIR]/lib/endorsed`.

Content Completion in CSS Stylesheets

A **Content Completion Assistant**, similar to *the one available for XML documents* offers the CSS properties and the values available for each property. It is activated with the **Ctrl Space (Command Space on OS X)** shortcut and is context-sensitive when invoked for the value of a property. The **Content Completion Assistant** also includes *code templates that can be used to quickly insert code fragments* into CSS stylesheets. The code templates that are proposed include form controls, actions, and **Author** mode operations.

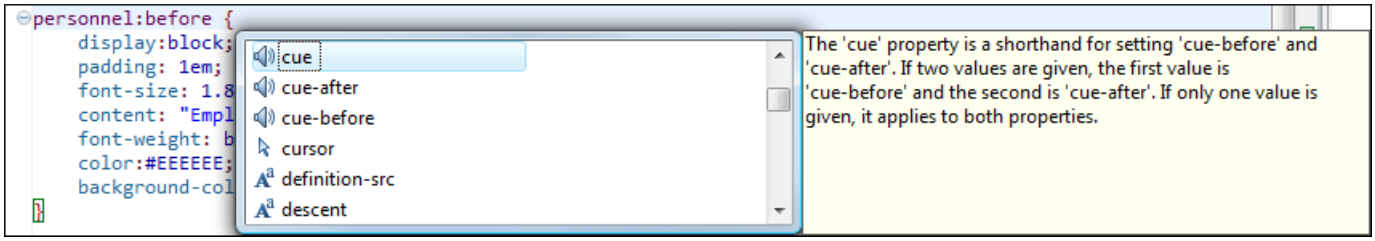


Figure 199: Content Completion in CSS Stylesheets

The properties and values available are dependent on the CSS Profile selected in the *CSS preferences* . The CSS 2.1 set of properties and property values is used for most of the profiles. However, with CSS 1 and CSS 3 specific proposal sets are used.

The profile **CSS 3 with Oxygen extensions** includes all the CSS 3 standard properties plus the *CSS extensions specific for Oxygen XML Editor plugin* 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 the element name used in the selector.

- Sorted by the entire selector string representation.

You can synchronize the selection in the **Outline** view with the cursor moves or changes you make in the stylesheet document. When you select an entry from the **Outline** view, Oxygen XML Editor plugin highlights the corresponding import or selector in the CSS editor.

To enable the **Outline** view, go to **Window** > **Show View** > **Outline**.

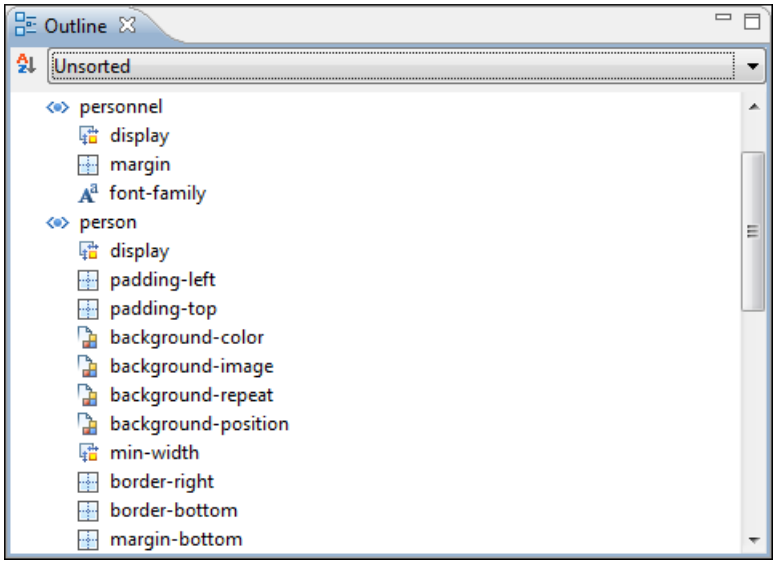


Figure 200: CSS Outline View

The selectors presented in this view can be found quickly using the key search field. When you press a sequence of character keys while the focus is in the view, the first selector that starts with that sequence is selected automatically.

Folding in CSS Stylesheets

In a large CSS stylesheet document, some styles can be collapsed so that only the styles that are needed remain in focus. The same *folding features available for XML documents* are also available in CSS stylesheets.

Note: To enhance your editing experience, you can select entire blocks (parts of text delimited by brackets) by double-clicking somewhere inside the brackets.

Formatting and Indenting CSS Stylesheets (Pretty Print)

If the edited CSS stylesheet becomes unreadable because of the bad alignment of the text lines, the *format and indent operation available for XML documents* is also available for CSS stylesheets. It works in the same way as for XML documents and is available as the same menu and toolbar action.

Minifying CSS Stylesheets

Minification (or *compression*) of a CSS document is the practice of removing unnecessary code without affecting the functionality of the stylesheet.

To minify a CSS, invoke the contextual menu anywhere in the edited document and choose the **Minify CSS** action. Oxygen XML Editor plugin opens a dialog box that allows you to:

- Set the location of the resulting CSS.
- Place each style rule on a new line.

After pressing **OK**, Oxygen XML Editor plugin performs the following actions:

- All spaces are normalized (all leading and trailing spaces are removed, while sequences of white spaces are replaced with single space characters).

- All comments are removed.

Note: The CSS minifier relies heavily upon the W3C CSS specification. If the content of the CSS file you are trying to minify does not conform with the specifications, an error dialog box will be displayed, listing all errors encountered during the processing.

The resulting CSS stylesheet gains a lot in terms of execution performance, but loses in terms of readability. The source CSS document is left unaffected.

Note: To restore the readability of a minified CSS, invoke the **Format and Indent** action from the **XML** menu, the **Source** submenu from the contextual menu, or **Source** toolbar. However, this action will not recover any of the deleted comments.

Editing LESS CSS Stylesheets

Oxygen XML Editor plugin provides support for stylesheets coded with the LESS dynamic stylesheet language. LESS extends the CSS language by adding features that allow mechanisms such as *variables*, *nesting*, *mixins*, *operators*, and *functions*. Oxygen XML Editor plugin offers additional LESS features that include:

- Open LESS files - the LESS extension is recognized and thus can be opened by the editor
- Validation - presents errors in LESS files
- Content completion - offers properties and the values available for each property
- Compile to CSS - options are available to compile LESS files to CSS

Note: Oxygen XML Editor plugin also support syntax highlighting in LESS files, although there may be some limitations in supporting all the LESS constructs.

For more information about LESS go to *http://lesscss.org/*.

Validating LESS Stylesheets

Oxygen XML Editor plugin includes a built-in *LESS CSS Validator*, integrated with general validation support. The *usual validation features* for presenting errors also available for LESS stylesheets.

Oxygen XML Editor plugin provides three validation methods:

- Automatic validation as you type - marks validation errors in the document as you are editing.
- Validation upon request, by pressing the **Validate** button from the **Validation** toolbar drop-down menu. An error list is presented in the message panel at the bottom of the editor.
- Validation scenarios, by selecting **Configure Validation Scenario(s)** from the **Validation** toolbar drop-down menu. Errors are presented in the message panel at the bottom of the editor. This is useful when you need to validate the current file as part of a larger LESS import hierarchy (for instance, you may change the URL of the file to validate to the root of the hierarchy).

Content Completion in LESS Stylesheets

A **Content Completion Assistant** offers the LESS properties and the values available for each property. It is activated with the **Ctrl Space (Command Space on OS X)** shortcut and is context-sensitive when invoked for the value of a property in a LESS file. The **Content Completion Assistant** also includes *code templates that can be used to quickly insert code fragments* into LESS stylesheets. The code templates that are proposed include form controls, actions, and **Author** mode operations.

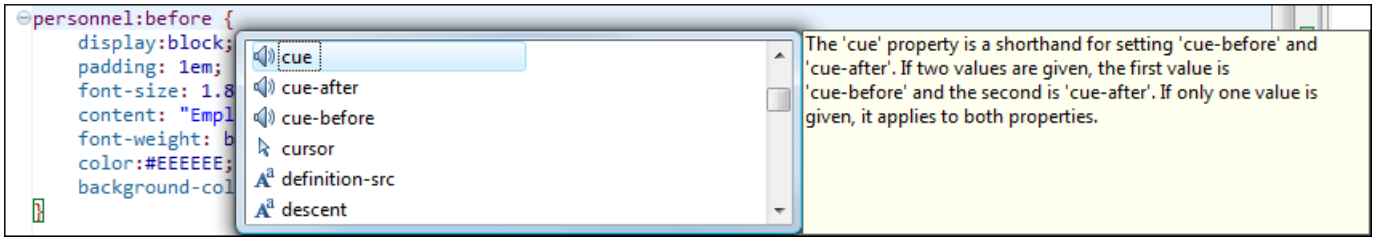


Figure 201: Content Completion in LESS Stylesheets

The properties and values available are dependent on the CSS Profile selected in the *CSS preferences* .

Compiling LESS Stylesheets to CSS

When editing LESS files, you can compile the files into CSS. Oxygen XML Editor plugin provides both manual and automatic options to compile LESS stylesheets into CSS.

Important: The LESS processor works well only with files having the *UTF-8* encoding. Thus, it is highly recommended that you always use the `utf-8` encoding when working with LESS files or the files they import (other LESS or CSS files). You can use the following directive at the beginning of your files:

```
@charset "utf-8";
```

You have two options for compiling LESS files to CSS:

1. Use the contextual menu in a LESS file and select **Compile to CSS (Ctrl Shift C (Meta Shift C on OS X))**.
2. Enable the option **Automatically compile LESS to CSS when saving** in the settings. To do so, *open the **Preferences** dialog box* and go to **Editor** > **Open** > **Save** > **Save hooks**. If enabled, when you save a LESS file it will automatically be compiled to CSS (this option is disabled by default).

   Important: If this option is enabled, when you save a LESS file, the CSS file that has the same name as the LESS file is overwritten without warning. Make sure all your changes are made in the LESS file. Do not edit the CSS file directly, as your changes might be lost.

Editing Relax NG Schemas

Oxygen XML Editor plugin provides a special type of editor for Relax NG schemas. This editor presents the usual text view of an XML document synchronized in real time with an outline view. The outline view has two display modes: the *standard outline* mode and the *components* mode.

Editing Relax NG Schema in the Master Files Context

Smaller interrelated modules that define a complex Relax NG Schema cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, an element defined in a main schema document is not visible when you edit an included module. Oxygen XML Editor plugin provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

You cat set a main Relax NG document either using the *master files support from the **Navigator** view*, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main schemas. Oxygen XML Editor plugin warns you if the current module is not part of the dependencies graph computed for the main schema. In this case, it considers the current module as the main schema.

The advantages of editing in the context of main file include:

- Correct validation of a module in the context of a larger schema structure.

- **Content Completion Assistant** displays all the referable components valid in the current context. This include components defined in modules other than the currently edited one.
- The **Outline** displays the components collected from the entire schema structure;

Relax NG Schema Diagram

This section explains how to use the graphical diagram of a Relax NG schema.

Introduction to Relax NG Schema Diagram View

Oxygen XML Editor plugin provides a simple, expressive, and easy to read **Schema Diagram** view for Relax NG schemas.

With this new feature you can easily develop complex schemas, print them on multiple pages or save them as JPEG, PNG, or BMP images. It helps both schema authors in developing the schema and content authors who are using the schema to understand it.

Oxygen XML Editor plugin is the only XML editor to provide a side by side source and diagram presentation and have them real-time synchronized:

- The changes you make in the Editor are immediately visible in the Diagram (no background parsing).
- Changing the selected element in the diagram selects the underlying code in the source editor.

Full Model View

When you create a new schema document or open an existing one, the editor panel is divided in two sections: one containing the schema diagram and the second the source code. The **Diagram** view has two tabbed panes offering a **Full Model View** and a **Logical Model View**.

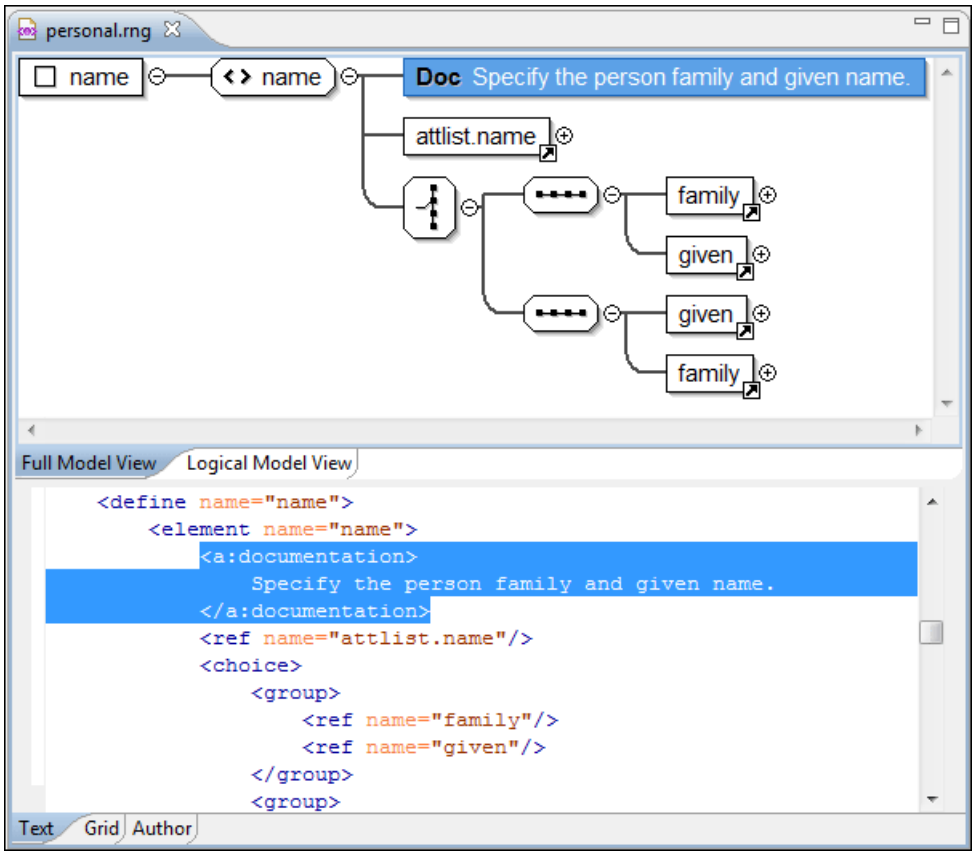


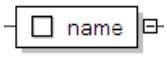
Figure 202: Relax NG Schema Editor - Full Model View

The following references can be expanded in place: patterns, includes, and external references. This expansion mechanism, coupled with the synchronization support, makes the schema navigation easy.

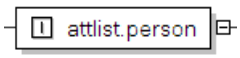
All the element and attribute names are editable: double-click any name to start editing it.

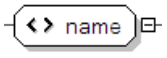
Symbols Used in the Schema Diagram

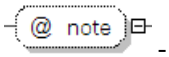
The **Full Model View** renders all the Relax NG Schema patterns with intuitive symbols:

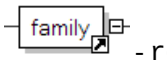
- 

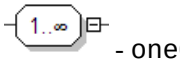
- `define` pattern with the `name` attribute set to the value shown inside the rectangle (in this example `name`).
- 

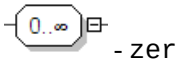
- `define` pattern with the `combine` attribute set to `interleave` and the `name` attribute set to the value shown inside the rectangle (in this example `attlist.person`).
- 

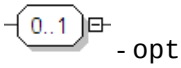
- `define` pattern with the `combine` attribute set to `choice` and the `name` attribute set to the value shown inside the rectangle (in this example `attlist.person`).
- 

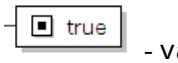
- `element` pattern with the `name` attribute set to the value shown inside the rectangle (in this example `name`).
- 

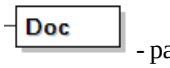
- `attribute` pattern with the `name` attribute set to the value shown inside de rectangle (in this case `note`).
- 

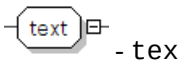
- `ref` pattern with the `name` attribute set to the value shown inside the rectangle (in this case `family`).
- 

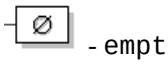
- `oneOrMore` pattern.
- 

- `zeroOrMore` pattern.
- 

- `optional` pattern.
- 
- `choice` pattern.
- 

- `value` pattern, used for example inside a `choice` pattern.
- 
- `group` pattern.
- 

- pattern from the Relax NG Annotations namespace (http://relaxng.org/ns/compatibility/annotations/1.0) which is treated as a documentation element in a Relax NG schema.
- 

- `text` pattern.
- 
- `empty` pattern.

Logical Model View

The **Logical Model View** presents the compiled schema which is a single pattern. The patterns that form the element content are defined as top level patterns with generated names. These names are generated depending of the elements name class.

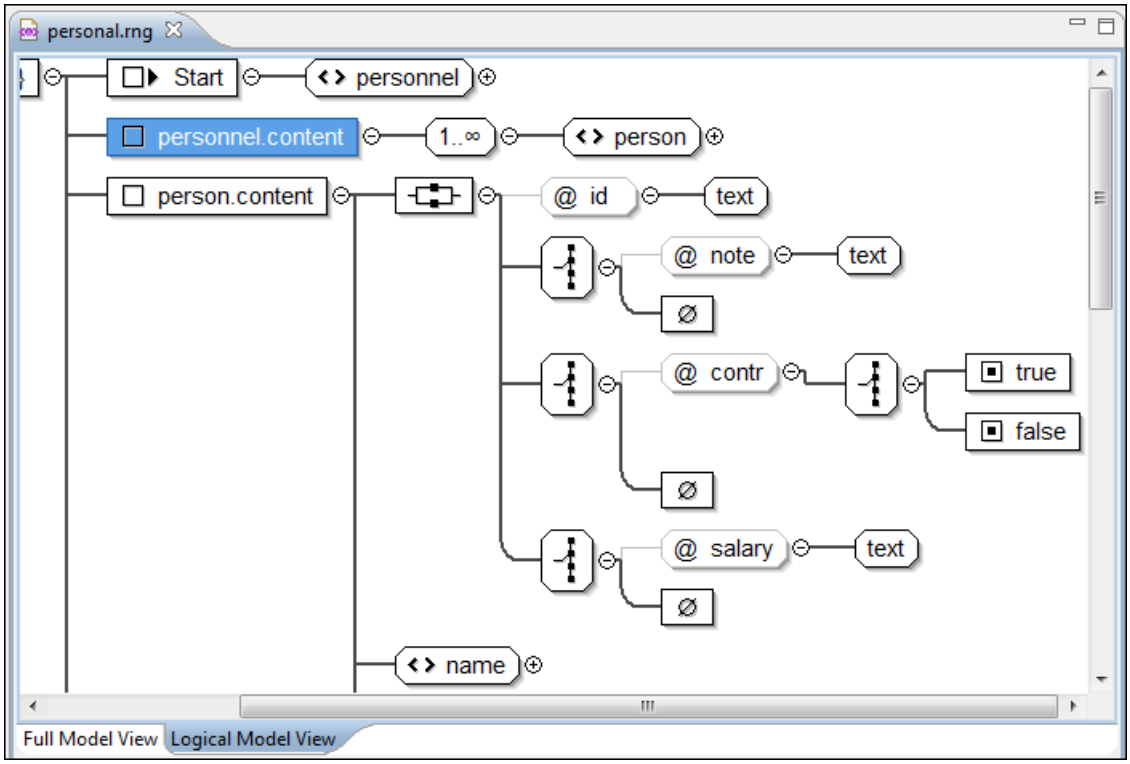


Figure 203: Logical Model View for a Relax NG Schema

Actions Available in the Schema Diagram View

When editing Relax NG schemas in **Full Model View**, the contextual menu offers the following actions:

Append child

Appends a child to the selected component.

Insert Before

Inserts a component before the selected component.

Insert After

Inserts a component after the selected component.

Edit attributes

Edits the attributes of the selected component.

Remove

Removes the selected component.

Show only the selected component

Depending on its state (selected/not selected), either the selected component or all the diagram components are shown.

Show Annotations

Depending on its state (selected/not selected), the documentation nodes are shown or hidden.

Auto expand to references

This option controls how the schema diagram is automatically expanded. If you select it and then edit a top-level element or you make a refresh, the diagram is expanded until it reaches referenced components. If this option is left unchecked, only the first level of the diagram is expanded, showing the top-level elements. For large schemas, the editor disables this option automatically.

Collapse Children

Collapses the children of the selected view.

Expand Children

Expands the children of the selected view.

Print Selection

Prints the selected view.

Save as Image

Saves the current selection as JPEG, BMP, SVG or PNG image.

Refresh

Refreshes the schema diagram according to the changes in your code. They represent changes in your imported documents or changes that are not reflected automatically in the compiled schema).

If the schema is not valid, you see only an error message in the **Logical Model View** instead of the diagram.

Relax NG Outline View

The Relax NG **Outline** view presents a list with the patterns that appear in the diagram in both the **Full Model View** and **Logical Model View** cases. It allows a quick access to a component by name. By default it is displayed on screen, but if you closed it you can reopen it from **Window** > **Show View** > **Outline**.

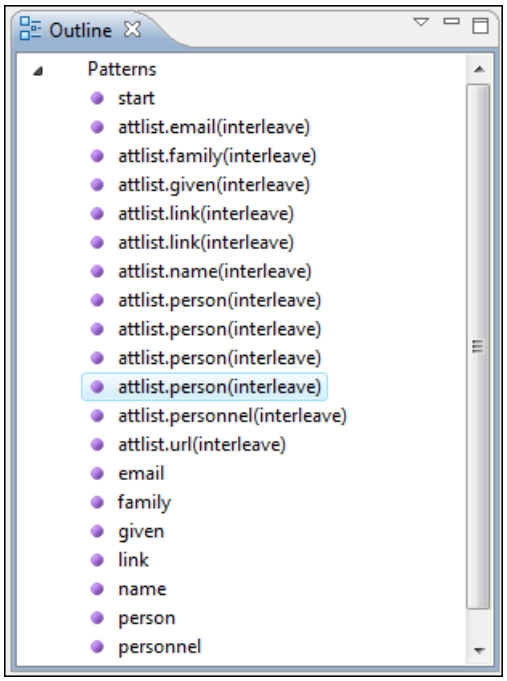


Figure 204: Relax NG Outline View

The tree shows the XML structure or the defined pattern (components) collected from the current document. By default, the **Outline** view presents the components.

When the **Show components** option is selected in the **View menu** on the **Outline** view action bar, the following option is available:

Show XML structure

Shows the XML structure of the current document in a tree-like manner.

The following actions are available in the **View menu** on the **Outline** view action bar when the **Show XML structure** option is selected:

Filter returns exact matches

The text filter of the **Outline** view returns only exact matches.

Selection update on cursor move

Allows a synchronization between **Outline** view and schema diagram. The selected view from the diagram will be also selected in the **Outline** view.

Show components

Shows the defined pattern collected from the current document.

Flat presentation mode of the filtered results

When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions

Show/hide comments and processing instructions in the **Outline** view.

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from *the Outline preferences panel*.

Configure displayed attributes

Displays the *XML Structured Outline preferences page*.

The following contextual menu actions are also available in the **Outline** view when the **Show XML structure** option is selected in the **View menu**:

Append Child

Displays a list of elements that you can insert as children of the current element.

Insert Before

Displays a list of elements that you can insert as siblings of the current element, before the current element.

Insert After

Displays a list of elements that you can insert as siblings of the current element, after the current element.

Edit Attributes

Opens a dialog box that allows you to edit the attributes of the currently selected component.

Toggle Comment

Comments/uncomments the currently selected element.

Search references

Searches for the references of the currently selected component.

Search references in

Searches for the references of the currently selected component in the context of a scope that you define.

Component dependencies

Displays the dependencies of the currently selected component.

Rename Component in

Renames the currently selected component in the context of a scope that you define.

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Delete

Deletes the currently selected component.

Expand All

Expands the structure of a component in the **Outline** view.

Collapse All

Collapses the structure of all the component in the **Outline** view.

The upper part of the **Outline** view contains a filter box that 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.

Searching and Refactoring Actions in RNG Schemas

Search Actions

The following search actions can be applied on named *defines* and are available from the **Search** submenu in the contextual menu of the current editor:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
- **Search Occurrences in File** - Searches all occurrences of the item at the cursor position in the currently edited file.

The following action is available from the **XSL** menu:

- **Show Definition** - Moves the cursor to the definition of the current element in the Relax NG (full syntax) schema.

  Note: You can also use the **Ctrl Single-Click (Command Single-Click on OS X)** shortcut on a reference to display its definition.

Refactoring Actions

The following refactoring actions can be applied on named *defines* and are available from the **Refactoring** submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the cursor position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
- **Rename Component in** - Opens the **Rename *component_type*** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

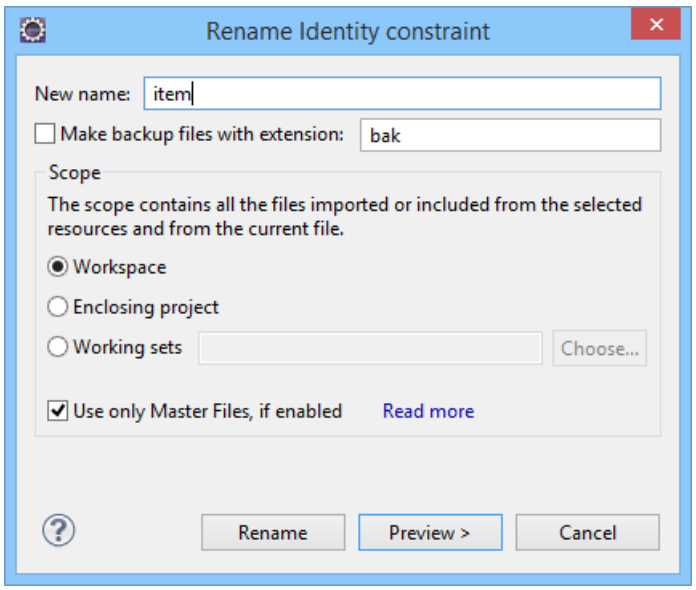


Figure 205: Rename Identity Constraint Dialog Box

RNG Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a schema. To open this view, go to **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** > **Resource Hierarchy/Dependencies**.

If you want to see the hierarchy of a schema, select the desired schema in the project view and choose **Resource Hierarchy** from the contextual menu.

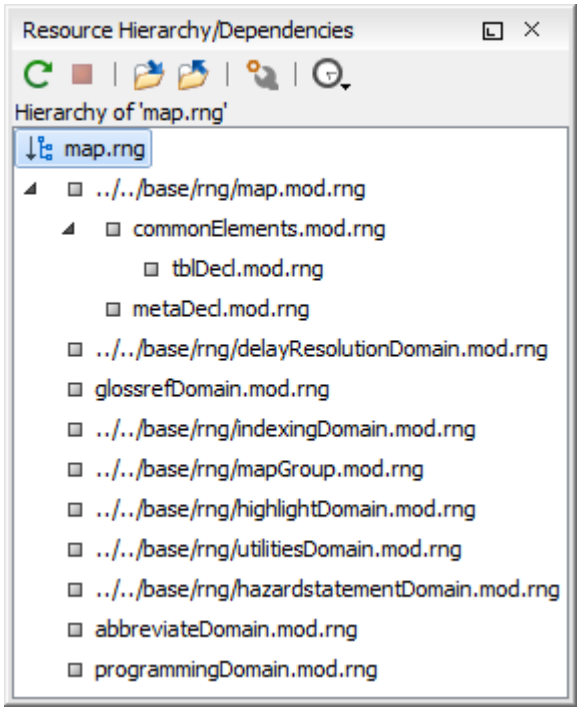


Figure 206: Resource Hierarchy/Dependencies View - hierarchy for `map.rng`

If you want to see the dependencies of a schema, select the desired schema in the project view and choose **Resource Dependencies** from the contextual menu.

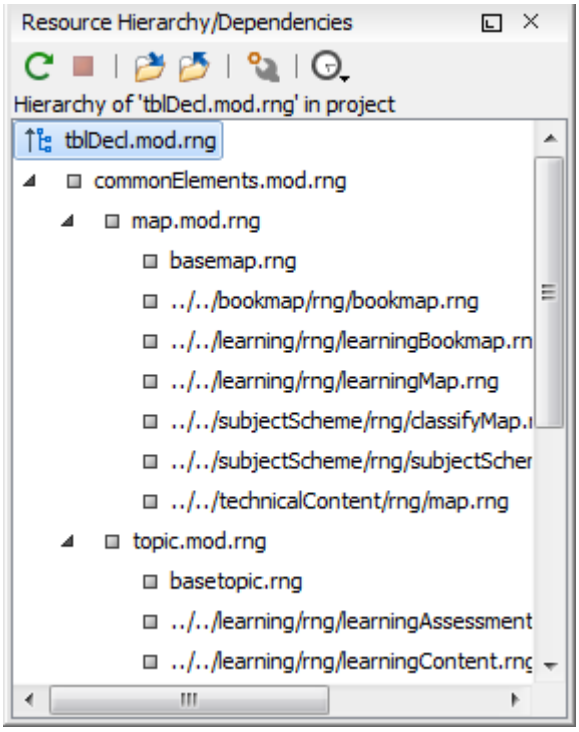


Figure 207: Resource Hierarchy/Dependencies View - Dependencies for `tblDecl.mod.rng`

The following actions are available in the **Resource Hierarchy/Dependencies** view:

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History

Provides access to the list of previously computed dependencies. Use the **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

 **Add to Master Files**

Adds the currently selected resource in *the **Master Files** directory*.

Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

Tip: When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon  .

Note: The **Move resource** or **Rename resource** actions give you the option to *update the references to the resource*.

Moving/Renaming RNG Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

When you select the **Rename** action in the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Rename resource** dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the **Move** action from the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Move resource** dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

If the **Update references of the moved resource(s)** option is enabled, a **Preview** option (which opens the **Preview** dialog box) is available for both actions. The **Preview** dialog box presents a list with the resources that are updated.

Note: Updating the references of a resource that is resolved through a catalog is not supported. Also, the update references operation is not supported if the path to the renamed or moved resource contains entities.

Component Dependencies View for RelaxNG Schemas

The **Component Dependencies** view allows you to see the dependencies for a selected Relax NG component. You can open the view from **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** > **Component Dependencies**.

If you want to see the dependencies of a RelaxNG component, select the desired component in the editor and choose the **Component Dependencies** action from the contextual menu. The action is available for all named defines.

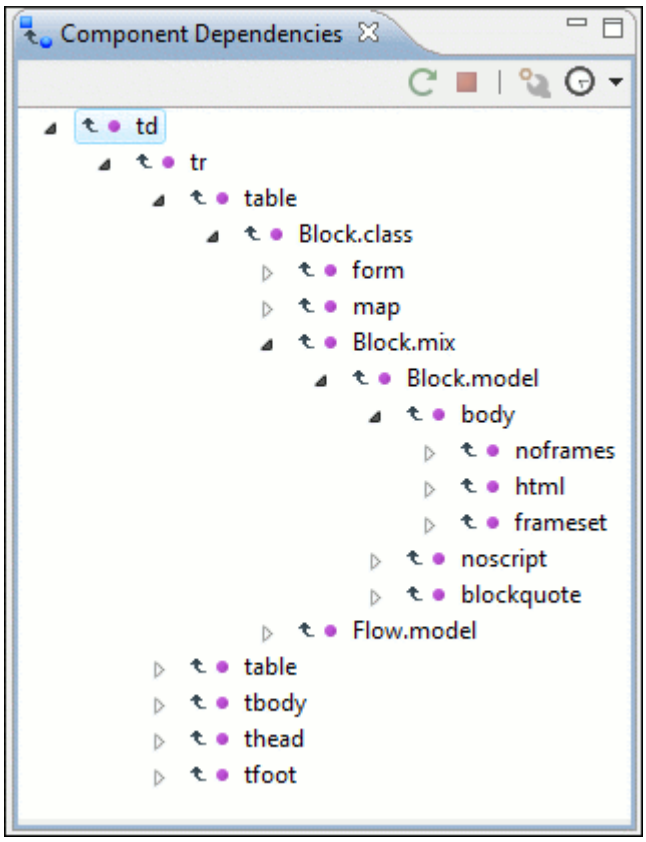


Figure 208: Component Dependencies View - Hierarchy for `xhtml.rng`

In the Component Dependencies view you have several actions in the toolbar:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependencies computing.

Configure

Allows you to configure a search scope to compute the dependencies structure. You can decide to use automatically the defined scope for future operations by checking the corresponding checkbox.

History

Allows you to repeat a previous dependencies computation.

The following actions are available on the contextual menu:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Shows the definition of the current selected component in the dependencies tree.

Tip: If a component contains multiple references to another components, a small table is displayed that contains all references. When a recursive reference is encountered, it is marked with a special icon .

RNG Quick Assist Support

The *Quick Assist* support improves the development work flow, offering fast access to the most commonly used actions when you edit XML Schema documents.

The *Quick Assist* feature is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the line number stripe on the left side of the editor. Also, you can invoke the quick assist menu by using the **Ctrl + 1** keys (**Meta 1** on Mac OS X) keyboard shortcuts.

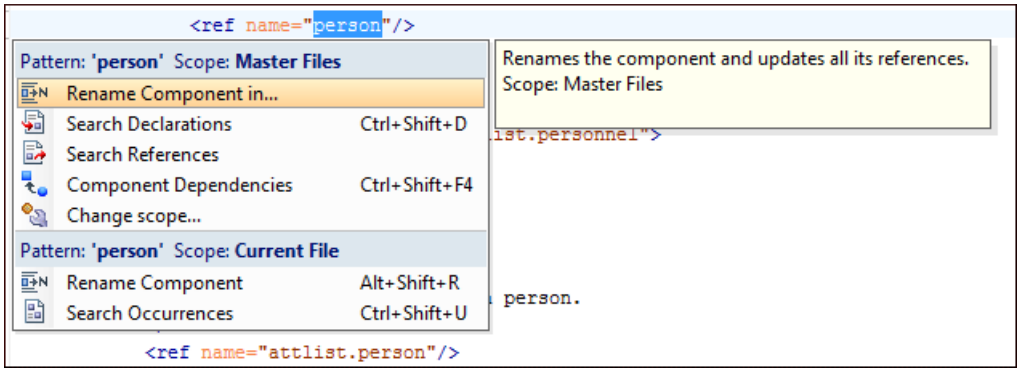


Figure 209: RNG Quick Assist Support

The quick assist support offers direct access to the following actions:

Rename Component in

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

Configuring a Custom Datatype Library for a RELAX NG Schema

A RELAX NG schema can declare a custom datatype library for the values of elements found in XML document instances. The datatype library must be developed in Java and it must implement the interface *specified on the www.thaiopensource.com website.*

The jar file containing the custom library and any other dependent jar file must be added to the classpath of the application, that is the jar files must be added to the folder `[OXYGEN_PLUGIN_DIR]/lib` and a line `<library name="lib/custom-library.jar"/>` must be added for each jar file to the file `[OXYGEN_PLUGIN_DIR]/plugin.xml` in the <runtime> element.

To load the custom library, restart the Eclipse platform.

Linking Between Development and Authoring

The **Author** mode is available on the Relax NG schema presenting the schema similar with the Relax NG compact syntax. It links to imported schemas and external references. Embedded Schematron is supported only in Relax NG schemas with XML syntax.

Editing NVDL Schemas

Some complex XML documents are composed by combining elements and attributes from different namespaces. More, the schemas that define these namespaces are not even developed in the same schema language. In such cases, it is difficult to specify in the document all the schemas which must be taken into account for validation of the XML document or for content completion. An NVDL (Namespace Validation Definition Language) schema can be used. This schema allows the application to combine and interleave multiple schemas of different types (W3C XML Schema, RELAX NG schema, Schematron schema) in the same XML document.

Oxygen XML Editor plugin provides a special type of editor for NVDL schemas. This editor presents the usual text view of an XML document synchronized in real time with an outline view. The outline view has two display modes: the *standard outline* mode and the *components* mode.

NVDL Schema Diagram

This section explains how to use the graphical diagram of a NVDL schema.

Introduction to NVDL Schema Diagram View

Oxygen XML Editor plugin provides a simple, expressive, and easy to read **Schema Diagram** view for NVDL schemas.

With this new feature you can easily develop complex schemas, print them on multiple pages or save them as JPEG, PNG, and BMP images. It helps both schema authors in developing the schema and content authors that are using the schema to understand it.

Oxygen XML Editor plugin is the only XML Editor to provide a side by side source and diagram presentation and have them real-time synchronized:

- The changes you make in the Editor are immediately visible in the Diagram (no background parsing).
- Changing the selected element in the diagram, selects the underlying code in the source editor.

Full Model View

When you create a schema document or open an existing one, the editor panel is divided in two sections: one containing the schema diagram and the second the source code. The diagram view has two tabbed panes offering a **Full Model View** and a **Logical Model View**.

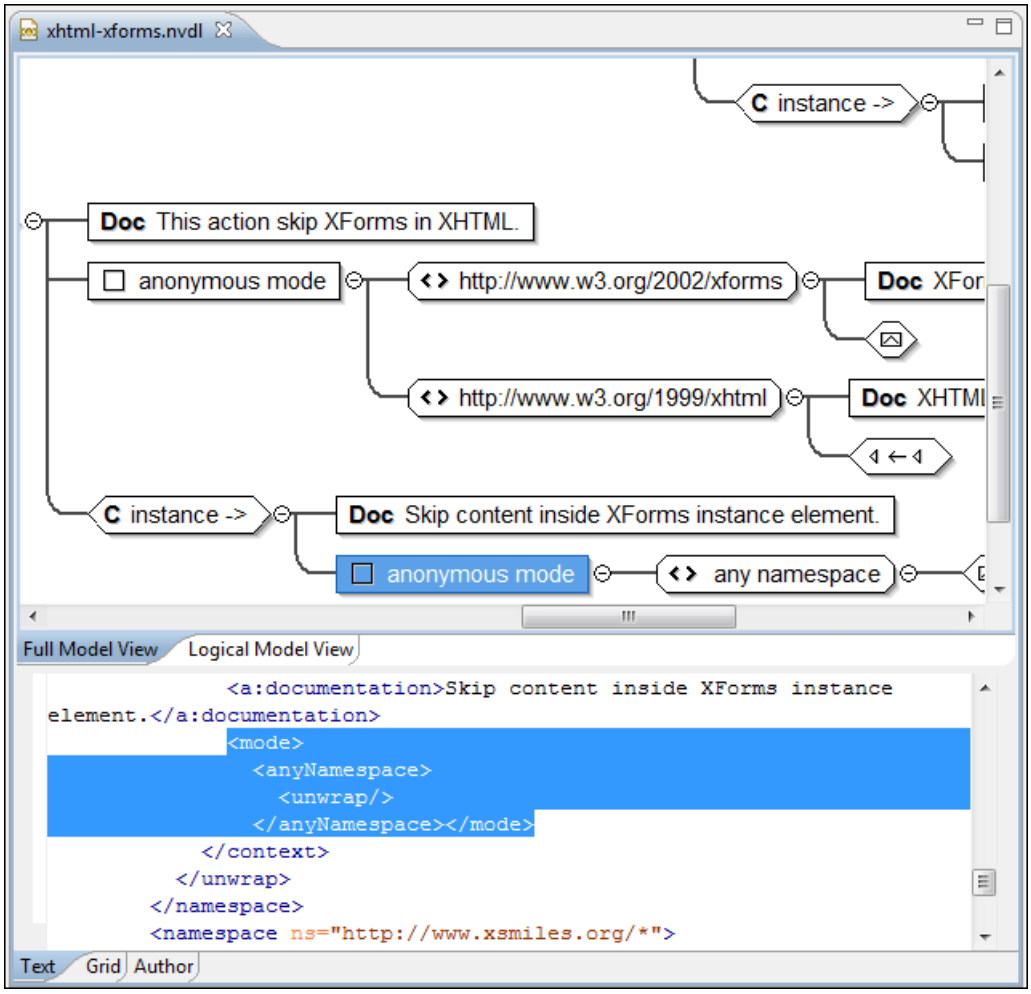


Figure 210: NVDL Schema Editor - Full Model View

The **Full Model View** renders all the NVDL elements with intuitive icons. This representation coupled with the synchronization support makes the schema navigation easy.

Double-click any diagram component in order to edit its properties.

Actions Available in the Diagram View

The contextual menu offers the following actions:

Show only the selected component

Depending on its state (selected/not selected), either the selected component or all the diagram components are shown.

Show Annotations

Depending on its state (selected/not selected), the documentation nodes are shown or hidden.

Auto expand to references

This option controls how the schema diagram is automatically expanded. For instance, if you select it and then edit a top-level element or you trigger a diagram refresh, the diagram will be expanded until it reaches the referenced components. If this option is left unchecked, only the first level of the diagram is expanded, showing the top-level elements. For large schemas, the editor disables this option automatically.

Collapse Children

Collapses the children of the selected view.

Expand Children

Expands the children of the selected view.

Print Selection

Prints the selected view.

Save as Image

Saves the current selection as image, in JPEG, BMP, SVG or PNG format.

Refresh

Refreshes the schema diagram according to the changes in your code (changes in your imported documents or those that are not reflected automatically in the compiled schema).

If the schema is not valid, you see only an error message in the **Logical Model View** instead of the diagram.

NVDL Outline View

The NVDL **Outline** view presents a list with the named or anonymous rules that appear in the diagram. It allows a quick access to a rule by name. It can be opened from the **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** > **Outline** menu.

Searching and Refactoring Actions in NVDL Schemas

Search Actions

The following search actions can be applied on `name`, `useMode`, and `startMode` attributes and are available from the **Search** submenu in the contextual menu of the current editor:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
- **Search Occurrences in File** - Searches all occurrences of the item at the cursor position in the currently edited file.

The following action is available from the **XSL** menu:

- **Show Definition** - Moves the cursor to its definition in the schema used by NVDL in order to validate it.

  Note: You can also use the **Ctrl Single-Click (Command Single-Click on OS X)** shortcut on a reference to display its definition.

Refactoring Actions

The following refactoring actions can be applied on `name`, `useMode`, and `startMode` attributes and are available from the **Refactoring** submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the cursor position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.

- **Rename Component in** - Opens the **Rename *component_type*** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

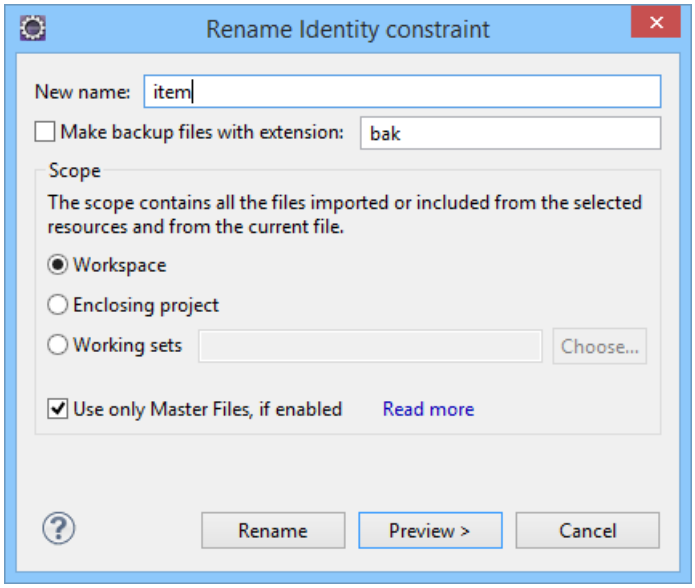


Figure 211: Rename Identity Constraint Dialog Box

Component Dependencies View for NVDL Schemas

The **Component Dependencies** view allows you to see the dependencies for a selected NVDL named mode. You can open the view from **Window** > **Show View** > **Other** > **<oXygen/> XML** > **Component Dependencies**.

If you want to see the dependencies of an NVDL mode, select the desired component in the editor and choose the **Component Dependencies** action from the contextual menu. The action is available for all named modes.

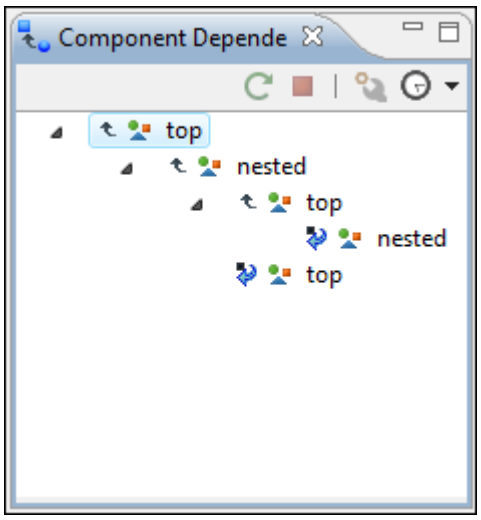


Figure 212: Component Dependencies View - Hierarchy for `test.nvdl`

In the **Component Dependencies** the following actions are available on the toolbar:

Refresh

Refreshes the dependencies structure.

Stop

Allows you to stop the dependencies computing.

Configure

Allows you to configure a search scope to compute the dependencies structure. If you decide to set the application to use automatically the defined scope for future operations, select the corresponding checkbox.

History

Repeats a previous dependencies computation.

The following actions are available in the contextual menu:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Shows the definition of the current selected component in the dependencies tree.

Tip: If a component contains multiple references to another component, a small table containing all references is displayed. When a recursive reference is encountered it is marked with a special icon .

Linking Between Development and Authoring

The **Author** mode is available on the NVDL scripts editor presenting them in a compact and easy to understand representation.

Editing JSON Documents

This section explains the features of the Oxygen XML Editor plugin JSON Editor and how to use them.

Editing JSON Documents in Text Mode

The **Text Mode** of the JSON editor provides the usual actions specific for a plain text editor: undo / redo, copy / paste, find / replace, drag and drop, and other editor actions like *validation* and *formatting and indenting (pretty print) document*.

You can use the two **Text** and **Grid** buttons available at the bottom of the editor panel if you want to switch between the editor **Text Mode** and **Grid Mode** .

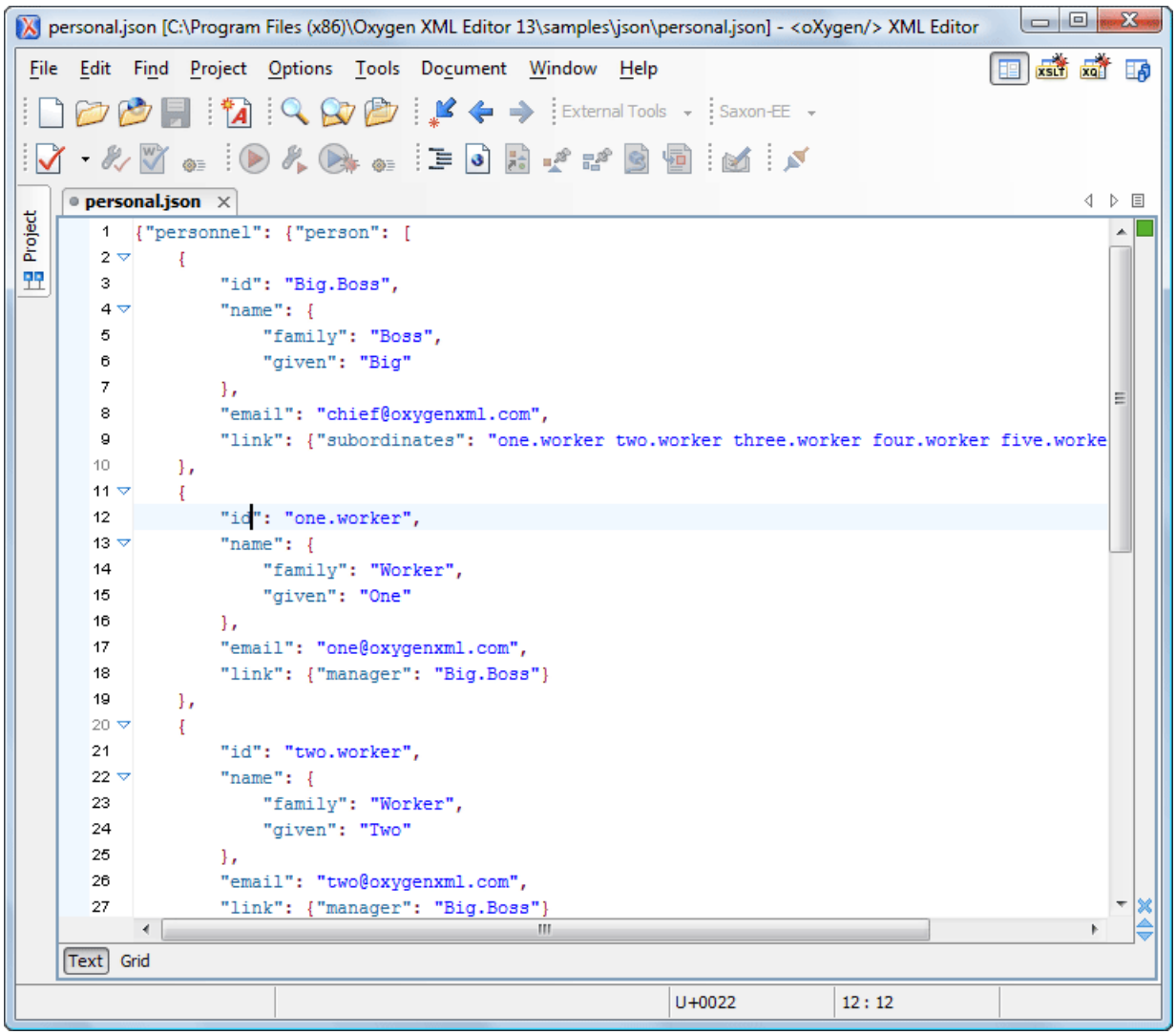


Figure 213: JSON Editor Text Mode

Syntax Highlight in JSON Documents

Oxygen XML Editor plugin supports *Syntax Highlight* for JavaScript / JSON editors and provides default configurations for the JSON set of tokens. You can customize the foreground color, background color and the font style for each JSON token type.

Folding in JSON

In a large JSON document, the data enclosed in the '{' and '}' characters can be collapsed so that only the needed data remain in focus. The *folding features available for XML documents* are available in JSON documents.

Editing JSON Documents in Grid Mode

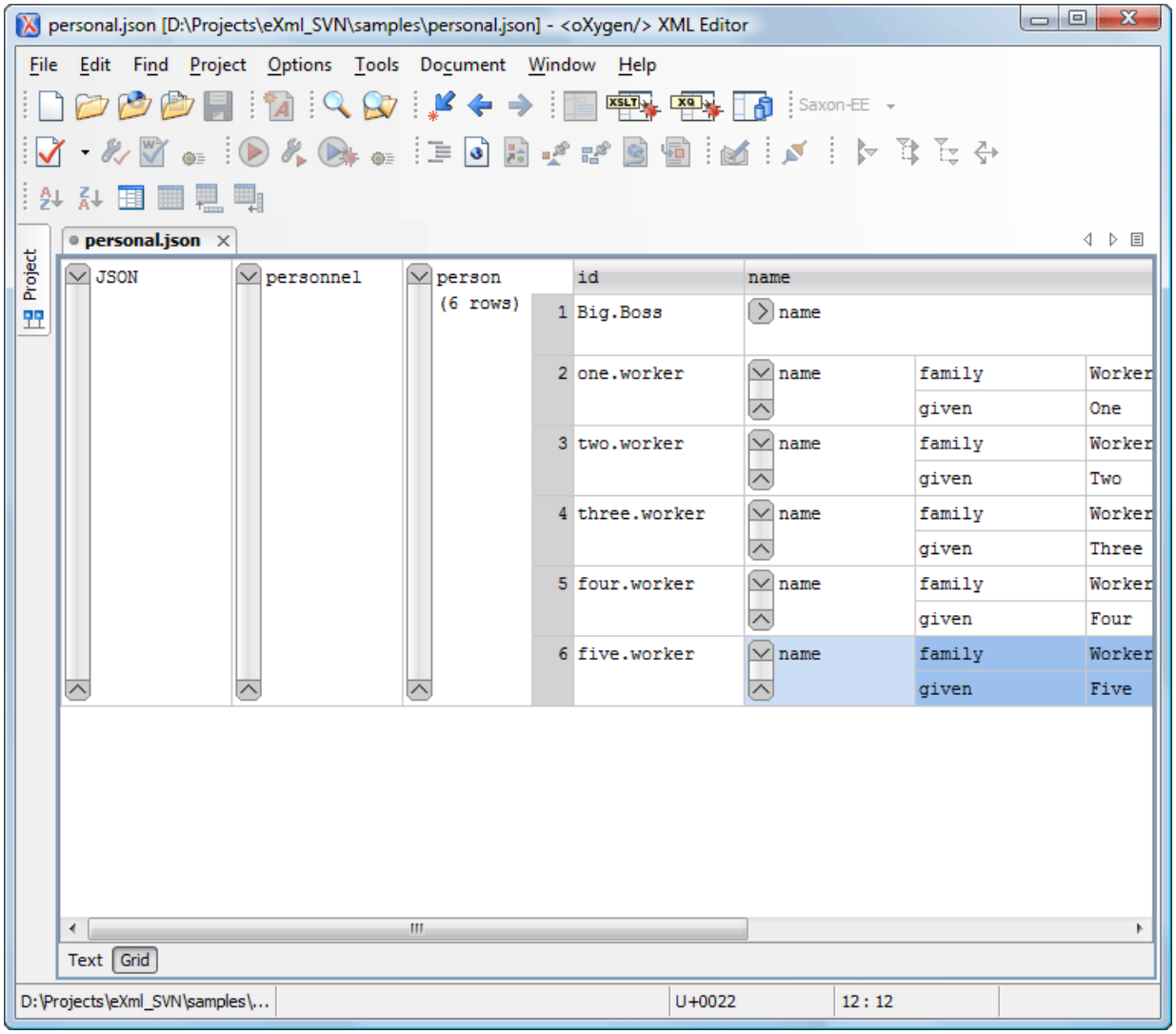


Figure 214: JSON Editor Grid Mode

Oxygen XML Editor plugin allows you to view and edit the JSON documents in the *Grid Mode*. The JSON is represented in **Grid** mode as a compound layout of nested tables in which the JSON data and structure can be easily manipulated with table-specific operations or drag and drop operations on the grid components. You can also use the following JSON-specific contextual actions:

Array

Useful when you want to convert a JSON *value* to *array*.

Insert value before

Inserts a value before the currently selected one.

Insert value after

Inserts a value after the currently selected one.

Append value as child

Appends a value as a child of the currently selected value.

You can *customize the JSON grid appearance* according to your needs. For instance you can change the font, the cell background, foreground, or even the colors from the table header gradients. The default width of the columns can also be changed.

JSON Outline View

The JSON **Outline** view displays the list of all the components of the JSON document you are editing. To enable the JSON **Outline** view, go to **Window** > **Show view** > **Outline**.

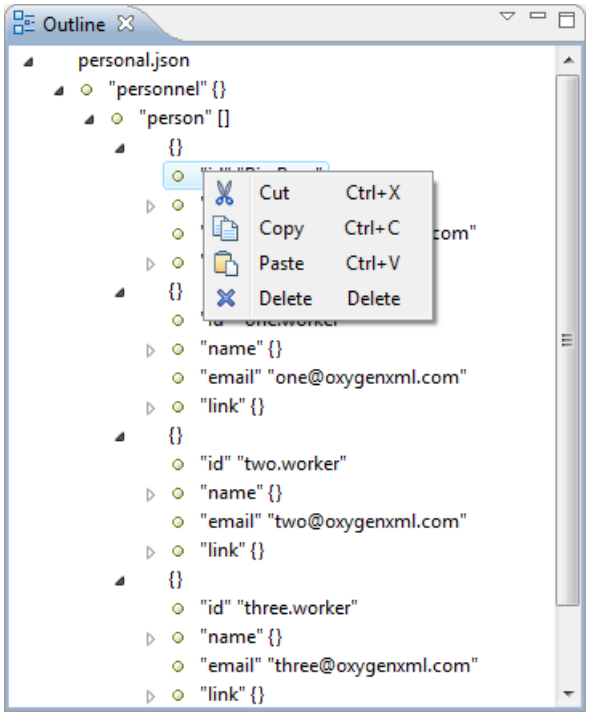


Figure 215: The JSON Outline View

The following actions are available in the contextual menu of the JSON **Outline** view:

- **Cut**
- **Copy**
- **Paste**
- **Delete**

The **View menu** on the JSON **Outline** view action bar allows you to enable **Selection update on cursor move**. This option controls the synchronization between the **Outline** view and source the document. Oxygen XML Editor plugin synchronizes the selection in the **Outline** view with the cursor moves or the changes you make in the JSON editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

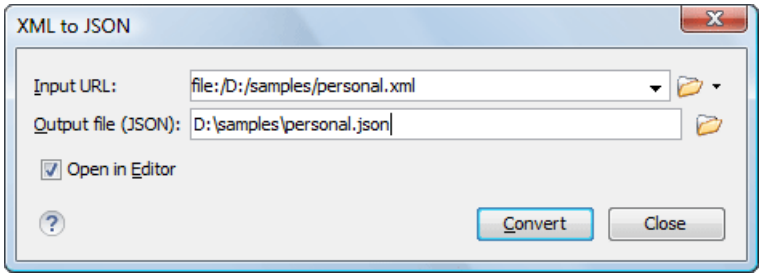
Validating JSON Documents

Oxygen XML Editor plugin includes a built-in JSON validator (based on the free JAVA source code available on www.json.org), integrated with the general validation support.

Convert XML to JSON

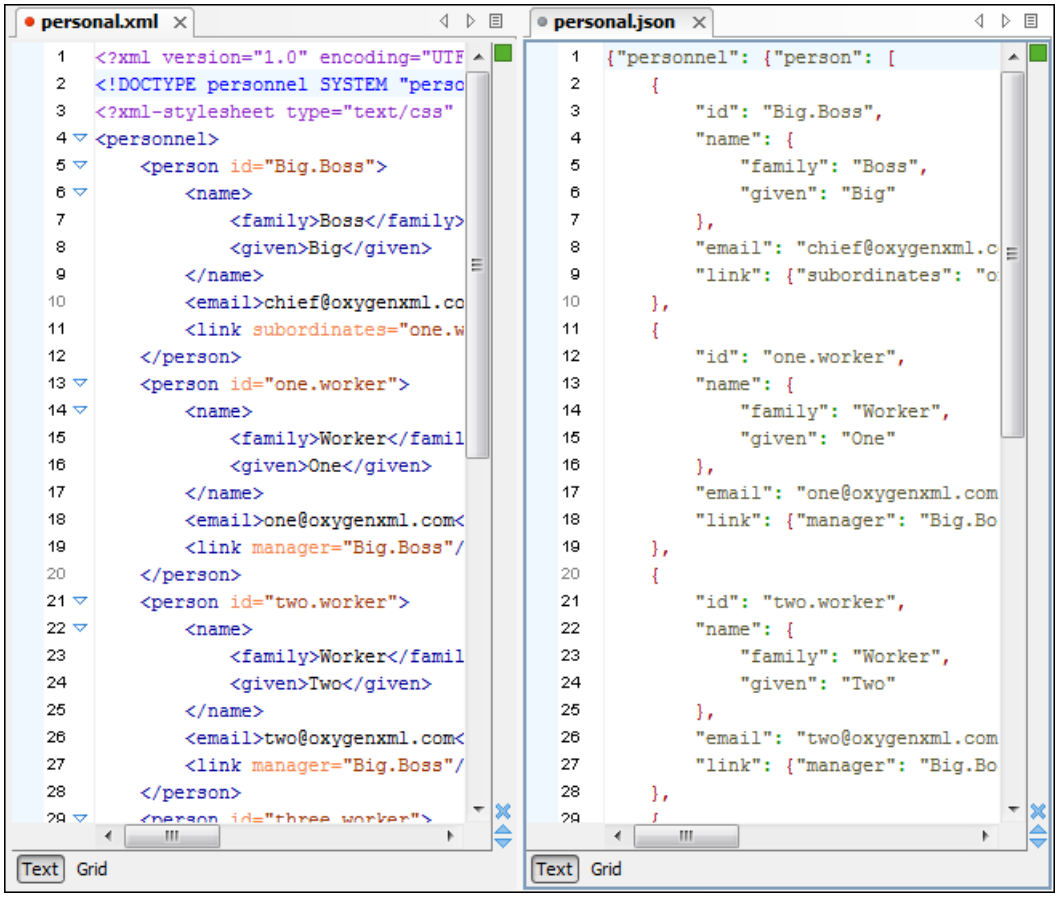
The steps for converting an XML document to JSON are the following:

1. Select the **XML to JSON** action from the **XML Tools** menu.
   The **XML to JSON** dialog box is displayed:



2. Choose or enter the **Input URL** of the XML document.
3. Choose the **Output file** that will contain the conversion JSON result.
4. Check the **Open in Editor** option to open the JSON result of the conversion in the Oxygen XML Editor plugin JSON Editor
5. Click the **OK** button.

The operation result will be a document containing the JSON conversion of the input XML URL.



Editing StratML Documents

Strategy Markup Language (StratML) is an XML vocabulary and schema for strategic plans. Oxygen XML Editor plugin supports StratML Part 1 (Strategic Plan) and StratML Part 2 (Performance Plans and Reports) and provides templates for the following documents:

- **Strategic Plan** (StratML Part 1)
- **Performance Plan** (StratML Part 2)
- **Performance Report** - (StratML Part 2)

- **Strategic Plan** - (StratML Part 2)

You can view the components of a StratML document in the **Outline** view. Oxygen XML Editor plugin implements a default XML with XSLT transformation scenario for this document type, called StratML to HTML.

Editing XLIFF Documents

XLIFF (*XML Localisation Interchange File Format*) is an XML-based format that was designed to standardize the way multilingual data is passed between tools during a localization process. Oxygen XML Editor plugin provides the following support for editing XLIFF documents:

XLIFF Version 1.2 and 2.0 Support:

- New file templates for XLIFF documents.
- A default CSS file (`xliff.css`) used for rendering XLIFF content in **Author** mode is stored in `[OXYGEN_DIR]/frameworks/xliff/css/`.
- Validation and content completion support using local catalogs. The default catalog (`catalog.xml`) for version 1.2 is stored in `[OXYGEN_DIR]/frameworks/xliff/xsd/1.2`, and for version 2.0 in `[OXYGEN_DIR]/frameworks/xliff/xsd/2.0`.

XLIFF Version 2.0 Enhanced Support:

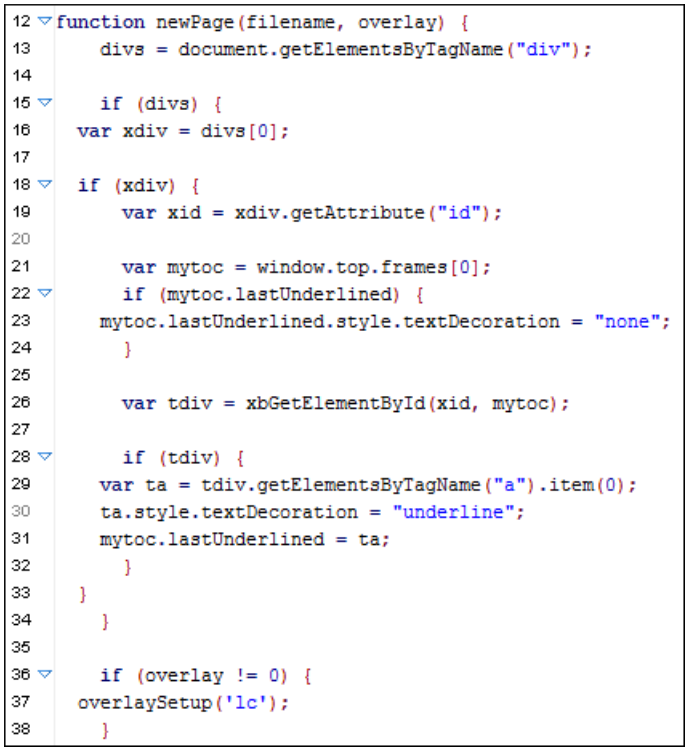
- Support for validating XLIFF 2.0 documents using modules. The default modules are stored in `[OXYGEN_DIR]/frameworks/xliff/xsd/2.0/modules`.

Editing JavaScript Documents

This section explains the features of the Oxygen XML Editor plugin JavaScript Editor and how you can use them.

JavaScript Editor Text Mode

Oxygen XML Editor plugin allows you to create and edit JavaScript files and assists you with useful features such as syntax highlight, content completion, and outline view. To enhance your editing experience, you can select entire blocks (parts of text delimited by brackets) by double-clicking somewhere inside the brackets.



```
function newPage(filename, overlay) {
    divs = document.getElementsByTagName("div");

    if (divs) {
  var xdiv = divs[0];

  if (xdiv) {
      var xid = xdiv.getAttribute("id");

      var mytoc = window.top.frames[0];
      if (mytoc.lastUnderlined) {
    mytoc.lastUnderlined.style.textDecoration = "none";
      }

      var tdiv = xbGetElementById(xid, mytoc);

      if (tdiv) {
    var ta = tdiv.getElementsByTagName("a").item(0);
    ta.style.textDecoration = "underline";
    mytoc.lastUnderlined = ta;
      }
  }
    }

    if (overlay != 0) {
  overlaySetup('lc');
    }
```

Figure 216: 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 cursor position. When you select this option, Oxygen XML Editor plugin moves the cursor 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.

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 cursor position.

Modify all matches

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 allows you to start editing. If matches with different letter cases are found, a dialog box is displayed that allows you select whether you want to modify only matches with the same letter case or all matches.

Open

Allows you to select one of the following actions:

- **Open File at Cursor** - select this action to open the source of the file located at the cursor position
- **Open File at Cursor in System Application** - select this action to open the source of the file located at the cursor position with the application that the system associates with the file

Compare

Select this option to open the **Diff Files** dialog box and compare the file you are editing with a file you choose in the dialog box.

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 cursor 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
- ▢ - object
- ● - property
- f() - method

Note: These icons decorate both the elements from the content completion list of proposals and from the **Outline** view.

Figure 217: 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.

If 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 Editor plugin present a list of all the components of the JavaScript document you are editing in the **Outline** view. To open the **Outline** view, go to **Window** > **Show View** > **Outline**.

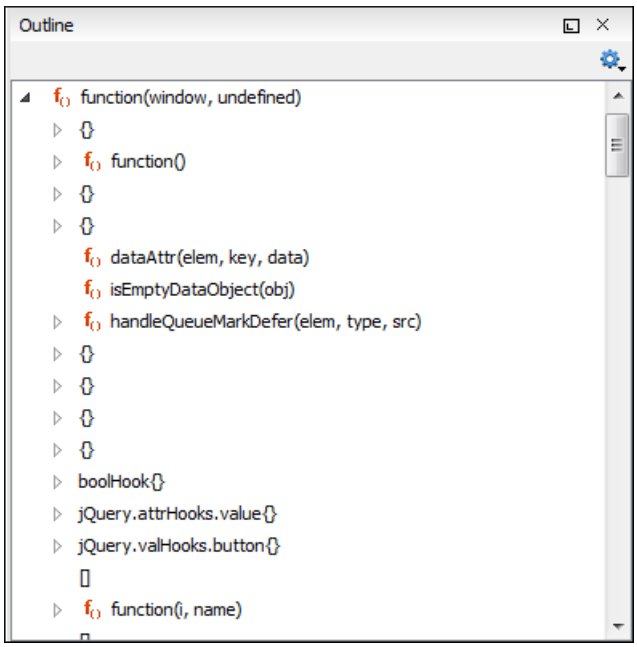


Figure 218: The JavaScript Outline View

The following icons decorate the elements in the **Outline** view depending on their type:

- - function
- - variable
- - object
- - property
- - method

The contextual menu of the JavaScript **Outline** view contains the usual **Cut**, **Copy**, **Paste**, and **Delete** actions. From the settings menu, you can enable the Update selection on cursor 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 Editor plugin 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 Editor plugin can validate a JavaScript document both on-request and automatically.

Editing XProc Scripts

An XProc script is edited as an XML document that is validated against a RELAX NG schema. If the script has an associated transformation scenario, then the XProc engine from the scenario is invoked as validating engine. The default engine for XProc scenarios is the Calabash engine which comes bundled with Oxygen XML Editor plugin version 17.1.

The content completion inside the element `input/inline` from the XProc namespace *http://www.w3.org/ns/xproc* offers elements from the following schemas depending both on the `port` attribute and the parent of the `input` element. When invoking the content completion inside the XProc element `inline`, the **Content Completion Assistant**'s proposals list is populated as follows:

- If the value of the `port` attribute is `stylesheet` and the `xslt` element is the parent of the `input` elements, the **Content Completion Assistant** offers XSLT elements.

- If the value of the `port` attribute is `schema` and the `validate-with-relax-ng` element is the parent of the `input` element, the **Content Completion Assistant** offers RELAX NG schema elements.
- If the value of the `port` attribute is `schema` and the `validate-with-xml-schema` element is the parent of the `input` element, the **Content Completion Assistant** offers XML Schema schema elements.
- If the value of the `port` attribute is `schema` and the `validate-with-schematron` element is the parent of the `input` element , the **Content Completion Assistant** offers either ISO Schematron elements or Schematron 1.5 schema elements.
- If the above cases do not apply, then the **Content Completion Assistant** offers elements from all the schemas from the above cases.

The XProc editor assists you in writing XPath expressions by offering a **Content Completion Assistant** and dedicated coloring schemes. To customize the coloring schemes, *open the **Preferences** dialog box* and go to **Editor** > **Syntax Highlight**.

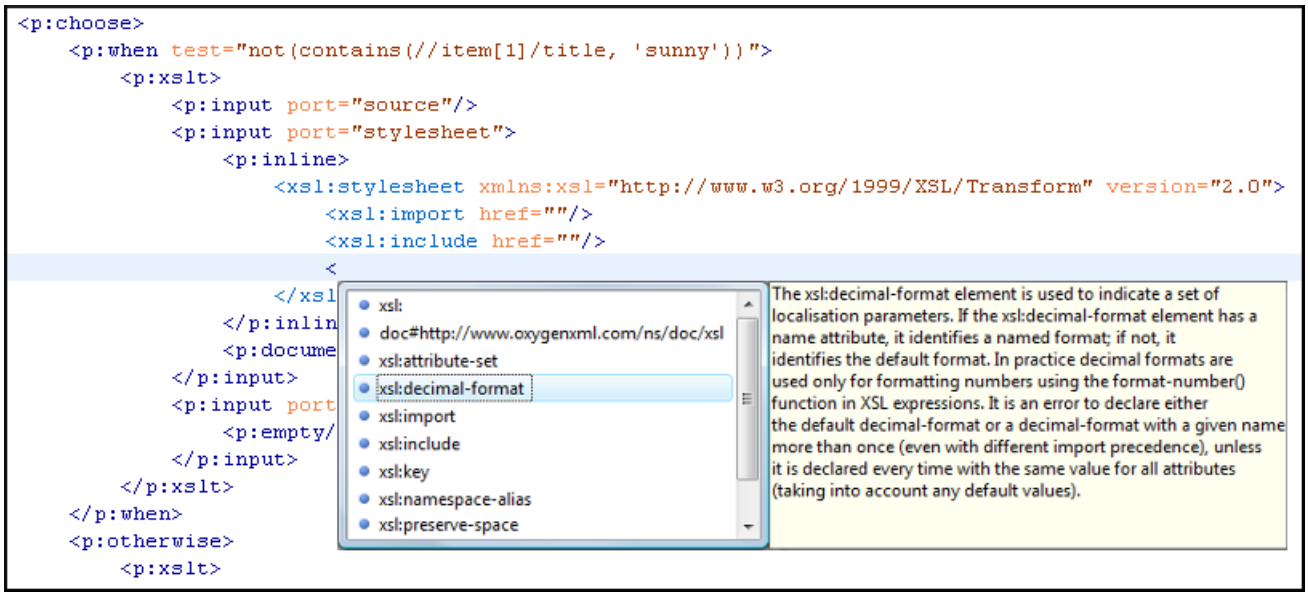


Figure 219: XProc Content Completion

Editing Schematron Schemas

Schematron is a simple and powerful Structural Schema Language for making assertions about patterns found in XML documents. It relies almost entirely on XPath query patterns for defining rules and checks. Schematron validation rules allow you to specify a meaningful error message. This error message is provided to you if an error is encountered during the validation stage.

The Skeleton XSLT processor is used for validation and conforms with ISO Schematron or Schematron 1.5. It allows you to validate XML documents against Schematron schemas or against combined RELAX NG / W3C XML Schema and Schematron.

Oxygen XML Editor plugin assists you in editing Schematron documents with schema-based content completion, syntax highlight, search and refactor actions, and dedicated icons for the **Outline** view. You can create a new Schematron schema using one of the Schematron templates available in the New Document wizard.

You can specify the query language binding to be used in the Schematron schema by doing the following:

- For embedded ISO schematron, *open the **Preferences** dialog box*, go to **XML** > **XML Parser** > **Schematron**, and select it in the **Embedded rules query language binding** option.
- For standalone ISO Schematron, specify the version by setting the query language to be used in a `queryBinding` attribute on the schema root element. For more information see the *Query Language Binding section of the Schematron specifications*.

- For Schematron 1.5 (standalone and embedded), *open the **Preferences** dialog box*, go to **XML** > **XML Parser** > **Schematron**, and select the version in the **XPath Version** option.

The Schematron editor renders the XPath expressions with dedicated coloring schemes . To customize the coloring schemes, *open the **Preferences** dialog box* and go to **Editor** > **Syntax Highlight**.

Note: When you create a Schematron document, Oxygen XML Editor plugin provides a built-in transformation scenario. You are able to use this scenario to obtain the XSLT style-sheet corresponding to the Schematron schema. You can apply this XSLT stylesheet to XML documents to obtain the Schematron validation results.

Validate an XML Document Against Schematron

To validate an XML document against a Schematron schema, select the **Validate** action from the **Validation** toolbar drop-down menu, or the **XML** menu, or from the **Validate** menu when invoking the contextual menu in the **Navigator** view.

If you would like to add a persistence association between your Schematron rules and the current edited XML document, use the **Associate Schema** action from the **Document** > **Schema** menu or the **Document** toolbar. A custom processing instruction is added into the document and the validation process will take into account the Schematron rules:

PI Added by the Associate Schema Action

```
<?xml-model href="percent.sch" type="application/xml"
schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

The possible errors that might occur during the validation process are presented in the **Problems** tab at the bottom area of the Oxygen XML Editor plugin window. Each error is flagged with a severity level that can be one of *warning*, *error*, *fatal* or *info*.

To set a severity level, Oxygen XML Editor plugin looks for the following information:

- The `role` attribute, which can have one of the following values:
  - `warn` or `warning`, to set the severity level to *warning*
  - `error`, to set the severity level to *error*
  - `fatal`, to set the severity level to *fatal*
  - `info` or `information`, to set the severity level to *info*
- The start of the message, after trimming leading white-spaces. Oxygen XML Editor plugin looks to match the following exact string of characters (case sensitive):
  - `Warning:`, to set the severity level to *warning*
  - `Error:`, to set the severity level to *error*
  - `Fatal:`, to set the severity level to *fatal*
  - `Info:`, to set the severity level to *info*

  Note: Displayed message does not contain the matched prefix.
- If none of the previous rules match, Oxygen XML Editor plugin sets the severity level to *error*.

Validating Schematron Documents

By default, a Schematron schema is validated as you type. To change this, *open the **Preferences** dialog box*, go to **Editor** > **Document Checking**, and disable the **Enable automatic validation** option.

To validate your Schematron document manually, select the **Validate** action from the **Validation** toolbar drop-down menu or the **XML** menu. When Oxygen XML Editor plugin validates a Schematron schema, it expands all the included modules so the entire schema hierarchy is validated. The validation problems are highlighted directly in the editor, making it easy to locate and fix any issues.

Oxygen XML Editor plugin offers an error management mechanism capable of pinpointing errors in XPath expressions and in the included schema modules.

Content Completion in Schematron Documents

Oxygen XML Editor plugin helps you edit a Schematron schema, offering, through the Content Completion Assistant, items that are valid at the cursor position. When you edit the value of an attribute that refers a component, the proposed components are collected from the entire schema hierarchy. For example, if the editing context is `phase/active/@pattern`, the Content Completion Assistant proposes all the defined patterns.

Note: For Schematron resources, the Content Completion Assistant collects its components starting from the master files. The master files can be defined in the project or in the associated validation scenario. For further details about the **Master Files** support go to *Defining Master Files at Project Level*.

If the editing context is an attribute value that is an XPath expression (like `assert/@test` or `report/@test`), the Content Completion Assistant offers the names of XPath functions, the XPath axes, and user-defined variables.

The **Content Completion Assistant** displays XSLT 1.0 functions and optionally XSLT 2.0 / 3.0 functions in the attributes *path*, *select*, *context*, *subject*, *test* depending on *the Schematron options* that are set in Preferences pages. If the Saxon 6.5.5 namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.6.0.7 namespace is declared in the Schematron schema (`xmlns:saxon="http://saxon.sf.net/"`) the content completion also displays the XSLT Saxon extension functions as in the following figure:

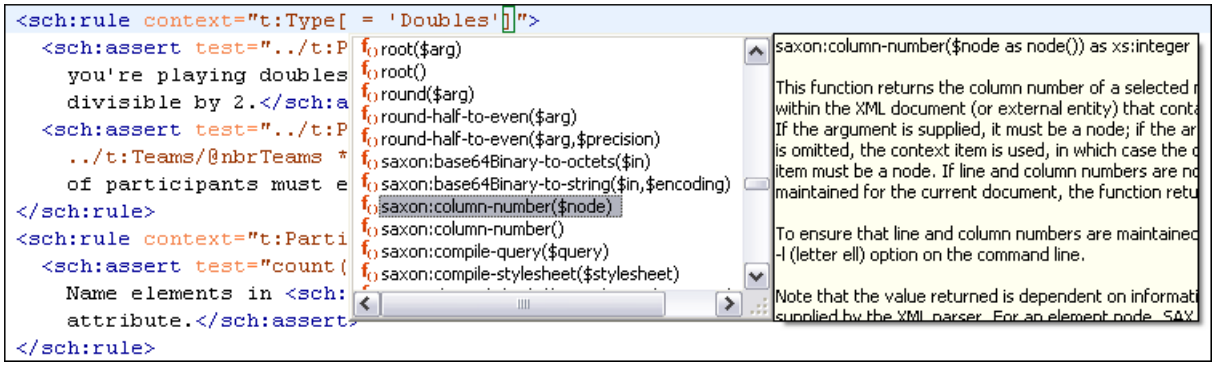


Figure 220: XSLT extension functions in Schematron schemas documents

The **Content Completion Assistant** also includes *code templates that can be used to quickly insert code fragments* into Schematron documents.

RELAX NG/XML Schema with Embedded Schematron Rules

Schematron rules can be embedded into an XML Schema through annotations (using the `appinfo` element), or in any element on any level of a RELAX NG Schema (taking into account that the RELAX NG validator ignores all elements that are not in the RELAX NG namespace).

Oxygen XML Editor plugin accepts such documents as Schematron validation schemas and it is able to extract and use the embedded rules. To validate an XML document with both RELAX NG schema and its embedded Schematron rules, you need to associate the document with both schemas. For example:

```
<?xml-model href="percent.rng" type="application/xml"
schematypens="http://relaxng.org/ns/structure/1.0"?>
<?xml-model href="percent.rng" type="application/xml"
schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

The second association validates your document with Schematron rules extracted from the RELAX NG Schema. Similarly, you can specify an XML Schema having the embedded Schematron rules.

```
<?xml-model href="percent.xsd" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

Note: When you work with XML Schema or Relax NG documents that have embedded Schematron rules Oxygen XML Editor plugin provides two built-in validation scenarios: **Validate XML Schema with embedded Schematron** for XML schema , and **Validate Relax NG with embedded Schematron** for Relax NG. You can use one of these scenarios to validate the embedded Schematron rules.

Editing Schematron Schema in the Master Files Context

Smaller interrelated modules that define a complex Schematron cannot be correctly edited or validated individually, due to their interdependency with other modules. For example, a diagnostic defined in a main schema document is not visible when you edit an included module. Oxygen XML Editor plugin provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

You cat set a main Schematron document either using the *master files support from the **Navigator** view*, or using a validation scenario.

To set a main file using a validation scenario, add validation units that point to the main schemas. Oxygen XML Editor plugin warns you if the current module is not part of the dependencies graph computed for the main schema. In this case, it considers the current module as the main schema.

The advantages of editing in the context of main file include:

- Correct validation of a module in the context of a larger schema structure.
- **Content Completion Assistant** displays all the referable components valid in the current context. This include components defined in modules other than the currently edited one.

Schematron Outline View

The Schematron **Outline** view presents a list of components in a tree-like structure. It allows a quick access to a component by name. By default it is displayed on screen, but if you closed it you can reopen it from **Window** > **Show View** > **Outline**.

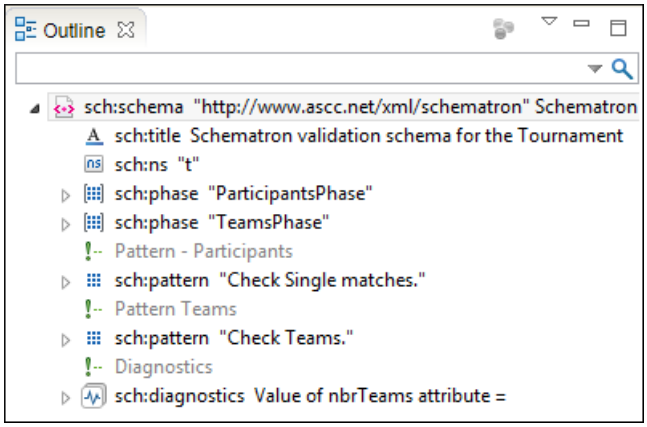


Figure 221: Schematron Outline View

The following actions are available in the **View menu** on the **Outline** view action bar:

Filter returns exact matches

The text filter of the **Outline** view returns only exact matches.

Selection update on cursor move

Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the cursor moves or the changes in the editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

Flat presentation mode of the filtered results

When active, the application flattens the filtered result elements to a single level.

Show comments and processing instructions

Show/hide comments and processing instructions in the **Outline** view.

Show element name

Show/hide element name.

Show text

Show/hide additional text content for the displayed elements.

Show attributes

Show/hide attribute values for the displayed elements. The displayed attribute values can be changed from *the Outline preferences panel*.

Configure displayed attributes

Displays the *XML Structured Outline preferences page*.

The following contextual menu actions are also available in the **Outline** view:

Append Child

Displays a list of elements that you can insert as children of the current element.

Insert Before

Displays a list of elements that you can insert as siblings of the current element, before the current element.

Insert After

Displays a list of elements that you can insert as siblings of the current element, after the current element.

Edit Attributes

Opens a dialog box that allows you to edit the attributes of the currently selected component.

Toggle Comment

Comments/uncomments the currently selected element.

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Delete

Deletes the currently selected component.

Expand All

Expands the structure of a component in the **Outline** view.

Collapse All

Collapses the structure of all the component in the **Outline** view.

The upper part of the **Outline** view contains a filter box that 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.

Schematron Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a Schematron schema. To open this view, go to **Window** > **Show View** > **Other** > **Oxygen XML Editor plugin** > **Resource Hierarchy/Dependencies**.

If you want to see the hierarchy of a schema, select the desired schema in the project view and choose **Resource Hierarchy** from the contextual menu.

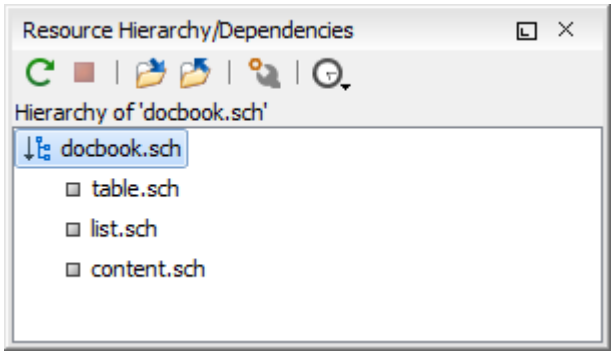


Figure 222: Resource Hierarchy/Dependencies View

If you want to see the dependencies of a schema, select the desired schema in the project view and choose **Resource Dependencies** from the contextual menu.

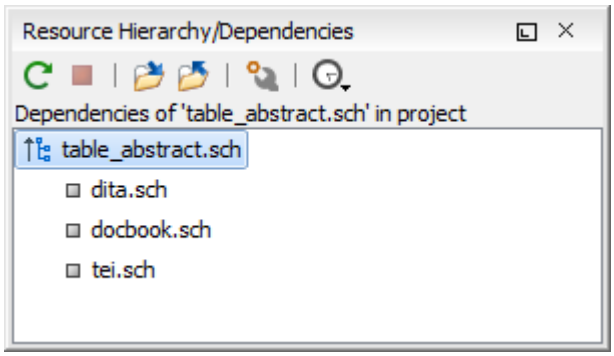


Figure 223: Resource Hierarchy/Dependencies View - Dependencies for `table_abstract.sch`

The following actions are available in the **Resource Hierarchy/Dependencies** view:

Refresh

Refreshes the Hierarchy/Dependencies structure.

Stop

Stops the hierarchy/dependencies computing.

Show Hierarchy

Allows you to choose a resource to compute the hierarchy structure.

Show Dependencies

Allows you to choose a resource to compute the dependencies structure.

Configure

Allows you to configure a scope to compute the dependencies structure. There is also an option for automatically using the defined scope for future operations.

History

Provides access to the list of previously computed dependencies. Use the **Clear history** button to remove all items from this list.

The contextual menu contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Copy location

Copies the location of the resource.

Move resource

Moves the selected resource.

Rename resource

Renames the selected resource.

Show Resource Hierarchy

Shows the hierarchy for the selected resource.

Show Resource Dependencies

Shows the dependencies for the selected resource.

 **Add to Master Files**

Adds the currently selected resource in *the **Master Files** directory*.

Expand All

Expands all the children of the selected resource from the Hierarchy/Dependencies structure.

Collapse All

Collapses all children of the selected resource from the Hierarchy/Dependencies structure.

Tip: When a recursive reference is encountered in the Hierarchy view, the reference is marked with a special icon .

Note: The **Move resource** or **Rename resource** actions give you the option to *update the references to the resource*.

Moving/Renaming Schematron Resources

You are able to move and rename a resource presented in the **Resource/Hierarchy Dependencies** view, using the **Rename resource** and **Move resource** refactoring actions from the contextual menu.

When you select the **Rename** action in the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Rename resource** dialog box is displayed. The following fields are available:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references** - Enable this option to update the references to the resource you are renaming.

When you select the **Move** action from the contextual menu of the **Resource/Hierarchy Dependencies** view, the **Move resource** dialog box is displayed. The following fields are available:

- **Destination** - Presents the path to the current location of the resource you want to move and gives you the option to introduce a new location.
- **New name** - Presents the current name of the moved resource and gives you the option to change it.
- **Update references of the moved resource(s)** - Enable this option to update the references to the resource you are moving, in accordance with the new location and name.

If the **Update references of the moved resource(s)** option is enabled, a **Preview** option (which opens the **Preview** dialog box) is available for both actions. The **Preview** dialog box presents a list with the resources that are updated.

Highlight Component Occurrences in Schematron Documents

When you position your mouse cursor over a component in a Schematron document, Oxygen XML Editor plugin searches for the component declaration and all its references and highlights them automatically.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected.

To change the default behavior of **Highlight Component Occurrences**, *open the **Preferences** dialog box* and go to **Editor** > **Mark Occurrences**. You can also trigger a search using the **Search** > **Search Occurrences in File** **Ctrl Shift U (Meta Shift U on OS X)** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.

Searching and Refactoring Operations in Schematron Documents

Search Actions

The following search actions can be applied on `pattern`, `phase`, or `diagnostic` types and are available from the **Search** submenu in the contextual menu of the current editor:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
- **Search Occurrences in File** - Searches all occurrences of the item at the cursor position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on `pattern`, `phase`, or `diagnostic` types and are available from the **Refactoring** submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the cursor position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
- **Rename Component in** - Opens the **Rename *component_type*** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

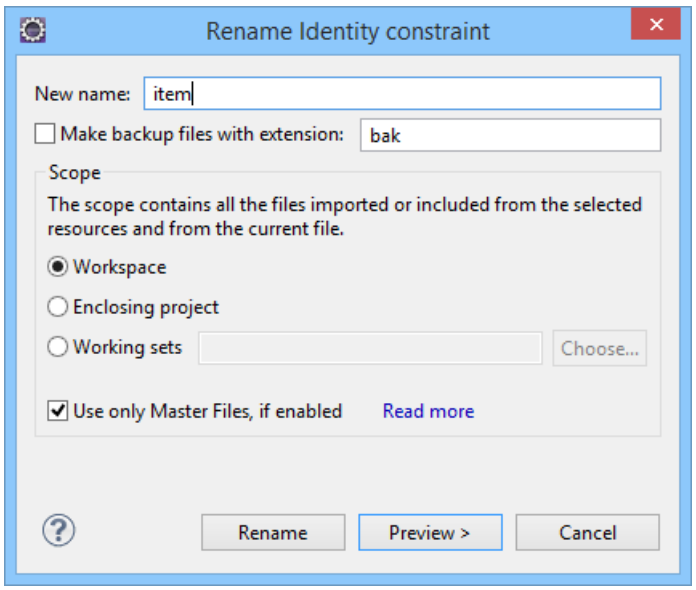


Figure 224: Rename Identity Constraint Dialog Box

Searching and Refactoring Operations Scope in Schematron Documents

The *scope* is a collection of documents that define the context of a search and refactor operation. To control it you can use the **Change scope** operation, available in the Quick Fix action set or on the **Resource Hierarchy/Dependency View** toolbar. You can restrict the scope to the current project or to one or multiple working sets. The **Use only Master Files, if enabled** check-box allows you to restrict the scope of the search and refactor operations to the resources from the **Master Files** directory. Click **read more** for details about the *Master Files support*.

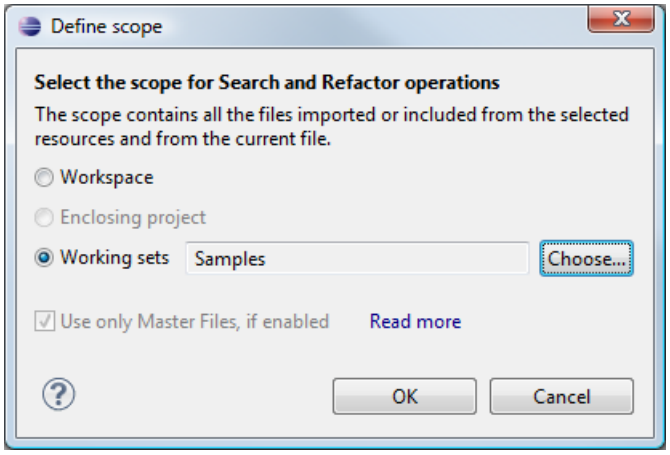


Figure 225: Define Scope Dialog Box

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.

Quick Assist Support in Schematron Documents

The *Quick Assist* feature is activated automatically when the cursor is positioned over the name of a component. It is accessible via a yellow bulb help marker placed on the cursor line, in the line number stripe on the left side of the editor. Also, you can invoke the quick assist menu by using the **Ctrl + 1** keys (**Meta 1** on Mac OS X) keyboard shortcuts.

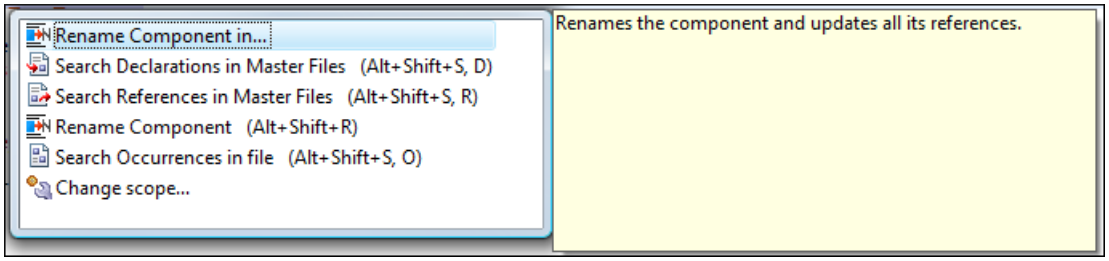


Figure 226: Schematron Quick Assist Support

The quick assist support offers direct access to the following actions:

Rename Component in

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

Editing Schematron Quick Fixes

Oxygen XML Editor plugin provides support for editing the *Schematron Quick Fixes*. You can define a library of quick fixes by editing them directly in the current Schematron file or in a separate file. Oxygen XML Editor plugin assists you in editing Schematron Quick Fixes with schema-based content completion, syntax highlighting, and validation as you type.

This section includes details about the Schematron Quick Fixes feature and how to customize them.

Customizing Schematron Quick Fixes

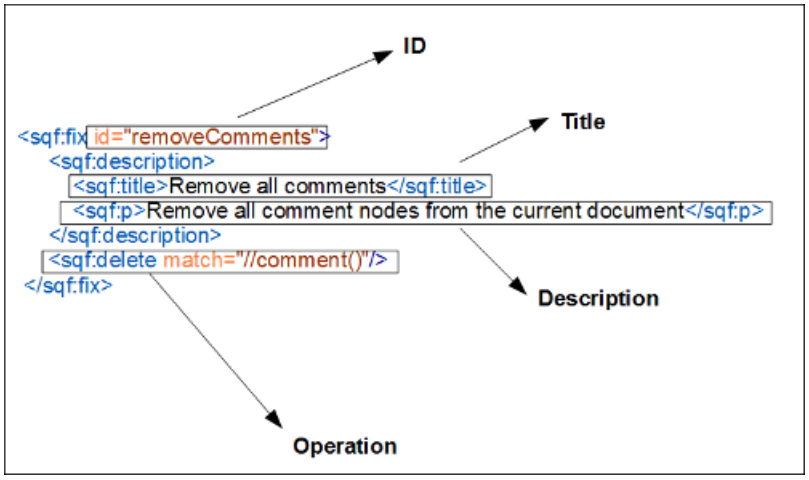
You can customize Schematron Quick Fixes by editing them directly in the current Schematron file or in a separate file. The Schematron Quick Fixes are an extension of the Schematron language and they allow you to define fixes for Schematron error messages. You can refer the quick fixes from the `assert` or `report` elements in the values of the `sqf:fix` attributes.

Defining a Schematron Quick Fix

The basics of a Schematron Quick Fix is defined by an ID, name, description, and the operations to be executed.

- **ID** - Defined by the `id` attribute from the `fix` element and must be unique in the current context. It is used to refer the quick fix from a `report` or `assert` element.
- **Name** - The name of the quick fix is defined by the `title` element.

- **Description** - Defined by the text in the paragraphs (`p`) of the `description` element.
- **Operations** - The following types of operations are supported:
  - `<sqf:add>` - To add a new node or fragment in the document.
  - `<sqf:delete>` - To remove a node from the document.
  - `<sqf:replace>` - To replace a node with another node or fragment.
  - `<sqf:stringReplace>` - To replace text content with other text or a fragment.



The assertion message that generates the quick fix is added as the `description` of the problem to be fixed. The `title` is presented as the name of the quick fix. The content of the paragraphs (`p`) within the `description` element are presented in the tooltip message when the quick fix is selected.

Schematron Quick Fix Operations

Add

The `<sqf:add>` element allows you to add a node to the instance. An anchor node is required to select the position for the new node. The anchor node can be selected by the `match` attribute. Otherwise, it is selected by the `context` attribute of the rule.

The `target` attribute defines the name of the node to be added. It is required if the value of the `node-type` attribute is set to anything other than "comment".

The `<sqf:add>` element has a `position` attribute and it determines the position relative to the anchor node. The new node could be specified as the first child of the anchor node, the last child of the anchor node, before the anchor node, or after the anchor node (`first-child` is the default value). If you want to add an attribute to the anchor node, do not use the `position` attribute.

Note: If you insert an element and its content is empty, Oxygen XML Editor plugin will insert the required element content.

An Example of the `<sqf:add>` Element:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
    xmlns:sqf="http://www.schematron-quickfix.com/validator/process" queryBinding="xslt2">
    <pattern>
        <rule context="head">
            <assert test="title" sqf:fix="addTitle">title element is missing.</assert>
            <sqf:fix id="addTitle">
                <sqf:description>
                    <sqf:title>Insert title element.</sqf:title>
                </sqf:description>
                <sqf:add target="title" node-type="element">Title text</sqf:add>
            </sqf:fix>
        </rule>
    </pattern>
</schema>
```

Specific Add Operations:

- **Insert Element** - To insert an element, use the `<sqf:add>` element, set the value of the `node-type` to "element", and specify the element *QName* with the `target` attribute. If the element has a prefix, it must be defined in the Schematron using a namespace declaration (`<ns uri="namespace" prefix="prefix"/>`).
- **Insert Attribute** - To insert an attribute, use the `<sqf:add>` element, set the value of the `node-type` to "attribute", and specify the attribute *QName* with the `target` attribute. If the attribute has a prefix, it must be defined in the Schematron using a namespace declaration (`<ns uri="namespace" prefix="prefix"/>`).
- **Insert Fragment** - If the `node-type` is not specified, the `<sqf:add>` element will insert an XML fragment. The XML fragment must be well formed. You can specify the fragment in the `add` element or by using the `select` attribute.
- **Insert Comment** - To insert a comment, use the `<sqf:add>` element and set the value of the `node-type` to "comment".
- **Insert Processing Instruction** - To insert a processing instruction, use the `<sqf:add>` element, set the value of the `node-type` to "pi" or "processing-instruction", and specify the name of the processing instruction in the `target` attribute.

Delete

The `<sqf:delete>` element allows you to remove any type of node (such as elements, attributes, text, comments, or processing instructions). To specify nodes for deletion the `<sqf:delete>` element can include a `match` attribute that is an XPath expression (the default value is `.`). If the `match` attribute is not included, it deletes the context node of the Schematron rule.

An Example of the `<sqf:delete>` Element:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2"
    xmlns:sqf="http://www.schematron-quickfix.com/validator/process">
    <pattern>
        <rule context="*[@xml:lang]">
            <report test="@xml:lang" sqf:fix="remove_lang">
                The attribute "xml:lang" is forbidden.</report>
            <sqf:fix id="remove_lang">
                <sqf:description>
                    <sqf:title>Remove "xml:lang" attribute</sqf:title>
                </sqf:description>
                <sqf:delete match="@xml:lang"/>
            </sqf:fix>
        </rule>
    </pattern>
</schema>
```

Replace

The `<sqf:replace>` element allows you to replace nodes. Similar to the `<sqf:delete>` element, it can include a `match` attribute. Otherwise, it replaces the context node of the rule. The `<sqf:replace>` element has three tasks. It identifies the nodes to be replaced, defines the replacing nodes, and defines their content.

An Example of the `<sqf:replace>` Element:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
    xmlns:sqf="http://www.schematron-quickfix.com/validator/process" queryBinding="xslt2">
    <pattern>
        <rule context="title">
            <report test="exists(ph)" sqf:fix="resolvePh" role="warn">
                ph element is not allowed in title.</report>
            <sqf:fix id="resolvePh">
                <sqf:description>
                    <sqf:title>Change the ph element into text</sqf:title>
                </sqf:description>
                <sqf:replace match="ph">
                    <value-of select="."/>
                </sqf:replace>
            </sqf:fix>
        </rule>
    </pattern>
</schema>
```

Other Attributes for Replace Operations:

- **`node-type`** - Determines the type of the replacing node. The permitted values include:
  - `keep` - Keeps the node type of the node to be replaced.
  - `element` - Replaces the node with an element.

- `attribute` - Replaces the node with an attribute.
- `pi` - Replaces the node with a processing instruction.
- `comment` - Replaces the node with a comment.

- **`target`** - By using a *QName* it gives the replacing node a name. This is necessary when the value of the `node-type` attribute is anything other than "comment".
- **`select`** - Allows you to choose the content of the replacing nodes. You can use XPath expressions with the `select` attribute. If the `select` attribute is not specified then the content of the `<sqf:replace>` element is used instead.

String Replace

The `<sqf:stringReplace>` element is different from the others. It can be used to find a sub-string of text content and replace it with nodes or other strings.

Attributes for the String Replace Operation:

- **`match`** - Allows you to select text nodes that contain the sub-strings you want to replace.
- **`select`** - Allows you to select the replacing fragment, in case you do not want to set it in the content of the `stringReplace` element.
- **`regex`** - Matches the sub-strings using a regular expression.

Note: Regular expressions in the `<sqf:stringReplace>` element always has the *dot matches all* flag set to "true". Therefore, the line terminator will also be matched by the regular expression.

Attention: The context of the content within the `<sqf:stringReplace>` element is set to the whole text node, rather than the current sub-string.

An Example of the `<sqf:stringReplace>` Element:

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron"
    xmlns:sqf="http://www.schematron-quickfix.com/validator/process" queryBinding="xslt2">
    <sch:pattern>
        <sch:rule context="text()">
            <sch:report test="matches(., '[oO][xX]ygen')" sqf:fix="changeWord">The oXygen word is not
allowed</sch:report>
            <sqf:fix id="changeWord">
                <sqf:description>
                    <sqf:title>Replace word with product</sqf:title>
                </sqf:description>
                <sqf:stringReplace regex="[oO][xX]ygen"><ph keyref="product"/></sqf:stringReplace>
            </sqf:fix>
        </sch:rule>
    </sch:pattern>
</sch:schema>
```

Formatting and Indenting Inserted Content

The content that is inserted by the **Add**, **Replace**, or **String Replace** operations is automatically indented unless you set the value of the `xml:space` attribute to `preserve` on the operation element. There are several methods available to format the content that is inserted:

- **`xsl:text`** - You can use an `xsl:text` element to format the inserted content and keep the automatic indentation, as in the following example:

```
<sqf:add position="last-child">
    <row><xsl:text>
    </xsl:text>
        <entry>First column</entry><xsl:text>
        </xsl:text>
        <entry>Second column</entry><xsl:text>
        </xsl:text>
    </row><xsl:text>
    </xsl:text>
</sqf:add>
```

- **`xml:space`** - Use the `xml:space` attribute and set its value to `preserve` to format the content and specify the spacing between elements, as in the following example:

```
<sqf:add node-type="element" target="codeblock" xml:space="preserve">
    /* a long sample program */
    Do forever
     Say "Hello, World"
    End</sqf:add>
```

The *Use-When* Condition

To restrict a quick fix or a specific operation to only be available if certain conditions are met, the `use-when` attribute can be included in the `<sqf:fix>` element or any of the SQF operation elements. The condition of the `use-when` attribute is an XPath expression and the fix or operation will be performed only if the condition is satisfied. In the following example, the `use-when` condition is applied to the `<sqf:fix>` element:

```
<sqf:fix id="last" use-when="$colWidthSummarized - 100 lt  $lastWidth" role="replace">
    <sqf:description>
        <sqf:title>Subtract the excessive width from the last element.</sqf:title>
    </sqf:description>
    <let name="delta" value="$colWidthSummarized - 100"/>
    <sqf:add match="html:col[last()]" target="width" node-type="attribute">
        <let name="newWidth" value="number(substring-before(@width,'%')) - $delta"/>
        <value-of select="concat($newWidth,'%')"/>
    </sqf:add>
</sqf:fix>
```

Executing Schematron Quick Fixes in Documents Other than the Current One

You can apply Schematron Quick Fixes over the nodes from referred documents (referred using XInclude or external entities), and you can access them as nodes in your current document.

Also, you can apply the quick fixes over other documents using the `doc()` function in the value of the `match` attribute. For example, you can add a new key in the `keylist.xml` file using the following operation:

```
<sqf:add match="doc('keylist.xml')/KeyList" target="Key" node-type="element" select="Key2">
```

Additional Elements Supported in the Schematron Quick Fixes

`<sqf:call-fix>`

This element calls another quick fix within a quick fix. The called quick fix must be defined globally or in the same Schematron rule as the calling quick fix. A calling quick fix adopts the activity elements of the called quick fix and should not include other activity elements. You can also specify which parameters are sent by using the `<sqf:with-param>` child element.

`<sqf:group>`

Allows you to group multiple quick fixes and refer them from an `assert` or `report` element.

`<sqf:fixes>`

Is defined globally and contains global fixes and groups of fixes.

`<sqf:keep>`

Used to copy the selected nodes that are specified by the `select` attribute.

Note: In Oxygen XML Editor plugin the copied nodes cannot be manipulated by the current or other activity elements.

`<sqf:param>`

Defines a parameter for a quick fix. If the parameter is defined as `abstract` then the `type` and default value should not be specified and the fix can be called from an abstract pattern that defines this parameter.

<sqf:user-entry>

Allows you to specify a value that will be inserted after the user selects the quick fix. If multiple `user-entry` elements are defined, Oxygen XML Editor plugin will display a dialog box for each one, in which the user can insert values.

Other SQF Notes

Note: The `sqf:defaultFix` attribute is ignored in Oxygen XML Editor plugin.

For more details on editing Schematron Quick Fixes, go to: *Schematron Quick Fix Specifications*

Validating Schematron Quick Fixes

By default, Schematron Quick Fixes are validated as you edit them within the Schematron file or while editing them in a separate file. To change this, *open the **Preferences** dialog box*, go to **Editor** > **Document Checking**, and disable the **Enable automatic validation** option.

To validate Schematron Quick Fixes manually, select the **Validate** action from the **Validation** toolbar drop-down menu or the **XML** menu. The validation problems are highlighted directly in the editor, making it easy to locate and fix any issues.

Content Completion in SQF

Oxygen XML Editor plugin helps you edit Schematron Quick Fixes embedded in a Schematron document, offering, through the Content Completion Assistant, items that are valid at the cursor position. When you edit the value of an attribute that references a quick fix *id*, the ids are collected from the entire definition scope. For example, if the editing context is `assert/@sqf:fix`, the Content Completion Assistant proposes all fixes defined locally and globally.

If the editing context is an attribute value that is an XPath expression (such as `sqf:add/@match` or `replace/@select`), the Content Completion Assistant offers the names of XPath functions, the XPath axes, and user-defined variables and parameters.

The **Content Completion Assistant** displays XSLT 1.0 functions (and optionally XSLT 2.0 / 3.0 functions) in the attributes *path*, *select*, *context*, *subject*, and *test*, depending on *the Schematron options* that are set in Preferences pages. If the Saxon 6.5.5 namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.6.0.7 namespace is declared in the Schematron schema (`xmlns:saxon="http://saxon.sf.net/"`) the content completion also displays the XSLT Saxon extension functions.

Highlight Quick Fix Occurrences in SQF

When you position your mouse cursor over a quick fix id in a Schematron document, Oxygen XML Editor plugin searches for the quick fix declaration and all its references and highlights them automatically.

Customizable colors are used: one for the quick fix definition and another one for its references. Occurrences are displayed until another quick fix is selected.

To change the default behavior of **Highlight Component Occurrences**, *open the **Preferences** dialog box* and go to **Editor** > **Mark Occurrences**. You can also trigger a search using the **Search** > **Search Occurrences in File (Ctrl Shift U (Meta Shift U on OS X))** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.

Searching and Refactoring Operations in SQF

Search Actions

The following search actions can be applied on quick fix ids and are available from the **Search** submenu in the contextual menu of the current editor:

- **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
- **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
- **Search Occurrences in File** - Searches all occurrences of the item at the cursor position in the currently edited file.

Refactoring Actions

The following refactoring actions can be applied on quick fix ids and are available from the **Refactoring** submenu in the contextual menu of the current editor:

- **Rename Component** - Allows you to rename the current component in-place. The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the cursor position are updated in real time to all occurrences of the component. To exit in-place editing, press the **Esc** or **Enter** key on your keyboard.
- **Rename Component in** - Opens the **Rename *component_type*** dialog box that allows you to rename the selected component by specifying the new component name and the files to be affected by the modification. If you click the **Preview** button, you can view the files affected by the **Rename Component** action.

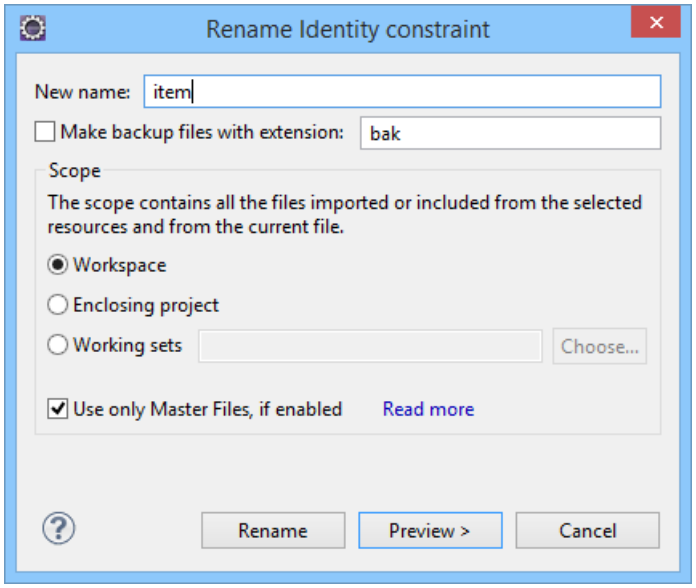


Figure 227: Rename Identity Constraint Dialog Box

Embed Schematron Quick Fixes in Relax NG or XML Schema

Schematron Quick Fixes can be embedded into a Relax NG or XML Schema within the Schematron rules from annotations (using the `appinfo` element), or in any Schematron rule of a RELAX NG Schema.

Oxygen XML Editor plugin is able to extract and use the embedded Schematron Quick Fixes. To make the Schematron Quick Fixes available, validate the document with both the RELAX NG schema and its embedded Schematron rules.

Editing XHTML Documents

XHTML documents with embedded CSS, JS, PHP, and JSP scripts are rendered with dedicated coloring schemes. To customize them, *open the **Preferences** dialog box* and go to **Editor** > **Syntax Highlight**.

Spell Checking

Oxygen XML Editor plugin includes an *automatic (as-you-type) spell checking feature*, as well as a manual spell checking action to opens a **Spelling** dialog box that offers a variety of options. To open this dialog box, use the **Check Spelling** action on the toolbar.

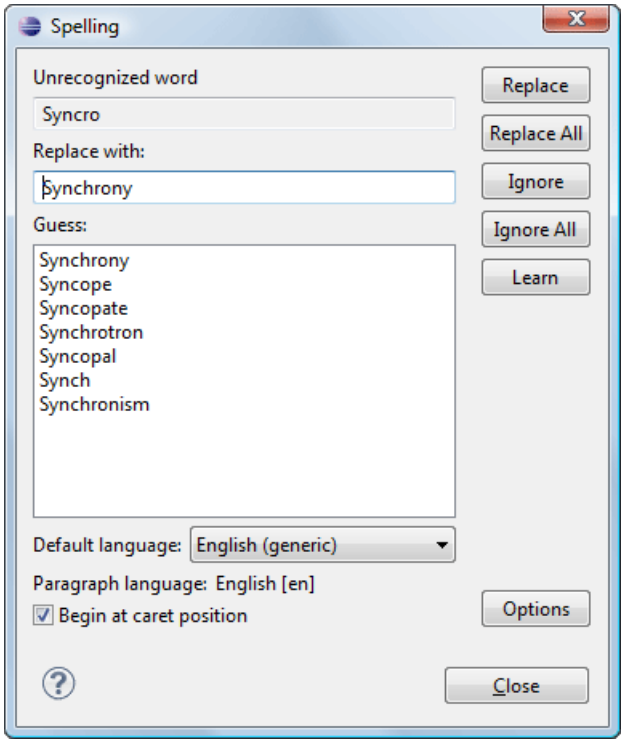


Figure 228: The Check Spelling Dialog Box

The **Spelling** dialog box contains the following fields:

- **Unrecognized word** - Contains the word that cannot be found in the selected dictionary. The word is also highlighted in the XML document.
- **Replace with** - The character string which is suggested to replace the unrecognized word.
- **Guess** - Displays a list of words suggested to replace the unknown word. Double click a word to automatically insert it in the document and resume the spell checking process.
- **Default language** - Allows you to select the default dictionary used by the spelling engine.
- **Paragraph language** - In an XML document you can mix content written in different languages. To tell the spell checker engine what language was used to write a specific section, you need to set the language code in the `lang` or `xml:lang` attribute to that section. Oxygen XML Editor plugin automatically detects such sections and instructs the spell checker engine to apply the appropriate dictionary.
- **Replace** - Replaces the currently highlighted word in the XML document, with the selected word in the**Replace with** field.
- **Replace All** - Replaces all occurrences of the currently highlighted word in the XML document, with the selected word in the**Replace with** field.

- **Ignore** - Ignores the first occurrence of the unrecognized word and allows you to continue checking the document.Oxygen XML Editor plugin skips the content of the XML elements *marked as ignorable*.
- **Ignore All** - Ignores all instances of the unknown word in the current document.
- **Learn** - Includes the unrecognized word in the list of valid words.
- **Options** - Sets the configuration options of the spell checker.
- **Begin at cursor position** - Instructs the spell checker to begin checking the document starting from the current cursor position.
- **Close** - Closes the dialog box.

Spell Checking Dictionaries

There are two spell checking engines available in Oxygen XML Editor plugin: **Hunspell** checker (default setting) and **Java** checker. You can set the spell check engine in the ***Spell checking engine*** preferences page. The dictionaries used by the two engines differ in format, so you need to follow specific procedures in order to add another dictionary to your installation of Oxygen XML Editor plugin.

Dictionaries for the Hunspell Checker

The Hunspell spell checker is open source and has LGPL license. The format of the Hunspell spell dictionary is supported by Mozilla, OpenOffice and the Chrome browser. Oxygen XML Editor plugin comes with the following built-in dictionaries for the Hunspell checker:

- English (US)
- English (UK)
- French
- German
- Spanish.

Each language-country variant combination has its specific dictionary. If you cannot find a Hunspell dictionary that is already built for your language, you can build the dictionary you need. To build a dictionary from this list follow *these instructions*.

Add Dictionaries and Term Lists for the Hunspell Checker

To add new spelling dictionaries to Oxygen XML Editor plugin, or to replace an existing one, follow these steps:

1. *Download the files* you need for your language dictionary.
2. The downloaded `.oxt` file is a *zip* archive. If you are creating a new dictionary, copy the `.aff` and `.dic` files from this archive in the `spell` subfolder of the Oxygen XML Editor plugin preferences folder.

   The Oxygen XML Editor plugin preferences folder is >, where `[APPLICATION-DATA-FOLDER]` is:

   - C:\Users\[USER-NAME]\AppData\Roaming on Windows Vista/7/8/10
   - `[USER-HOME-FOLDER]/Library/Preferences` on OS X
   - `[USER-HOME-FOLDER]` on Linux
3. If you are updating an existing dictionary, copy the `.aff` and `.dic` files into the folder `[OXYGEN_DIR]/dicts/spell`.
4. Restart the application after copying the dictionary files.

Note: You can specify that Oxygen XML Editor plugin includes dictionaries and term lists from a custom location by selecting the **Include dictionaries and term list from** option and specifying its path 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 Editor plugin comes with the following built-in dictionaries for the Java checker:

- English (US)

- English (UK)
- English (Canada)
- French (France)
- French (Belgium)
- French (Canada)
- French (Switzerland)
- German (old orthography)
- German (new orthography)
- Spanish

A pre-built dictionary can be added by copying the corresponding `.dar` file to the folder `[OXYGEN_DIR]/dicts` and restarting Oxygen XML Editor plugin. There is one dictionary for each language-country variant combination.

Learned Words

Spell checker engines rely on dictionary to decide that a word is correctly spelled. To tell the spell checker engine that an unknown word is actually correctly spelled, you need to add that word to its dictionary. There are two ways to do this:

- Press the **Learn** button from the ***Spelling*** *dialog box*.
- Invoke the contextual menu on an unknown word, then press **Learn word**.

Learned words are stored into a persistent dictionary file. Its name is composed of the currently checked language code and the `.tdi` extension, for example `en_US.tdi`. It is located in the:

- `[HOME_DIR]/AppData/Roaming/com.oxygenxml/spell` folder on Windows Vista/7/8/10.
- `[HOME_DIR]/Application Data/com.oxygenxml/spell` folder on Windows XP.
- `[HOME_DIR]/Library/Preferences/com.oxygenxml/spell` folder on OS X.
- `[user-home-folder]/com.oxygenxml/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 (Elements)

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 box*, or 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

Oxygen XML Editor plugin includes an option to automatically check the spelling as you type. This feature is disabled by default, but can be enabled and configured in the ***Spell Check*** *preferences page*. When the option is enabled, unknown words are highlighted and they offer a contextual menu that includes 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 following locations:

-
- The contextual menu of the **Navigator** 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.

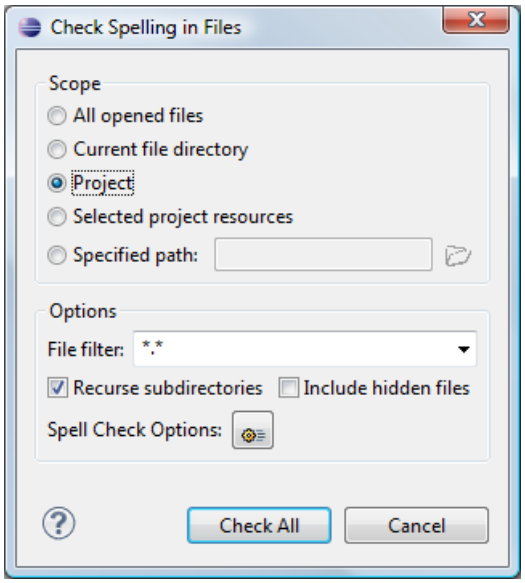


Figure 229: Check Spelling in Files Dialog Box

The following scopes are available:

- **All opened files** - The spell check is performed in all opened files.
- **Directory of the current file** - All the files in the folder of the current edited file.
- **Project files** - All files from the current project.
- **Selected project files** - The selected files from the current project.
- **Specified path** - Checks the spelling in the files located at a path that you specify.

The **Options** section includes the following options:

- **File filter** - Allow you to filter the files from the selected scope.
- **Recurse subdirectories** - When enabled, the spell check is performed recursively for the specified scope. The one exception is that this option is ignored if the scope is set to **All opened files**.
- **Include hidden files** - When enabled, the spell check is also performed in the hidden files.
- **Spell Check Options** - The spell check processor uses the options available in the ***Spell Check** preferences panel* .

When you invoke the **Check Spelling in Files** action in the **DITA Maps Manager** view, a different dialog box is displayed:

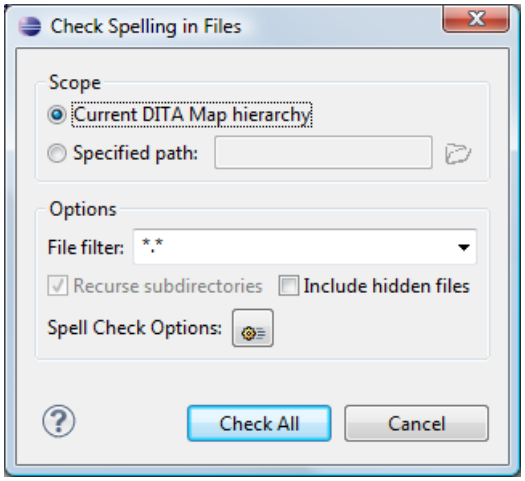


Figure 230: Check Spelling in Files Dialog Box (Invoked from the DITA Maps Manager View)

The following scopes are available:

- **Current DITA Map hierarchy** - All the files referenced in the currently selected DITA map, opened in the **DITA Maps Manager** view
- **Specified path** - checks the spelling in the files located at a path that you specify

AutoCorrect Misspelled Words

Oxygen XML Editor plugin includes an *AutoCorrect* feature to automatically correct misspelled words, as well as to insert certain symbols or other text, as you type in **Author** mode. Oxygen XML Editor plugin includes a default list of commonly misspelled words and symbols, but you can modify the list to suit your needs. You can also choose to have the *AutoCorrect* feature use suggestions from the main spell checker. The suggestions will only be used if the misspelled words are not found in the Replacements table.

When enabled, the *AutoCorrect* feature can be used to do the following:

- Automatically correct misspelled words while you edit in **Author** mode.
- Easily insert symbols. For example, if you want to insert a ® character, you would type `(R)`.
- Quickly insert text fragments.

AutoCorrect is enabled by default. To configure this feature, *open the **Preferences** dialog box* and go to **Editor** > **Edit Modes** > **Author** > **AutoCorrect**.

The actual operation of replacing a word is triggered by a space, dash, or punctuation mark (`, . ; : ? !`). After a correction, the affected string is highlighted. The highlight is removed upon the next editing action (text insertion or deletion). If you hover over the highlight, a small widget appears below the word. If you hover over the widget, it expands and you can click it to present a drop-down list that includes the following options:

- **Change back to "..."** - Reverts the correction back to its original form.
- **Stop Automatically Correcting "..."** - This option is presented if the correction is performed based on the *AutoCorrect Replacements Table* and selecting it will delete the corresponding entry from the Replacements Table.
- **Learn Word "..."** - This option is presented if the ***Use additional suggestions from the spell checker** option* is enabled in the *AutoCorrect preferences page* and the correction is performed based on the *Spell Checker*. Selecting this option will add the item to the list of *learned words*.
- **AutoCorrect options** - Opens the *AutoCorrect Preferences* on page 936 page that allows you to configure the feature.

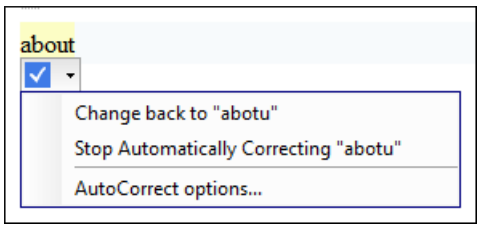


Figure 231: The AutoCorrect Widget

The *AutoCorrect* feature results in the following types of substitutions in regards to case-sensitivity:

- Words with all lower-case characters will be replaced with lower-case substitutions (for example, "abotu" is replaced with "about").
- Words with irregular-case characters will be replaced with lower-case substitutions ("ABotU" is replaced with "about").
- Words with all upper-case characters will be replaced with upper-case substitutions ("ABOTU" is replaced with "ABOUT").
- Words starting with an upper-case character will be replaced with substitutions having the same pattern ("Abotu" is replaced with "About").

The *AutoCorrect* feature also uses the list of ***ignored elements from the Spell Check preferences page***. All elements (along with their descendant elements) included in this list will be ignored by the *AutoCorrect* engine.

Add Dictionaries for the AutoCorrect Feature

To add new dictionaries for the *AutoCorrect* feature, or to replace an existing one, follow these steps:

1. *Download the files* you need for your language dictionary.
2. If you are creating a new dictionary, copy the downloaded `.dat` files to the `autocorrect` subfolder of the Oxygen XML Editor plugin preferences folder.

   The Oxygen XML Editor plugin preferences folder is >, where `[APPLICATION-DATA-FOLDER]` is:

   - C:\Users\[USER-NAME]\AppData\Roaming on Windows Vista/7/8/10
   - `[USER-HOME-FOLDER]/Library/Preferences` on OS X
   - `[USER-HOME-FOLDER]` on Linux

3. If you are updating an existing dictionary, copy the `.dat` file to the folder `[OXYGEN_DIR]/dicts/autocorrect`.
4. Restart the application after copying the dictionary files.

Note: You can setup Oxygen XML Editor plugin to use dictionaries from a custom location configured in *the **Dictionaries** preferences page*.

Handling Read-Only Files

The default workbench behavior applies when editing read-only files in the **Text** mode. For all other modes no modification is allowed provided that the file remains read-only.

You can check out the read-only state of the file by looking in the *Properties view*. If you modify the file properties from the operating system and the file becomes writable, you are able to modify it on the spot without having to reopen it.

Associating a File Extension with Oxygen XML Editor plugin

To associate a file extension with Oxygen XML Editor plugin on Windows:

- Go to the **Start** menu and click **Control Panel**.
- Go to **Default Programs**.
- Click **Associate a file type or protocol with a program**.
- Click the file extension you want to associate with Oxygen XML Editor plugin, then click **Change program**.
- In the **Open With** dialog box, click **Browse** and select Oxygen XML Editor plugin.

To associate a file extension with Oxygen XML Editor plugin on Mac OS:

- 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 Editor plugin.
- With Oxygen XML Editor plugin selected, click **Change All**.

Chapter 7

DITA Authoring

Topics:

DITA is an XML standard, an architectural approach, and a writing methodology, developed by technical communicators for technical communicators. It provides a standardised architectural framework for a common structure for content that promotes the consistent creation, sharing, and re-use of content.

Some of the benefits of using DITA include the following:

- **Flexibility** - DITA is a topic-based architecture and it offers flexibility in content organization.
- **Modularity** - DITA allows for content reuse that saves time and reduces the number of modifications.
- **Structured Authoring** - DITA offers a standardized, methodological approach that helps to reduce authoring time and improve consistency.
- **Single-Source Publishing** - DITA provides the ability to change content in one place and have the change propagate everywhere.
- **Multiple Output Formats** - DITA supports multiple types of output.
- **Inheritance** - The DITA inheritance model makes it easy to specialize topics or elements within topics and you only have to define how the element is different from its immediate ancestor.
- **Process Automation** - DITA offers various ways to automate processes, such as with index or glossary production, output delivery, validation, and more.
- **Specialization** - DITA allows you to define your own information types and semantic elements/attributes to suit the needs of your particular content model.
- **Multi-Lingual** - DITA is a translation-friendly structure that supports numerous languages and text encodings.
- **Conditional Profiling** - DITA supports conditional text processing and profiling to filter content in the publishing stage.

This chapter is designed to be a guide to help content authors who use DITA. It also presents the Oxygen XML Editor plugin features that are specific to working with DITA documents and concepts.

Working with DITA Maps

In the DITA standard architecture you create documents by collecting topics into maps.

DITA Maps

A DITA map organizes a set of topics into a hierarchy. In most output formats, the structure of the map becomes the structure of the table of contents. Oxygen XML Editor plugin provides support for *creating* and *managing DITA maps* through the ***DITA Maps Manager***. There are also specialized types of DITA maps, such as a *book map*, which is intended for creating the structure of a book.

Sub-Maps

You do not have to create an entire publication using a single map. It is generally good practice to break up a large publication into several smaller *sub-maps* that are easier to manage. You can reuse sub-maps in a variety of different publications by including them in each of the main maps. The ***DITA Maps Manager*** provides support for easily creating and managing sub-maps.

Chunking DITA Maps

By default, many output types place a single topic on each output page. In some cases you may want to *output multiple topics as a single output page (also known as chunking)*. To support this, Oxygen XML Editor plugin provides an ***Edit Properties** dialog box* that allows you to easily configure the attributes of a topic to control how your table of contents and topics are rendered in the output.

Validating a Map

You should *validate your maps* to make sure that the individual topics are valid and that the relationships between them are working. Oxygen XML Editor plugin provides a validation function for DITA maps that performs a comprehensive validation of a map and its topics.

Opening a Map

There are several ways to open a DITA map and you can choose to open it in the **DITA Maps Manager** or in the XML editor. Use any of the following methods to open a map:

- To open a sub-map in its own tab in the **DITA Maps Manager**, simply double click it (or right-click it and select **Open**).
- To open a map in the XML editor from the **DITA Maps Manager**, right-click it and select **Open Map in Editor**.
- To open a map in the **DITA Maps Manager**, you can right click a map file in the **Navigator** view and select **Open in DITA Maps Manager**.

To watch a video on DITA editing , go to *http://oxygenxml.com/demo/DITA_Editing.html*.

DITA Maps Manager

Oxygen XML Editor plugin 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 the topics and maps, make changes, and apply transformation scenarios to obtain various output formats.

The **DITA Maps Manager** includes a variety of useful actions to help you edit and organize the structure of your DITA maps and topics. The actions that are available and their functions depend on the type of nodes that are selected in the **DITA Maps Manager**. If you select multiple sibling nodes, the result of the actions will be applied to all the selected nodes. If you select multiple nodes that are not on the same hierarchical level, the actions will be applied to the parent node and the child nodes will inherit certain attributes from the parent node.

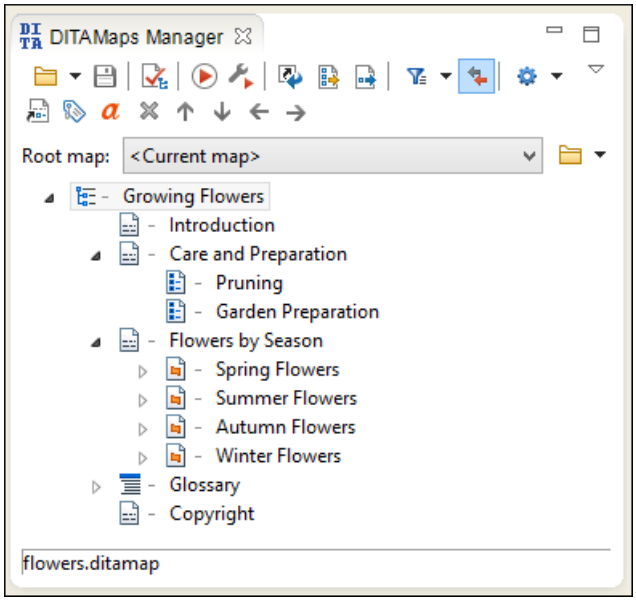


Figure 232: The DITA Maps Manager View

Opening Maps in the DITA Maps Manager

The **DITA Maps Manager** view supports multiple open maps at the same time, with each one presented in its own tab. To open a DITA map in the **DITA Maps Manager**, use any of the following methods:

- To open a sub-map in its own tab, simply double click it (or right-click it and select **Open**).
- Right click a map file in the **Navigator** view and select **Open in DITA Maps Manager**.

If your map references other DITA maps, they will be shown, expanded, in the **DITA Maps Manager** view and you will be able to navigate their content. To edit the sub-maps and their content, you need to open each referenced map separately.

Note: You can choose to not expand referenced maps in the **DITA Maps Manager** view by disabling the **Display referenced content** option in the *Author preferences page*.

Drag and Drop in the DITA Maps Manager

You can move topics or nodes in the same map, or between different maps, by dragging and dropping them into the desired position. You can arrange the nodes by dragging and dropping one or more nodes at a time. You can arrange multiple topics by dragging them while pressing the **Ctrl** or **Shift** key. Drop operations can be performed before, after, or as child of the targeted node.

Drag and drop operations include:

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. The mouse pointer changes to a ⊞ symbol to indicate that a copy operation is being 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 by using the **Promote** (**Alt Left Arrow**) and **Demote** (**Alt Right Arrow**) operations.

DITA Maps Manager Toolbar

The toolbar includes the following actions (also available in the**DITA Maps** menu) and their availability depend on the nodes that are selected:

Note: If multiple nodes are selected, the availability of the actions depend on the nodes that are selected.

Open Drop-down Menu

You can use this drop-down menu to open new DITA maps or to reopen recently viewed maps. The drop-down menu contains the following:

- List of recently viewed DITA maps that can be selected to reopen them.
- **Clear history** - Clears the history list of the recently viewed DITA maps.
- **Open** - Allows you to open the map in the **DITA Maps Manager** view. You can also open a map by dragging it from the file system explorer and dropping it into the **DITA Maps Manager** view.
- **Browse workspace** - Opens a file browser dialog box allowing you to select a file from the local workspace.
- **Open URL** - Displays the **Choose DITA Map** dialog box that allows you to access any resource identified through a URL (defined by a protocol, host, resource path, and an optional port). The following actions are available in this drop-down menu:
  - **Browse for local file** - Opens a local file browser dialog box, allowing you to select a local DITA map.
  - **Browse workspace** - Opens a file browser dialog box allowing you to select a file from the local workspace.
  - **Browse for remote file** - Displays *the Open using FTP/SFTP dialog box* that allows you to open a remotely stored DITA map.
  - **Browse for archived file** - Displays the **Archive Browser** dialog box that allows you to browse the content of an archive and choose a DITA map.
  - **Browse Data Source Explorer** - Opens the **Data Source Explorer** that allows you to browse the data sources defined in the *Data Sources preferences page*.

    Tip: You can open the **Data Sources** preferences page by using the **Configure Database Sources** shortcut from the **Open URL** dialog box.

  - **Search for file** - Displays the **Find Resource** dialog box that allows you to search for a DITA map.

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.

Configure Transformation Scenario(s)

Allows you to *associate 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 referenced documents. This action is useful when the referenced documents are modified externally. When they are modified and saved from the Oxygen XML Editor plugin, 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.

Tip: If you want to print the expanded content, you should consider changing the **Styles** drop-down to **+ Print ready**.

Open Map in Editor

For complex operations that cannot be performed in the simplified **DITA Maps Manager** view (for instance, 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**.

Profiling/Conditional Text Drop-down Menu

This drop-down menu contains the following actions:

- **Show Profiling Colors and Styles** - Enable this option to turn on conditional styling. To configure the colors and styles *open the **Preferences** dialog box* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text** > **Colors and Styles**.
- **Show Profiling Attributes** - Enable this options to display 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 in both the **DITA Maps Manager** view and in the **Author** view.
- **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 enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.
- **Profiling Settings** - Opens the preferences page for adding and editing the profiling conditions that you can apply in the **DITA Maps Manager** view and the **Author** view. When a profiling condition set is applied, the keys that are defined in the DITA map are gathered by filtering out the excluded content.

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.

Settings

Allows you to choose whether or not to **Show extended toolbar** and **Show root map toolbar**.

Root map

Specifies a master DITA map that Oxygen XML Editor plugin uses to establish a *key space* that you can use with any other DITA map that is contained by the master map.

Browse Drop-down menu

You can use this drop-down menu to browse for root maps with the following choices:

- **Browse for local file** - Opens a local file browser dialog box, allowing you to select a local root map.
- **Browse workspace** - Allows you to select a root map from the local workspace.
- **Browse for remote file** - Displays *the Open using FTP/SFTP dialog box* that allows you to select a remotely stored root map.
- **Browse for archived file** - Displays the **Archive Browser** dialog box that allows you to browse the content of an archive and choose a root map.

- **Browse Data Source Explorer** - Opens the **Data Source Explorer** that allows you to browse the data sources defined in the *Data Sources preferences page*.

  Tip: You can open the **Data Sources** preferences page by using the **Configure Database Sources** shortcut from the **Open URL** dialog box.

- **Search for file** - Displays the **Find Resource** dialog box to search for a root map.

Insert Topic Reference

Opens *the **Insert Reference** dialog box* that allows you to insert references to targets such as anchors, topics, maps, topic sets, or key definitions.

Edit Properties

Opens the **Edit Properties** dialog box that allows you to configure the properties of a selected node. You can find more details about this dialog box in the *Edit Properties in DITA Maps* on page 423 topic.

Edit Attributes

Opens a small in-place editor that allows you to edit the attributes of a selected node. You can find more details about this action in the *Attributes View in Author Mode* on page 88 topic.

Delete

Deletes the selected node.

Move Up

Moves the selected node up within the DITA map tree.

Move Down

Moves the selected node down within the DITA map tree.

Promote

Moves the selected node up one level to the level of its parent node.

Demote

Moves the selected node down one level to the level of its child nodes.

Contextual Menu of the DITA Maps Manager

The following actions can be invoked from the contextual menu on the *root map* of an opened DITA map:

Open Map in Editor

For complex operations that cannot be performed in the simplified **DITA Maps Manager** view (for instance, editing a relationship table) you can open the map in the main editing area.

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.

Export DITA Map

Allows you to choose a destination for exporting the DITA map.

Find Unreferenced Resources

Allows you to search for orphaned resources that are not referenced in the DITA maps.

Edit Properties

Opens the **Edit Properties** dialog box that allows you to configure the properties of a selected node. You can find more details about this dialog box in the *Edit Properties in DITA Maps* on page 423 topic.

Append Child submenu

Container sub-menu for a number of actions that create a map node as a child of the currently selected node:

- **New** - Opens a dialog box that allows you to configure some options for *inserting a new topic*.
- **Reference** - Inserts a reference to a topic file. You can find more details about this action in the *Inserting References* topic.
- **Reference to the currently edited file** - Inserts a reference to the currently edited file. You can find more details about this action in the *Inserting References* topic.
- **Key Reference** - Opens an ***Insert Key Definition*** *dialog box* that allows you to insert a key reference.
- **Key Reference with Keyword** - Opens a simplified ***Insert Key Definition*** *dialog box* that allows you to define a key and a value inside a *keyword*.
- A set of actions that open the ***Insert Reference*** *dialog box* that allow you to insert various reference specializations (such as **Anchor Reference**, **Glossary Reference**, **Map Reference**, **Navigation Reference**, **Topic Group**, **Topic Head**, **Topic Reference**, **Topic Set**, **Topic Set Reference**).

Search References

Searches all references to the current topic in the entire DITA map.

Refactoring submenu

The following actions are available from this submenu:

Rename resource

Allows you to *change the name of a resource linked in the edited DITA map*.

Move resource

Allows you to *change the location on disk of a resource linked in the edited DITA map*.

XML Refactoring

Opens *the XML Refactoring tool wizard* that presents refactoring operations to assist you with managing the structure of your XML documents.

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*.

Paste

Allows you to paste content from the clipboard into 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.

Expand All

Allows you to expand the entire DITA map structure.

Collapse All

Allows you to collapse the entire DITA map structure.

The following actions are available when the contextual menu is invoked from a node that is not an immediate child node of the *root map*:

Note: If multiple nodes are selected, the availability of the actions depend on the nodes that are selected.

Open

Opens in the editor the resources referenced by the nodes that you select.

Open Map in Editor (available when invoking on a sub-map)

Opens the currently selected DITA map in the editor.

Open parent DITA map (available when invoking on a topic reference or a sub-map reference)

Opens the parent DITA map of the currently selected reference in the **DITA Maps Manager**.

Edit Attributes (only available for relationship table nodes)

Opens a small in-place editor that allows you to edit the attributes of a selected node. You can find more details about this action in the *Attributes View in Author Mode* on page 88 topic.

Edit Profiling Attributes (only available for relationship table nodes)

Allows you to change the *profiling attributes* defined on the selected node.

Search References

Searches all references to the current topic in the entire DITA map.

Refactoring submenu

The following actions are available from this submenu:

Rename resource

Allows you to *change the name of a resource linked in the edited DITA map*.

Move resource

Allows you to *change the location on disk of a resource linked in the edited DITA map*.

XML Refactoring

Opens *the XML Refactoring tool wizard* that presents refactoring operations to assist you with managing the structure of your XML documents.

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*.

Copy

Copies the currently selected node to the clipboard.

Expand All

Allows you to expand the entire DITA map structure.

Collapse All

Allows you to collapse the entire DITA map structure.

The following actions are available when the contextual menu is invoked on a child node of a DITA map (sub-maps need to be opened in the **DITA Maps Manager** to access these actions since they are in a read-only state in the parent map):

Note: If multiple nodes are selected, the availability of the actions depend on the nodes that are selected.

Open

Opens in the editor the resources referenced by the nodes that you select.

Edit Properties

Opens the **Edit Properties** dialog box that allows you to configure the properties of a selected node. You can find more details about this dialog box in the *Edit Properties in DITA Maps* on page 423 topic.

Append Child submenu

Container sub-menu for a number of actions that create a map node as a child of the currently selected node:

- **New** - Opens a dialog box that allows you to configure some options for *inserting a new topic*.
- **Reference** - Inserts a reference to a topic file. You can find more details about this action in the *Inserting References* topic.
- **Reference to the currently edited file** - Inserts a reference to the currently edited file. You can find more details about this action in the *Inserting References* topic.
- **Key Reference** - Opens an ***Insert Key Definition*** *dialog box* that allows you to insert a key reference.
- **Key Reference with Keyword** - Opens a simplified ***Insert Key Definition*** *dialog box* that allows you to define a key and a value inside a *keyword*.
- A set of actions that open the ***Insert Reference*** *dialog box* that allow you to insert various reference specializations (such as **Anchor Reference**, **Glossary Reference**, **Map Reference**, **Navigation Reference**, **Topic Group**, **Topic Head**, **Topic Reference**, **Topic Set**, **Topic Set Reference**).

Insert After submenu

Container sub-menus for a number of actions that create a map node as a sibling of the currently selected node:

- **New** - Opens a dialog box that allows you to configure some options for *inserting a new topic*.
- **Reference** - Inserts a reference to a topic file. You can find more details about this action in the *Inserting References* topic.
- **Reference to the currently edited file** - Inserts a reference to the currently edited file. You can find more details about this action in the *Inserting References* topic.
- **Key Reference** - Opens an ***Insert Key Definition*** *dialog box* that allows you to insert a key reference.
- **Key Reference with Keyword** - Opens a simplified ***Insert Key Definition*** *dialog box* that allows you to define a key and a value inside a *keyword*.
- A set of actions that open the ***Insert Reference*** *dialog box* that allow you to insert various reference specializations (such as **Anchor Reference**, **Glossary Reference**, **Map Reference**, **Navigation Reference**, **Topic Group**, **Topic Head**, **Topic Reference**, **Topic Set**, **Topic Set Reference**).

Search References

Searches all references to the current topic in the entire DITA map.

Refactoring submenu

The following actions are available from this submenu:

Rename resource

Allows you to *change the name of a resource linked in the edited DITA map*.

Move resource

Allows you to *change the location on disk of a resource linked in the edited DITA map*.

XML Refactoring

Opens *the XML Refactoring tool wizard* that presents refactoring operations to assist you with managing the structure of your XML documents.

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

Deletes the currently selected node and copies it to the clipboard.

Copy

Copies the currently selected node to the clipboard.

Paste

Allows you to paste content from the clipboard into 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 currently selected node from the DITA map.

Organize

Allows you to organize the DITA map with the several submenu actions:

- **Move Up** - Moves the selected node up within the DITA map tree.
- **Move Down** - Moves the selected node down within the DITA map tree.
- **Promote** - Moves the selected node up one level to the level of its parent node.
- **Demote** - Moves the selected node down one level to the level of its child nodes.

Expand All

Allows you to expand the entire DITA map structure.

Collapse All

Allows you to collapse the entire DITA map structure.

To watch our video demonstration about the **DITA Maps Manager** view, go to *http://oxygenxml.com/demo/DITA_Maps_Manager.html*.

Creating a Map

To create a *DITA map*, Subject scheme, bookmap, or other types of DITA maps, follow these steps:

1. Go to **File** > **New** > **New from Templates**.
   A **New** document dialog box is opened that allows you to select a document type from various folders.
2. Select one of the **DITA Map** templates from the **Framework templates** folder.
3. Click the **Next** button.
4. Select a parent folder and the file name and click **Finish**.
5. Save the map after opening it in the **DITA Maps Manager** or the Editor.

Selecting a Root Map

Oxygen XML Editor plugin allows you to select a DITA map as a *key space*, or *root map*, for all the other DITA maps and topics in the project. Specifying the correct *root map* helps to prevent validation problems when you work with `keyrefs` and also acts as the foundation for content completion. All the *keys* that are defined in a *root map* are available in the maps that the *root map* contains.

There are several ways to select or change the *root map*:

- The easiest method is to use the **Root map** drop-down menu in the **DITA Maps Manager** toolbar to select the appropriate *root map*.
- If you insert a *key reference* using the **Cross Reference** action from the **Link** drop-down menu (from the toolbar or **Link** submenu of the contextual menu) and keys are not gathered from the expected DITA map, you can change the root map by using the **Change Root Map** link in the **Choose Key** dialog box that is opened when you click the **Choose Key Reference** button.
- If you insert a *content key reference* or *key reference* using the **Reuse Content** action (from the toolbar, **DITA** menu, or **Reuse** submenu of the contextual menu) and keys are not gathered from the expected DITA map, you can

change the root map by using the **Change Root Map** link in the **Choose Key** dialog box that is opened when you click the  **Choose Key Reference** button.

To watch our video demonstration about the DITA Root Map support, go to *http://oxygenxml.com/demo/DITA_Root_Map.html*.

Creating DITA Sub-Maps

You can break up a large DITA map into more manageable pieces by creating sub-maps. A sub-map is simply a DITA map that is included by another DITA map. There is no separate markup for a sub-map.

For example, if you are creating a book, you might use one sub-map for each chapter of the book. If you are reusing a set of topics in multiple publications, you might collect them into a map and reuse the map as a sub-map in multiple other maps, rather than referencing the topics individually from the new maps.

You add a sub-map to a map the same way that you would *add a new topic or insert an exiting topic into a map*, except you choose a map rather than a topic to create or add. When adding a sub-map, to a map, make sure that you use a `mapref` element or a `topicref` element with the `format` attribute set to `ditamap`. In most cases, Oxygen XML Editor plugin takes care of this for you.

You can move sub-maps around in a map *just as you would move a topic*.

Creating a Book Map in DITA

If you want to create a traditional book in DITA, you can use a book map to organize your topics into a book. A DITA book map is a specialized type of map, intended for creating output that is structured like a book. A book map allows you to add book-specific elements such as `frontmatter`, `part`, `chapter`, `appendix`, and `backmatter` to the map. How these book-specific elements are processed for publication is up to the processing script for each media. See the *DITA documentation* for details.

You can find additional support for creating books in DITA in the DITA for Publishers plugin, which is included with .

To create a book in DITA using a book map:

1. Create a new book map. **File** > **New** > **Framework templates** > **DITA Map** > **map** > **Bookmap**.
2. Create the structure of your book by adding the appropriate book sections and defining containers for chapters and any appendices. To add sections to a book map, or children to a section, right-click the book map or section icon and choose **Append child**. The selections offered in the **Append child** and **Insert After** menus will adjust depending on the element they are applied to. Consult the DITA documentation to fully understand the structure of a DITA book map and where to create each element.
3. Create special elements like an *index* and a *table of contents*. The index and table of contents will be generated by the build process, based on the content of the map and the topics it points to.
4. *Add topics* to your chapters to add content to your book. You may find it easier to manage if you *use sub-maps* to create the content of your chapters. This keeps your book map from becoming long and difficult to manage.

Managing DITA Maps

You may want to manage your DITA maps in a variety of ways, such as:

- Change the order and nesting of topics in a map.
- Add or remove topics from the map.
- Add keys to the topics in the map.
- Add profiling (conditions) to the items in the map.
- Change other properties of the items in the map.

Changing the Order and Nesting of Topics in a Map

You can change the order and nesting of the topics in a map in several ways:

- By dragging and dropping topics within the **DITA Maps Manager**.

- By showing the extended **DITA Maps Manager** toolbar (press the settings icon on the **DITA Maps Manager** toolbar and select the extended toolbar) and then using the arrow keys ( ) on the toolbar to move topics around in the map.

Add and Remove Topics from a Map

You can *add* or remove topics from a map in a number of ways. Some ways to remove a topic from a map include:

- Highlight the topic and press **Delete**.
- Highlight the topic and click the **Delete** button on the **DITA Maps Manager** extended toolbar.

Add Keys to Topic in a Map

You can *add keys to topics* in a map by right-clicking an item in the map and choosing **Edit Properties**.

Profiling Sections of a Map

You can profile (make conditional) any section of a map by *adding one or more profiling attributes* to the topics in a map. In the hierarchical structure of a map, any profiling applied to a topic that has children applies to those children as well. You cannot include a child topic if its parent topic is excluded.

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.

Adding Topics to a DITA Map

When you are working in DITA, there are several approaches that you can use to create topics and maps. You can start by first creating topics and then assembling your finished topics into one or more documents by creating one or more maps, or you can start by creating a map and then adding new topics to it as you work.

The topics-first approach is generally more appropriate if you intend to do a lot of content reuse, as it encourages you to think of each topic as an independent unit that can be combined with other topics in various ways. The map-first approach will be more familiar to you if you are used to creating books or manuals as a whole. Oxygen XML Editor plugin supports both approaches.

A DITA map organizes content hierarchically, so you can add a topic as a child of the map root or as a child or sibling of any item already in the map. Therefore, the first step to adding a topic to a map is always to choose the place it will be inserted into the map.

Adding Existing Topics to a Map

At the XML-level, a topic is added to a map by adding a reference to the map that points to the topic. There are a variety of reference types that you can use. The default type is the `topicref` element. See the *DITA documentation* for the full range of reference elements and their uses. Oxygen XML Editor plugin provides several tools for inserting reference elements into a map:

Using the Insert Reference Dialog Box

The ***Insert Reference*** *dialog box* allows you to create various reference types and configure the most commonly used attributes. You can open the **Insert Reference** dialog box with any of the following methods:

- Right-click an item in the current map where you want to add the reference, select **Append child** or **Insert After** and select the type of reference to enter.
- If the topic you want to add is currently open in the editor, you can right-click an item in the current map where you want to add the reference and select **Reference to the currently edited file**.
- Selecting an item in the map and click the **Insert Reference** button from the **DITA Maps Manager** toolbar.
- Select **Insert Reference** from the **DITA Maps** menu.

Dragging and Dropping a File into the DITA Maps Manager

You can add a topic to a DITA map by dragging and dropping the file into the **DITA Maps Manager**. You can drag and drop files from any of the following:

- Your OS file system explorer
- The **Project** view

Adding topics this way will not open the **Insert Reference** dialog box, but you can adjust all the same properties by invoking the contextual menu from the topic and selecting **Edit Properties**.

Adding a New Topic to a Map

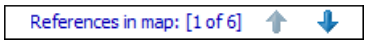
To add a new topic to a map:

1. Right-click the place in the current map where you want to add the new topic.
2. To insert the topic as a child of the selected node, select **Append Child** > **New**. To insert the topic as a sibling to the current node, select **Insert After** > **New**.

   This opens a ***New*** *file template dialog box* that allows you to select the type of document and assists you with naming it.

Adding Multiple References to the Same Topic in a Map

Oxygen XML Editor plugin allows you to add multiple references to the same topic in a DITA map. Whenever multiple references to the same topic are detected, an indicator will appear in the top-right corner of the **Author** mode editor that shows the number of times the topic is referenced in the DITA map. It also includes navigation arrows that allow you to jump to the next or previous reference.



Moving and Renaming Resources

You can move or rename resources on disk directly from Oxygen XML Editor plugin. To do this, use one of the following actions available in the **Refactoring** submenu of the contextual menu when invoked on a resource in the **DITA Maps Manager** view:

Refactoring > Rename resource

This action allows you to change the name of a resource linked in the edited DITA map, using the **Rename resource** dialog box. This dialog box contains the following options:

- **New name** - Presents the current name and allows you to change it.
- **Update references** - Enable this checkbox to update all references of the file in the edited DITA map and in the files referenced from the DITA map, preserving the *completeness* of the DITA map.
- **Preview** - Select this button to display a preview of the changes Oxygen XML Editor plugin is about to make.
- **Rename** - Executes the **Rename resource** operation.
- **Cancel** - Cancels the **Rename resource** operation. No changes are applied.

Refactoring > Move resource

This action allows you to change the location of a resource linked in the edited DITA map, using the **Move resource** dialog box. This dialog box contains the following options:

- **Destination** - Specifies the target location 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 referenced from the DITA map, preserving the *completeness* of the DITA map.
- **Preview** - Select this button to display a preview of the changes Oxygen XML Editor plugin 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.

Insert References in DITA Maps

A DITA map may contain various types of references. The targets of the references can be a variety of different references, such as anchors, chapters, maps, topics, topic sets, or key definitions. You can insert references to such targets with the ***Insert Reference** dialog box*.

This section explains how to insert and configure references (such as topic references, topic groups, topic headings, and key definitions) in a DITA map.

Insert Reference Dialog Box

The **Insert Reference** dialog box allows you to insert and configure references in DITA maps. There are numerous types of references that can be inserted into maps. They include references to topics, other maps, anchors, glossary terms, and keys. You can also use this dialog box to configure the attributes of a reference, add profiling or metadata, and define keys.

To open the **Insert Reference** dialog box, use one of the following methods:

- Select  **Reference**, **Reference to the currently edited file**, or any of the other specific reference actions that are available from the **Append Child** and **Insert After** submenus when invoking the contextual menu in the **DITA Maps Manager**.
  - To insert the reference as a child of the current node, select the reference from the **Append Child** submenu.
  - To insert the reference as a sibling of the current node (below the current node in the map), select the reference from the **Insert After** submenu.

  Note: The content of these submenus depends on the node that is selected in the DITA map tree when the contextual menu is invoked. For example, if the selected node is a topic reference (`topicref`), its possible child nodes include the following elements: `anchorref`, `chapter`, `keydef`, `mapref`, `topicgroup`, `topichead`, `topicref`, `topicset`, and `topicsetref`.

- Click the  **Insert Reference** button on the **DITA Maps Manager** extended toolbar. This action will insert the reference as a sibling of the current node (below the current node in the map).
- Select  **Insert Reference** from the **DITA Maps** menu. This action will insert the reference as a sibling of the current node (below the current node in the map).

For the  **Reference** or **Reference to the currently edited file** actions, a **Reference type** drop-down list is displayed at the top of the **Insert Reference** dialog box and you can select the type of reference you want to insert. Depending on the place where the reference will be inserted, Oxygen XML Editor plugin will propose only valid reference types . When you change the reference type, the fields in the various tabs of the dialog box are reconfigured depending upon the availability of the associated attributes. For the other reference actions in the **Append Child** and **Insert After** submenus, the reference type is automatically chosen based upon the invoked action and you cannot change it.

The main section of the dialog box includes the following tabs: **Target**, **Keys**, **Attributes**, **Metadata**, and **Profiling**.

Target Tab

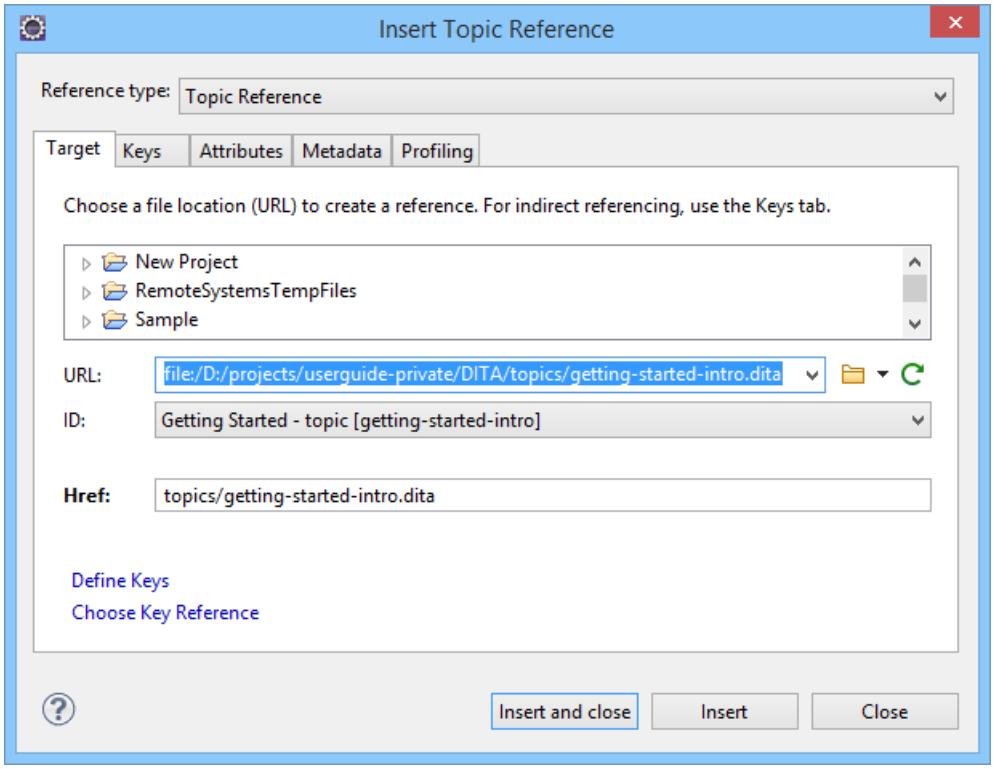


Figure 233: Insert Reference Dialog Box - Target Tab

The **Target** tab of the **Insert Reference** dialog box allows you to specify information about the target reference. It includes the following sections and fields:

Choose a file location section

You can browse for and select the source target file by using the file window in this section.

URL

Displays the path to the target and allows you to select or change it by using the combo box or browsing tools.

ID

The drop-down list displays all of the targets that are available for the selected target.

Href

The selected target automatically modifies this value to point to the corresponding `href` attribute of the target element.

Note: If the **Reference type** is a **Navigation Reference**, the **Href** field is changed to **Mapref**, since a `navref` element requires a `mapref` attribute instead.

Keys Tab

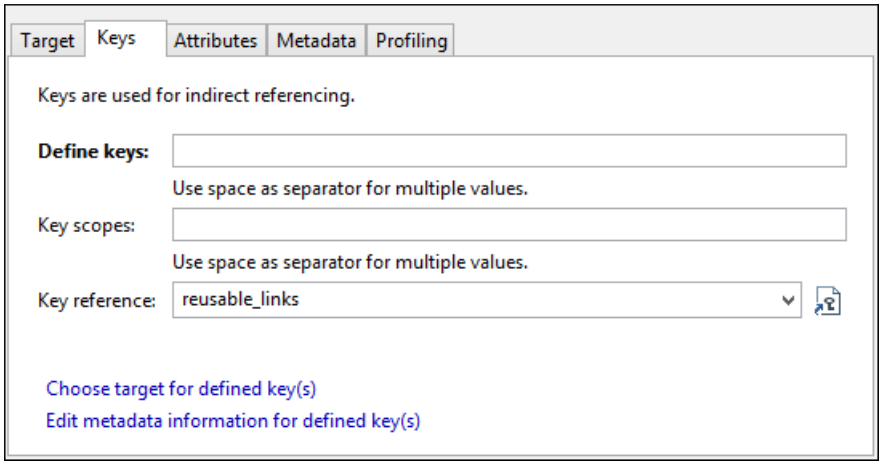


Figure 234: Insert Reference Dialog Box - Keys Tab

The **Keys** tab allows you to use and *define keys* for indirect referencing. For more information, see the *Working with Keys* on page 459 topic. This tab includes the following:

Define keys

Use this text field to define the `keys` attribute for the target.

Key scopes [This option is only available if the Built-in DITA-OT 2.x (with experimental DITA 1.3 support) option is enabled in the *DITA preferences page*]

Use this text field to define or edit the value of a *`keyscope` attribute*. *Key scopes* allow you to specify different sets of key definitions for different map branches.

Key reference

Instead of using the **Target** tab to select a file that contains the target reference, you can reference a key definition by using this text field. Use the **Choose key reference** button to access the list of keys that are already defined in the current root map.

Attributes Tab

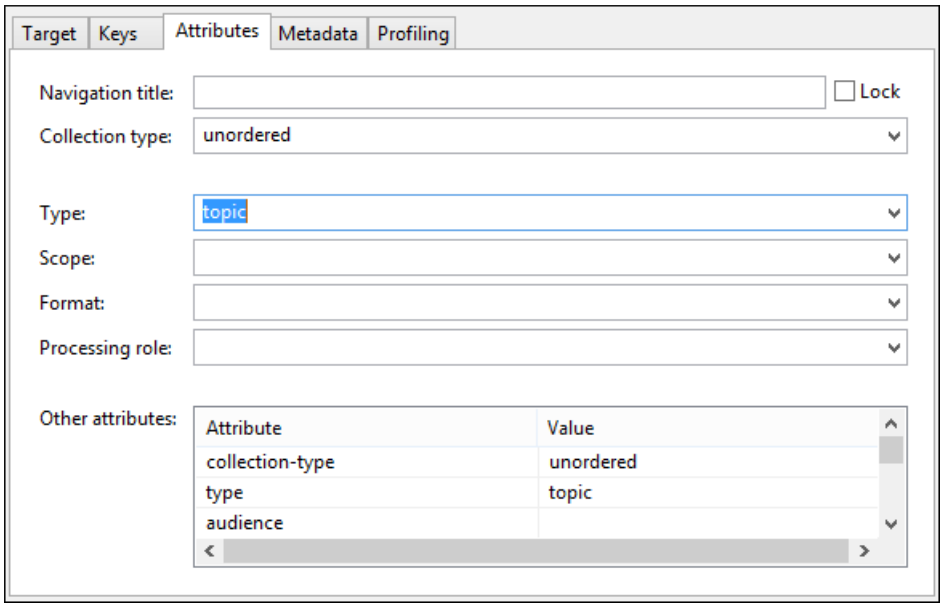


Figure 235: Insert Reference Dialog Box - Attributes Tab

The **Attributes** tab of the **Insert Reference** dialog box allows you to insert and edit attribute values for the target reference. This tab includes the following sections and actions:

Navigation title

This text field allows you to specify a custom navigation title for the target reference. You can enforce the use of the specified title by enabling the **Lock** checkbox.

Collection type

This drop-down list allows you to select the `collection-type` attribute to create hierarchical linking between topics in a DITA map (for example, `unordered`, `sequence`, `choice`, `family`, `-dita-use-conref-target`).

Type

Allows you to select a `type` attribute (such as `topic`, `task`, `concept`, etc.) for the target element. If you want this attribute to always be populated with a detected value (based on the specifications for the target file), enable the **Type** checkbox for the **Always fill values for attributes** option in the *DITA preferences page*.

Scope

This property corresponds to the `scope` attribute of the target element. It is populated automatically, based on the selected file type, unless its value for the selected target file is the same as the default attribute value. If you want this attribute to always be populated with a detected value (based on the specifications), enable the **Scope** checkbox for the **Always fill values for attributes** option in the *DITA preferences page*.

Format

This property corresponds to the `format` attribute of the target element. It is populated automatically, based on the selected file type, unless its value for the selected target file is the same as the default attribute value. If you want this attribute to always be populated with a detected value (based on the specifications), enable the **Format** checkbox for the **Always fill values for attributes** option in the *DITA preferences page*.

Processing Role

This drop-down list allows you to set the `processing-role` attribute to one of the allowed values for DITA reference elements (for example, `resource-only`, `normal`, `-dita-use-conref-target`).

Other attributes table

This table contains the attributes that are available for the selected reference. You can use this table to insert or edit the values of any of the listed attributes. Clicking a cell in the **Value** column allows you to use the combo box to enter, edit, or select attribute values.

Metadata Tab

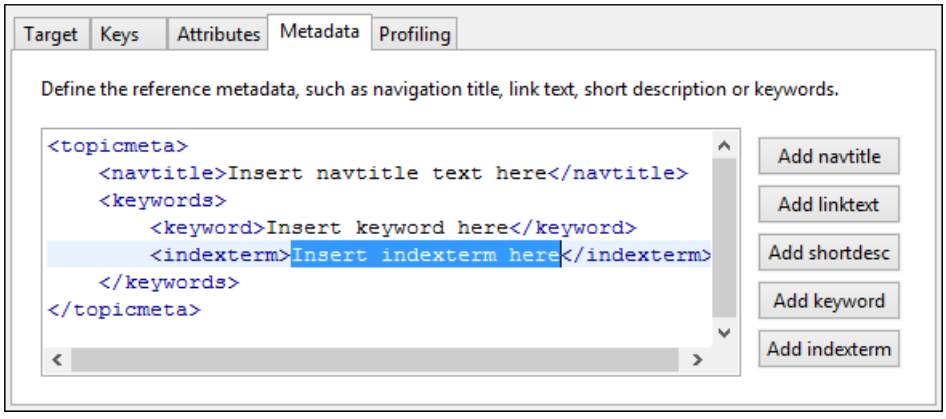


Figure 236: Insert Reference Dialog Box - Metadata Tab

The **Metadata** tab allows you to add metadata elements to the target reference. Use the buttons on the right side of the tab to insert specific metadata elements (you can add the following metadata elements: `navtitle`, `linktext`, `shortdesc`, `keyword`, `indexterm`). The metadata elements are inserted inside a `topicmeta` element. The editing window allows you to easily insert and modify the content of the metadata that will be inserted.

Profiling Tab

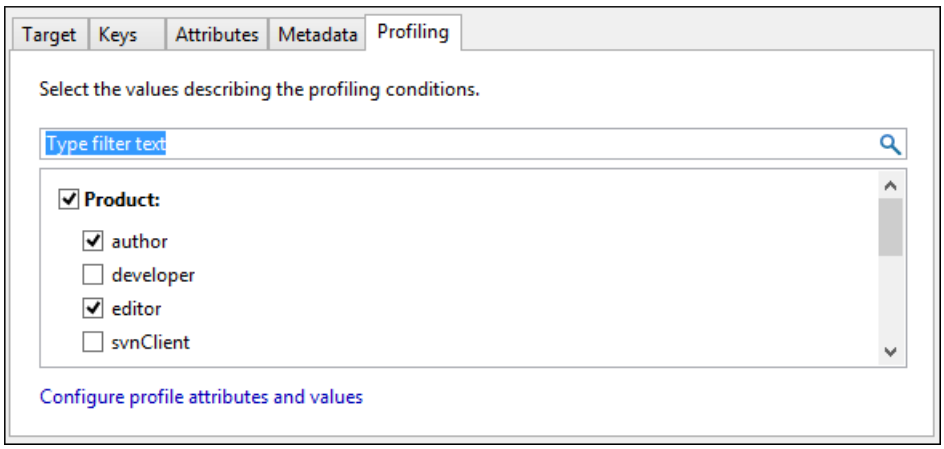


Figure 237: Insert Reference Dialog Box - Profiling Tab

The **Profiling** tab allows you to select or change profiling attributes for the selected reference. This tab displays profiling attributes and their values as determined by the following:

- If your root DITA map references a DITA subject scheme map that defines values for the profiling attributes, those values are used.
- Otherwise, a basic default set of profiling attributes and values are used.

When you modify a value in this tab, the change will also automatically be reflected in the **Attributes** tab. For more information, see the *DITA Profiling / Conditional Text* on page 474 section.

Finalizing Your Insert Reference Configuration

Once you click **Insert** or **Insert and close**, the configured reference is added in the map.

Tip: You can easily insert multiple references by keeping the **Insert Reference** dialog box opened, using the **Insert** button.

Inserting Topic Headings

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.

You can insert a topic heading by doing the following:

- Select **Topic Head** from the **Append Child** or **Insert After** submenus when invoking the contextual menu in the **DITA Maps Manager** view.
- *Open the DITA map in the XML editor* and select the **Insert Topic Heading** action from the main toolbar (or from the **Insert** submenu of the contextual menu).

Those actions open the ***Insert Topic Head** dialog box* that allows you to easily insert a `topichead` element. A **Navigation title** (`navtitle` attribute) is required but other attributes can also be specified from this dialog box (such as **Type**, **Scope**, **Format**, etc.)

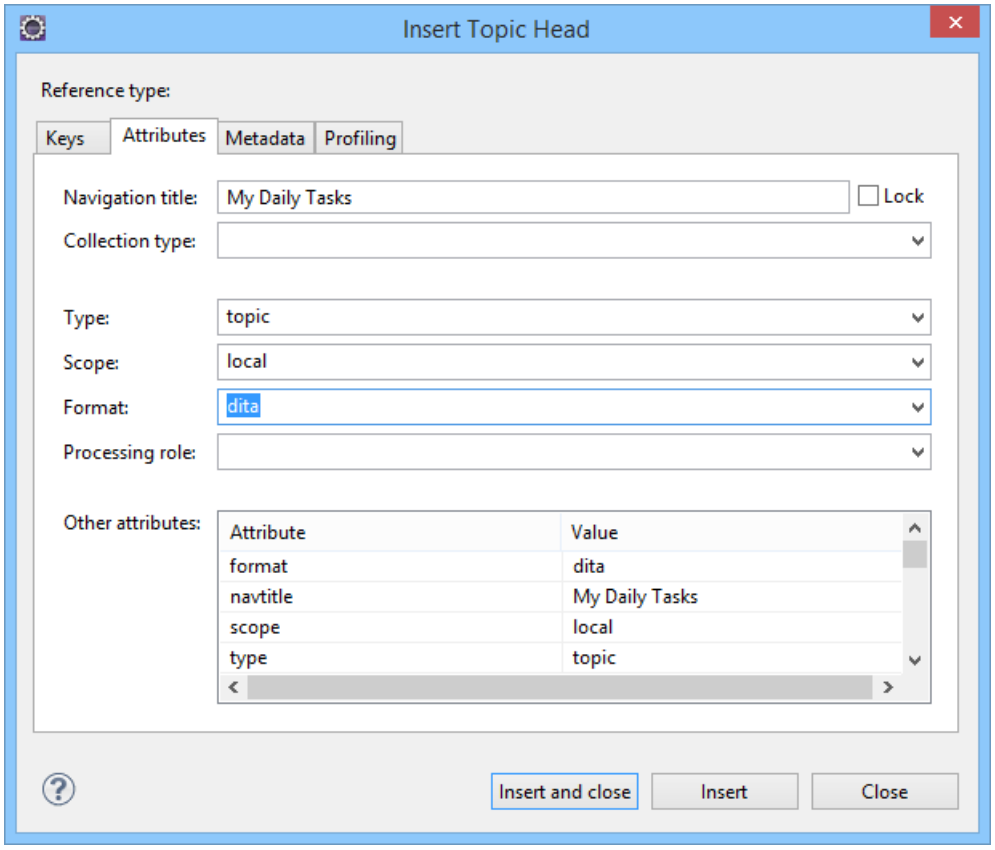


Figure 238: Insert Topic Heading Using the Insert Reference Dialog Box

Inserting Topic Groups

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).

You can insert a topic group by doing the following:

- Select **Topic Group** from the **Append Child** or **Insert After** submenus when invoking the contextual menu in the **DITA Maps Manager** view.
- *Open the DITA map in the XML editor* and select the **Insert Topic Group** action from the main toolbar (or from the **Insert** submenu of the contextual menu).

Those actions open the ***Insert Topic Group** dialog box* that allows you to easily insert a `topicgroup` element and various attributes can be specified (such as **Collection type**, **Type**, **Scope**, **Format**, etc.)

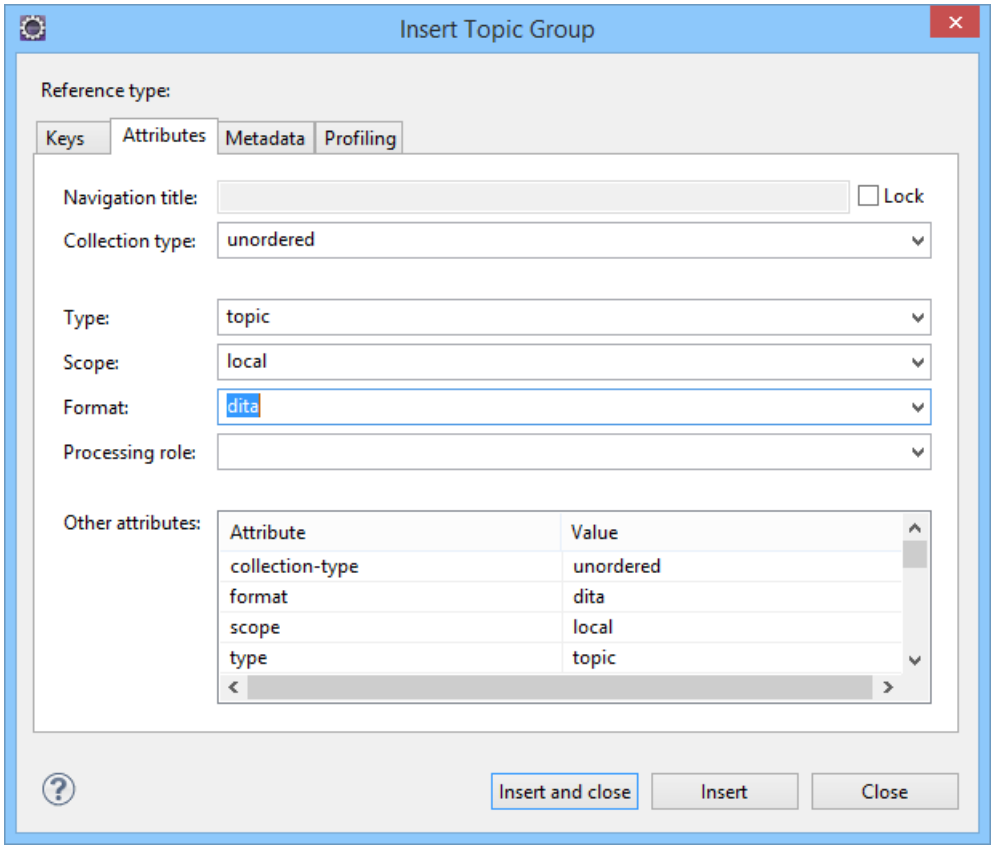


Figure 239: Insert Topic Group Using the Insert Reference Dialog Box

Inserting and Defining Keys in DITA Maps

DITA uses *keys* to insert content that may have different values in various circumstances. Keys provide the means for indirect referencing in DITA. This can make it easier to manage and to reuse content. In DITA, keys are defined in maps and can then be reused and referenced throughout the whole structure of the map.

The following example is a DITA map that defines various values for the `product` key:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
  <!-- product name -->
  <keydef keys="product" product="basic">
    <topicmeta>
      <keywords>
        <keyword>Basic Widget</keyword>
      </keywords>
    </topicmeta>
  </keydef>
  <keydef keys="product" product="pro">
    <topicmeta>
      <keywords>
        <keyword>Professional Widget</keyword>
      </keywords>
    </topicmeta>
  </keydef>
  <keydef keys="product" product="enterprise">
    <topicmeta>
      <keywords>
        <keyword>Enterprise Widget</keyword>
      </keywords>
    </topicmeta>
  </keydef>
</map>
```

Note: The profiling of the names is now contained in the map, where it only has to occur once to reuse throughout the whole map structure.

Key Definition with a Keyword

To insert a *key definition with a keyword* in a DITA map, follow these steps:

1. Open the DITA map in the **DITA Maps Manager**.
2. Invoke the contextual menu and select **Key Definition** from the **Append Child** or **Insert After** submenu (depending on where you want to insert the *key definition* in the DITA map). This opens an **Insert Key Definition** dialog box.
3. Go to the **Keys** tab and enter the name of the key in the **Define keys** field.
4. Go to the **Metadata** tab and click **Add keyword**. In the editing window enter the key value inside the `keyword` element.

   Note: You can profile the key by using the **Profiling** tab and other attributes can also be defined in the **Attributes** tab.

5. Once you are done configuring the *key definition*, click **Insert and close**.

Alternatively, there is a simplified method that can be used if you simply want to define a key with a value inside a *keyword*, without configuring any profiling or other attributes. To use the simplified method, follow these steps:

1. Open the DITA map in the **DITA Maps Manager**.
2. Invoke the contextual menu and select **Key Definition with Keyword** from the **Append Child** or **Insert After** submenu (depending on where you want to insert the *key definition* in the DITA map). This opens a simplified **Insert Key Definition** dialog box.
3. Enter the name of the key in the **Key** field and its value in the **Keyword** field.
4. Click **Insert and close** to finalize the operation.

Key Definition with a Target

To insert a *targeted key definition* (for example, to target a resource such as an image or topic) in a DITA map, follow these steps:

1. Open the DITA map in the **DITA Maps Manager**.
2. Invoke the contextual menu and select **Key Definition** from the **Append Child** or **Insert After** submenu (depending on where you want to insert the *key definition* in the DITA map). This opens an **Insert Key Definition** dialog box.
3. Go to the **Keys** tab and enter the name of the key in the **Define keys** field.
4. Go to the **Target** tab and select a target resource (such as an image or topic).

   Note: You can profile the key by using the **Profiling** tab and other attributes can also be defined in the **Attributes** tab.

5. Once you are done configuring the *targeted key definition*, click **Insert and close**.

Edit Properties in DITA Maps

The **DITA Maps Manager** view includes a feature that allows you to view and edit the properties of a selected node.

The **Edit properties** action is available on both the **DITA Maps Manager** toolbar and in the contextual menu. The action opens the **Edit Properties** dialog box and it includes several tabs with various functions and fields that are initialized with values based upon the node for which the action was invoked.

Note: If you select multiple sibling nodes and invoke the **Edit properties** action, only the **Profiling** tab will be available and your modifications in that tab will be applied to all the selected nodes. If you select multiple nodes that are not on the same hierarchical level, the other tabs will also be available and your modifications will be applied to the parent node (the child nodes will inherit the attributes of the parent node).

You can use the **Edit Properties** dialog box to modify or define attributes, metadata, profiling, or keys in DITA maps or topics. You can also use it to modify the title of root maps.

At the top of the **Edit Properties** dialog box, the **Reference type** drop-down list displays the type of the selected node and it depends on the node for which the action was invoked.

The main section of the dialog box includes the following tabs: **Target**, **Keys**, **Attributes**, **Metadata**, and **Profiling**. The availability of the tabs and their functions depend on the selected node. For example, if you invoke the action on a root map, only the **Attributes**, **Metadata**, and **Profiling** tabs are enabled and the **Title** property can be configured. Also, if you select multiple nodes, only the **Profiling** tab is available.

Target Tab

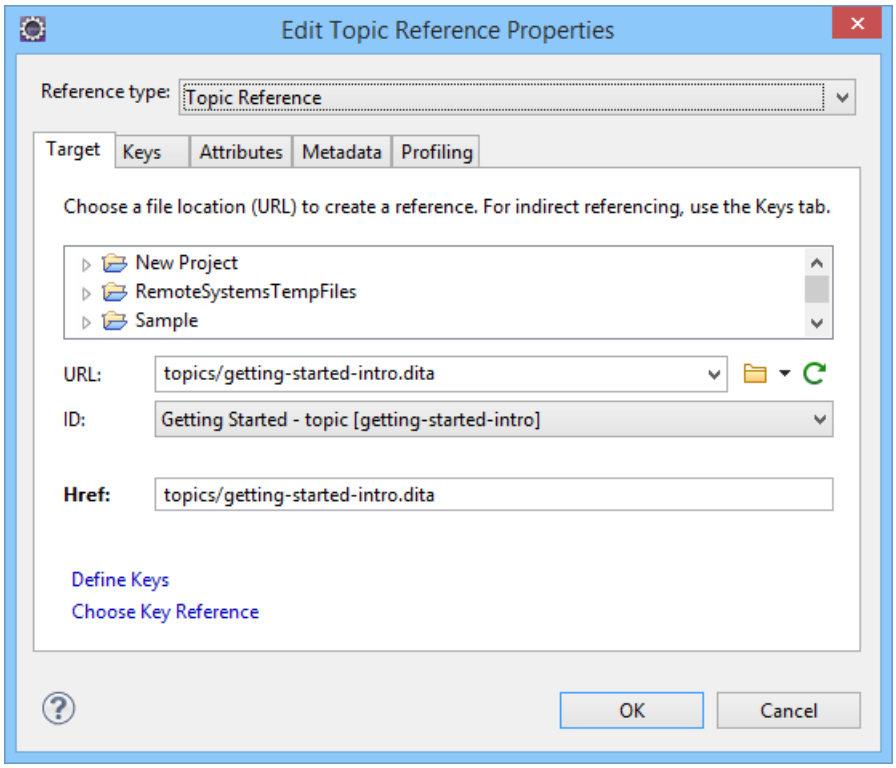


Figure 240: Edit Properties Dialog Box - Target Tab

The **Target** tab of the **Edit Properties** dialog box displays information about the target node on which the action was invoked and allows you to change the target. It includes the following sections and fields:

Choose a file location section

You can browse for and select the source target file by using the file window in this section.

URL

Displays the path to the target and allows you to select or change it by using the combo box or browsing tools.

ID

The drop-down list displays all of the targets that are available for the selected target.

Href

The selected target automatically modifies this value to point to the corresponding `href` attribute of the target element.

Note: If the **Reference type** is a **Navigation Reference**, the **Href** field is changed to **Mapref**, since a `navref` element requires a `mapref` attribute instead.

Keys Tab

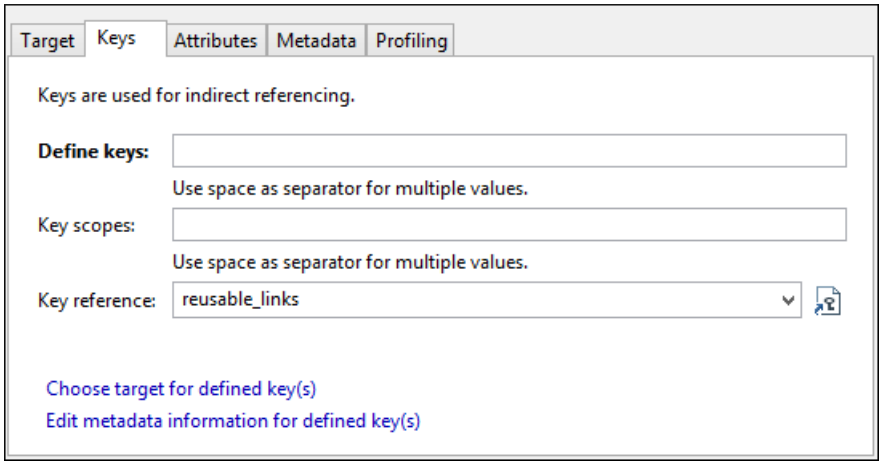


Figure 241: Edit Properties Dialog Box - Keys Tab

The **Keys** tab allows you to use and *define keys* for indirect referencing. For more information, see the *Working with Keys* on page 459 topic. This tab includes the following:

Define keys

Use this text field to define the `keys` attribute for the target.

Key scopes [This option is only available if the Built-in DITA-OT 2.x (with experimental DITA 1.3 support) option is enabled in the *DITA preferences page*]

Use this text field to define or edit the value of a *`keyscope` attribute*. *Key scopes* allow you to specify different sets of key definitions for different map branches.

Key reference

Use this combo box (or the **Choose key reference** button) to select a key that is already defined in the root map.

Attributes Tab

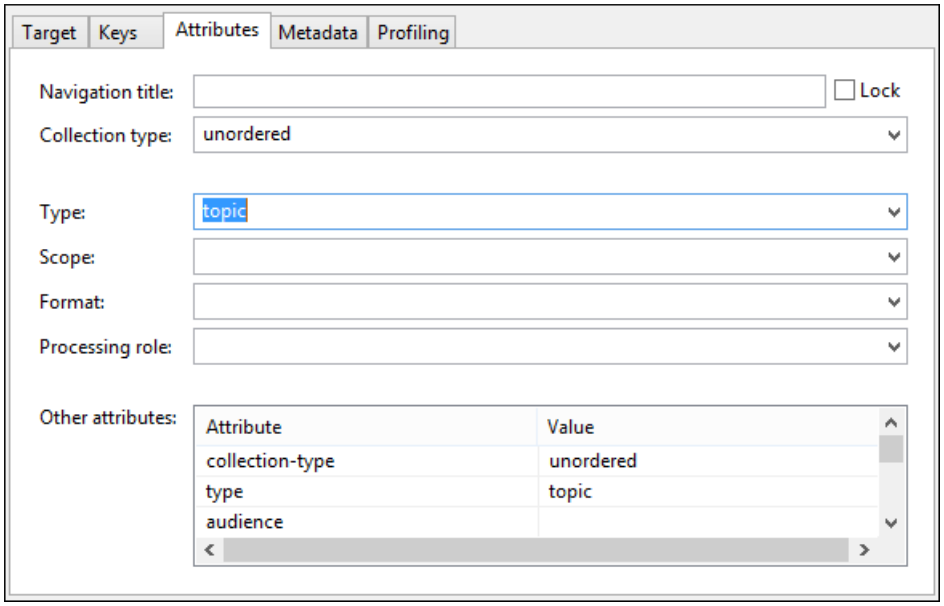
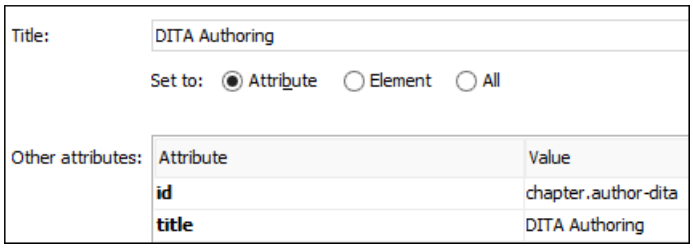


Figure 242: Edit Properties Dialog Box - Attributes Tab

The **Attributes** tab of the **Edit Properties** dialog box allows you to insert and edit attribute values for the target node for which the action was invoked.

If the target is a root map, the tab displays the title of the map. You can change it in the **Title** text field and assign it to an **Attribute**, **Element**, or **All**.



For other types of targets, the tab includes the following sections and fields that can be used to edit the attributes of the target:

Navigation title

This text field allows you to specify a custom navigation title for the target reference. You can enforce the use of the specified title by enabling the **Lock** checkbox.

Collection type

This drop-down list allows you to select the `collection-type` attribute to create hierarchical linking between topics in a DITA map (for example, `unordered`, `sequence`, `choice`, `family`, `-dita-use-conref-target`).

Type

Allows you to select a `type` attribute (such as `topic`, `task`, `concept`, etc.) for the target element. If you want this attribute to always be populated with a detected value (based on the specifications for the target file), enable the **Type** checkbox for the **Always fill values for attributes** option in the *DITA preferences page*.

Scope

This property corresponds to the `scope` attribute of the target element. It is populated automatically, based on the selected file type, unless its value for the selected target file is the same as the default attribute value. If you want this attribute to always be populated with a detected value (based on the specifications), enable the **Scope** checkbox for the **Always fill values for attributes** option in the *DITA preferences page*.

Format

This property corresponds to the `format` attribute of the target element. It is populated automatically, based on the selected file type, unless its value for the selected target file is the same as the default attribute value. If you want this attribute to always be populated with a detected value (based on the specifications), enable the **Format** checkbox for the **Always fill values for attributes** option in the *DITA preferences page*.

Processing Role

This drop-down list allows you to set the `processing-role` attribute to one of the allowed values for DITA reference elements (for example, `resource-only`, `normal`, `-dita-use-conref-target`).

Other attributes table

This table contains the attributes that are available for the selected reference. You can use this table to insert or edit the values of any of the listed attributes. Clicking a cell in the **Value** column allows you to use the combo box to enter, edit, or select attribute values.

Metadata Tab

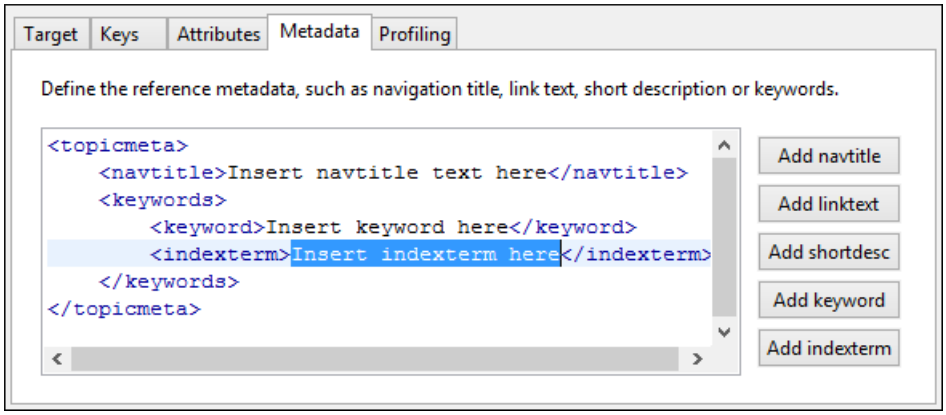


Figure 243: Edit Properties Dialog Box - Metadata Tab

The **Metadata** tab allows you to add metadata elements to the target node. Use the buttons on the right side of the tab to insert specific metadata elements (you can add the following metadata elements: `navtitle`, `linktext`, `shortdesc`, `keyword`, `indexterm`). The metadata elements are inserted inside a `topicmeta` element. The editing window allows you to easily insert and modify the content of the metadata that will be inserted.

Profiling Tab

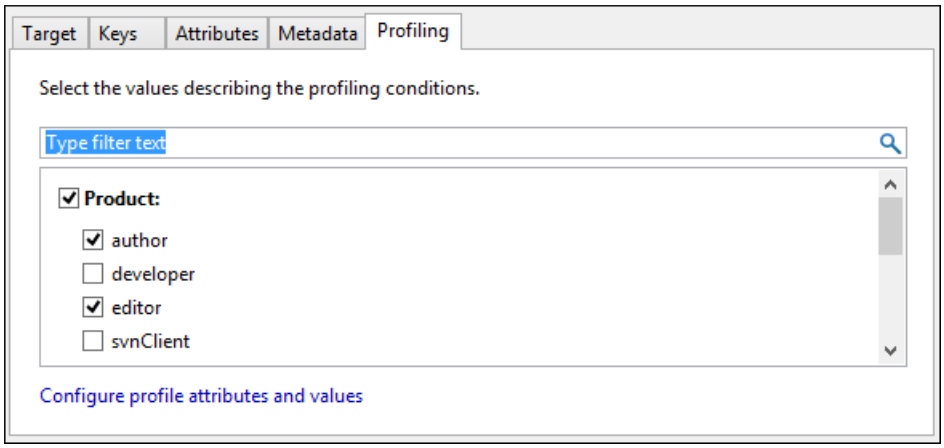


Figure 244: Edit Properties Dialog Box - Profiling Tab

The **Profiling** tab allows you to select or change profiling attributes for the selected target nodes. This tab displays profiling attributes and their values as determined by the following:

- If your root DITA map references a DITA subject scheme map that defines values for the profiling attributes, those values are used.
- Otherwise, a basic default set of profiling attributes and values are used.

When you modify a value in this tab, the change will also automatically be reflected in the **Attributes** tab. For more information, see the *DITA Profiling / Conditional Text* on page 474 section.

Finalizing Your Modifications

Once you click **OK**, all your changes are applied to the target node.

Creating a Table of Contents in DITA

In DITA, the order and hierarchy of a document's table of contents is based directly on *the DITA map that defines the document*. In many media, the creation of a table of contents, based on the map, is automatic. For example, you do not have to do anything special to create a table of contents in WebHelp output.

In other media, you need to tell DITA where the table of contents should occur. For example, in a book you need to tell DITA where to place the table of contents in the structure of the book, and whether to generate other common content lists, such as a list of figures or tables. You do this by *using a `bookmap` to define your book*, and adding the appropriate elements to the `frontmatter`.

To configure a table of contents and other book lists:

1. Open your *bookmap* in the **DITA Maps Manager**.
2. Right-click the bookmap and select **Append child** > **Frontmatter**. The ***Insert Reference** dialog box* appears.
3. Click **Insert and Close** to insert the `frontmatter` element.
4. Right-click the `frontmatter` element and create a `booklists` element using **Append child** > **Book Lists**.
5. Use the same steps to create a `toc` element and to add any additional `booklists` elements you want, such as `tablelist`.

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 reference topics using URIs instead of local paths
- You have URI content references in your DITA topic files and you want to map them to local files when the map is transformed

In such situations, you have to *add the catalog to Oxygen XML Editor plugin*. The **DITA Maps Manager** view will solve the displayed topic refs through the added XML catalog, as will DITA map transformations (for PDF output, XHTML output, etc.)

Finding Resources Not Referenced in DITA Maps

Over the course of time large projects can accumulate a vast amount of resources from a variety of sources. Especially in organizations with a large number of content writers or complex project structures, organizing the project resources can become a challenge. Over time a variety of actions can cause resources to become orphaned from DITA maps. To assist you with organizing project resources, Oxygen XML Editor plugin includes an action, **Find Unreferenced Resources**, that searches for orphaned resources that are not referenced in DITA maps.

To perform this search, open the DITA map in the **DITA Maps Manager**, invoke the contextual menu on the DITA map, and select **Find Unreferenced Resources**. This action opens the **Find Unreferenced Resources** dialog box, which allows you to specify some search parameters:

- **DITA Maps** - Provides a list of DITA maps to be included in the search and allows you to **Add** maps to the list or **Remove** them.
- **Folders** - Provides a list of folders to be included in the search and allows you to **Add** or **Remove** specific folders.
- **Filters** - Provides three combo boxes that allow you to filter the search to include or exclude certain files or folders:
  - **Include files** - Allows you to filter specific files to include in the search.
  - **Exclude files** - Allows you to filter specific files to exclude from the search.
  - **Exclude folders** - Allows you filter specific folders to exclude from the search.

  Note: In any of the filter combo boxes you can enter multiple filters by separating them with a comma and you can use the *?* and * wildcards. Use the drop-down arrow to select a previously used filter pattern.

Chunking DITA Topics

By default, when a *DITA map* is published to an online format, each topic becomes a separate page in the output. In some cases, you may want to combine multiple source topics into one output page. For instance, you may want to combine several types of information into a single page, or you may have chosen to create many small DITA topics for reuse purposes but feel they are too small to be useful to a reader by themselves.

To chunk DITA topics, you set the chunking attribute on the `topicref` that contains the sub-topics in a DITA map. There are a number of different values you can set on the chunking attribute. See the *DITA documentation* for full details.

To achieve the effects you want in your topics and table of contents, you may also need to set the `toc` and `collection-type` attributes on the sub-topics or container topic to suitable values. See the DITA documentation for details.

You can set the `collection-type` attribute on your topics using the **Edit Properties** action in the **DITA Maps Manager**. To set the `toc` and `chunk` attributes, you must open the map file in the editor and add or edit the attributes directly (double-click the map icon in the **DITA Maps Manager** to open the map in the editor).

DITA Map Validation and Completeness Check

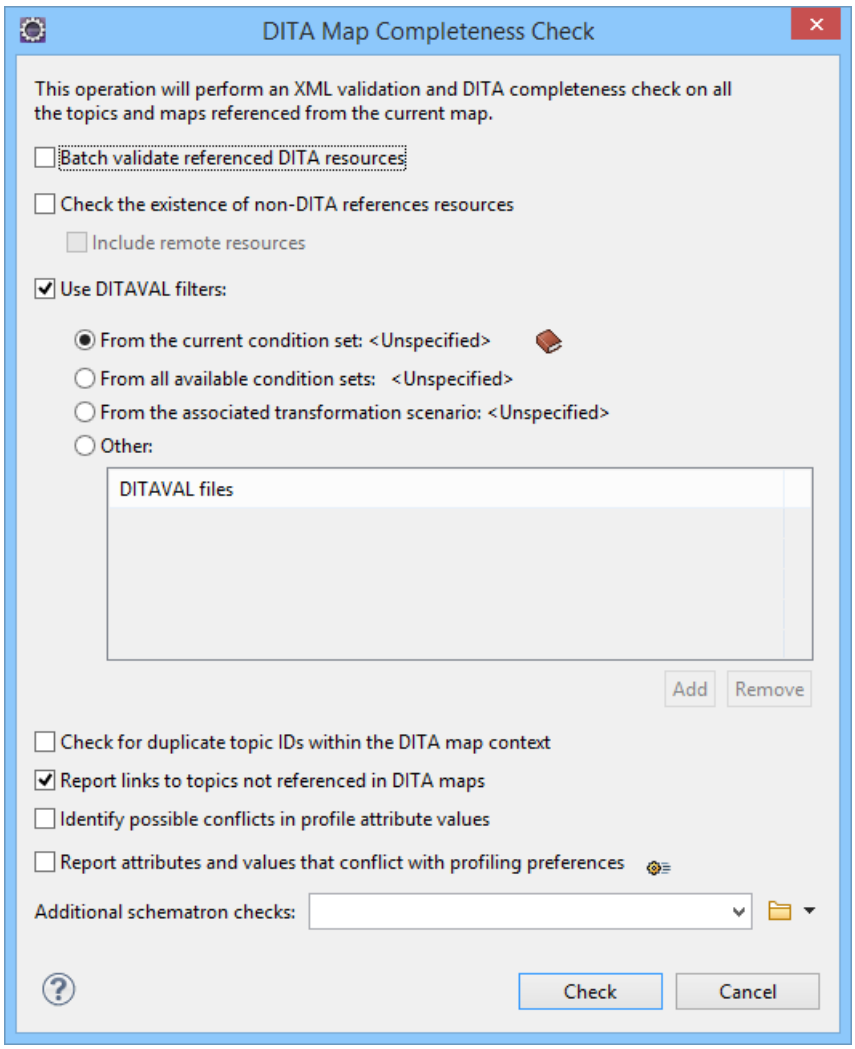
*The **Validate and Check for Completeness** action* allows you to validate a DITA map and configure the options for the validation process. You can find the **Validate and Check for Completeness** action in the toolbar of *the **DITA Maps Manager** view*. Invoking this action opens the **DITA Map Completeness Check** dialog box, which allows you to configure the DITA map validation.

Validation Process

The validation process of a DITA map includes the following:

- Verifies that the file paths of the topic references are valid. For example, if an `href` attribute points to an invalid file path, it is reported as an error in the message panel at the bottom of the editor.
- Validates each referenced topic and map. Each topic file is opened and validated against the appropriate DITA DTD. If another DITA map is referenced in the main one, the referenced DITA map is verified recursively, applying the same algorithm as for the main map.
- If errors or warnings are found, they are displayed in a separate message pane at the bottom of the editor and clicking them takes you to the location of the error or warning in the file where it was found.

DITA Map Completeness Check Dialog Box



You can configure the validation process with the following options that 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 referenced 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 referenced resources to non-DITA files.
  - **Include remote resources** - Enable this option 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. You can choose between the following options:
  - **From the current condition set** - The map is filtered using the condition set applied currently in the **DITA Maps Manager** view. Clicking the **Details** icon opens a topic in the Oxygen XML Editor plugin user guide that explains how to create a profiling condition set.

  - **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. Use the **Add** or **Remove** buttons to configure the list.
- **Check for duplicate topic IDs within the DITA map context** - Checks for multiple topics with the same ID in the context of the entire map.
- **Report links to topics not referenced in DITA maps** - Checks that all referenced topics are linked in the DITA map.
- **Identify possible conflicts in profile attribute values** - When the profiling attributes of a topic contain values that are not found in parent topic profiling attributes, the content of the topic is overshadowed when generating profiled output. This option reports these 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 (you can click the button to open this 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 Editor plugin uses for the validation of DITA resources.

Validating a DITA Map

You should validate your maps regularly to make sure that your topics are valid, and all of the relationships between them are working. Changing one topic, image, or piece of metadata may create errors in references that rely on them. You may not discover these problems all at once. Validate your map to catch all of these kinds of problems. The longer you wait between validating your maps, the more difficult it may be to detect and correct any errors you find.

To validate a map:

1. In *the **DITA Maps Manager** view*, make sure that the tab that holds your *root map* is selected and that the **Root map** selection is set either to the name of your *root map* or to `<current map>`.
2. It is a good practice to refresh your DITA map before running the validation process. To do so, select the DITA map in the **DITA Maps Manager** view and click **File** > **Refresh (F5)**.
3. Click the **Validate and Check for Completeness** button on the **DITA Maps Manager** toolbar to open the ***DITA Map Completeness Check*** dialog box.
4. If you are using profiling, check the **Use DITAVAL filters** box and select the appropriate option.
5. Select any other options you want to check.
6. Click **Check**.
   The validator runs over the entire map. This process may take a while if the map is large. If there are any errors or warnings, they are displayed in a separate message pane at the bottom of the editor. You can click each error or warning to jump to the file where the problem was found.

Working with DITA Topics

DITA is a structured writing format. Structure can have several meanings, all of which are relevant to DITA.

Information Types

The structure of a piece of content refers to how the words and images are selected and organized to convey information. One approach to structured writing is to divide content into discrete blocks containing different types of information, and then to combine those blocks to form publications. DITA is based on this approach, and encourages the author to write in discrete blocks called topics. DITA provides three base topic types (concept, task, and reference), a number of extended topic types, and the capability to create new topic types through specialization.

Text Structure

Every piece of text is made up of certain text structures, such as paragraphs, lists, and tables. DITA supports text structures through XML elements such as `p`, `ol`, and `simpletable`. *The DITA markup* specifies the text structures, but not how they will be published in different types of media. The formatting of text structures is determined by the output transformations and may be customized to meet the needs of a variety of different organizations and media.

Semantic Structure

Semantic structure is structure that shows the meaning of things. For example:

- A `task` element specifies that a block of content contains the description of a task
- A `codeblock` element specifies that a block of text consists of programming code
- A `uicontrol` element specifies that a word is the name of a control in a computer GUI
- The `platform` profiling attribute specifies that a particular piece of content applies only to certain computing platforms

Semantic structure is important in a structured writing system because it allows both authors and readers to find content, and it allows processing scripts to process various pieces of content differently, based on their role or meaning. This can be used to do things such as filtering content related to a specific product so that you can produce documentation on many products from the same source.

There can be many forms of semantics captured in a document set. DITA captures some of these in topics and some of them in maps. If you are using a CMS, it may capture additional semantics.

Document Semantics

Documents consist of a number of different elements that may be made up of the same basic text structures as the rest of the text, but have a special function within the structure of the document. For instance, both tables of contents and indexes are lists, but they play a special role in the document. Chapters and sections are just sequences of paragraphs and other text structures, yet they are meaningful in the structure of the document. In some cases, such as indexes and tables of contents, these structures can be generated from semantic information embedded in the source. For instance, a table of contents can be built by reading the titles of chapters and sections. DITA provides elements to describe common document semantics.

Subject Matter Semantics

In some cases, the semantics of the content relate directly to the subject matter that the content describes. For instance, DITA supports tags that allow you to mark a piece of text as the name of a window in a software application (`wintitle`), or to mark a piece of text as applying only to a particular product.

Audience Semantics

In some cases, the semantics of the content relate to the audience that it is addressed to. For instance, a topic might be addressed to a particular role, or to a person with a particular level of experience. DITA provides an `audience` element to capture audience metadata.

Creating Topic Structures

Oxygen XML Editor plugin provides a number of tools to help you create topic structures:

- The **Content Completion Assistant**, which shows you which elements can be created at the current position
- The **Model** view, which shows you the complete structure supported by the current element
- The **Outline** view, which shows you the current structure of your document
- The **DITA** toolbar, which helps you insert many common structures

Creating a New DITA Topic

The basic building block for DITA information is the DITA topic. DITA provides a variety of specialized 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 box.
- *Reference* - For reference information.
- *Topic* - The base topic type from which all other topic types are specialized. Typically, it is used when a more specialized topic type is inappropriate.

To add a new topic to a DITA map, follow these steps:

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 **Append Child** > **New**. To insert the topic as a sibling to the current node, choose **Insert After** > **New**.
   A dialog box is displayed that allows you to create a new DITA topic using various types of DITA file templates and provides some options that help you to configure the new topic.

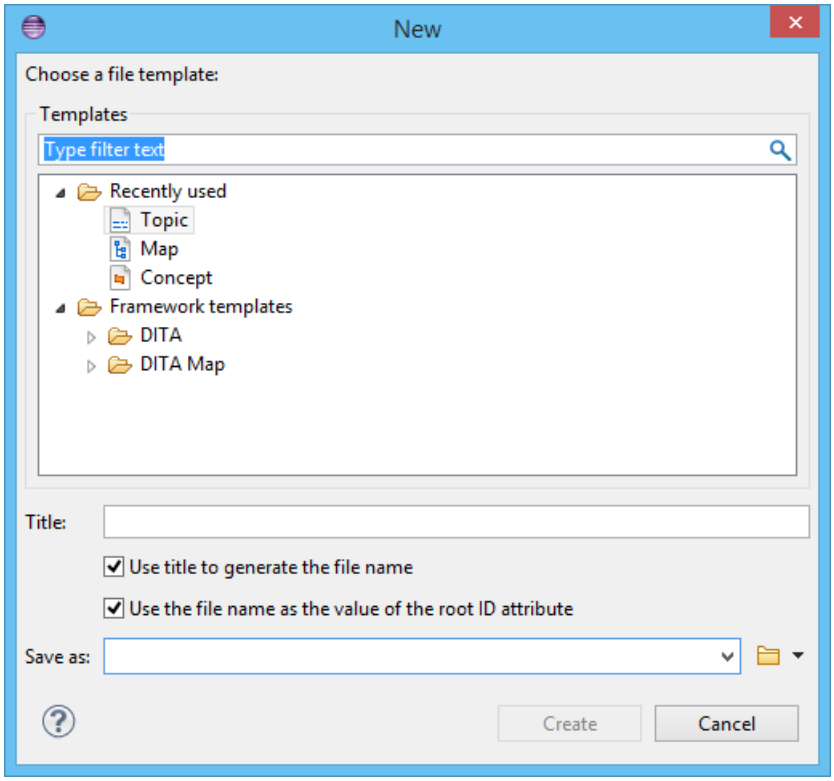


Figure 245: The New DITA Topic Dialog Box

Note: The templates that appear in this dialog box include all templates that have an associated properties file and the `type` property is set to `dita`, as well as templates that do not have an associated properties file or the `type` property is not defined.

3. Select the appropriate DITA topic type from the list templates. You can use the **filter** text field to search for a template.
4. You can use the following options to preconfigure some topic creation tasks:
   a) **Title** - The text entered in this field will be used as the value of the root `title` element for the new topic. The title is set only if the selected template contains a root with a `title` element as its first child.
   b) **Use title to generate the file name** - Select this option to use the text entered in the **Title** field to automatically generate a file name. The generated name will transform spaces into underscores (_), all illegal characters will be removed, and all upper case characters changed to lower case (the generated name can be seen in the **Save as** field).
   c) **Use the file name as the value of the root ID attribute** - Select this option to use the file name (without the file extension) in the **Save as** field as the value of the root `id` attribute for the new topic.

5. Select a file name and path in the **Save as** field.
6. Click the **Create** button.
   A reference (`topicref`) to the new topic is added to the current map and the new topic is opened in the editor.

Editing DITA Topics

Oxygen XML Editor plugin provides a number of features to help you edit DITA topics.

Author Mode

DITA is an *XML format*, although you do not have to write raw XML to create and edit DITA topics. Oxygen XML Editor plugin provides a graphical view of your topics in ***Author** mode*. Your topics will likely open in **Author** mode by default, so this is the first view you will see when you open or edit a DITA topic. If your topic does not open in **Author** mode, just click **Author** at the bottom left of the editor window to switch to this mode.

Author mode presents a graphical view of the document you are editing, similar to the view you would see in a word processor. However, there are some differences, including:

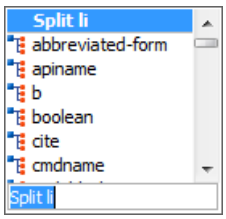
- **Author** mode is not a **WYSIWYG** view. It does not show you exactly what your content will look like when printed or displayed on-screen. The appearance of your output is determined by the DITA publishing process, and your organization may have modified that process to change how the output is displayed. Oxygen XML Editor plugin has no way of determining what your final output will look like or where line breaks or page breaks will fall. Treat **Author** mode as a friendly visual editing environment, not a faithful preview of your output.
- Your document is still an XML document. **Author** mode creates a visual representation of your document by applying a CSS stylesheet to the XML. You can see the XML at any time by switching to ***Text** mode*. You, or someone in your organization, can change how the **Author** view looks by changing the CSS stylesheet or providing an alternate stylesheet.
- Your aim in editing a DITA document is not to make it look right, but to create a complete and correct DITA XML document. **Author** mode keeps you informed of the correctness of your content by highlighting XML errors in the text and showing you the current status in a box at the top right of the editor window. Green means that your document is valid, yellow means valid with warnings, and red means invalid. Warnings and errors are displayed when you place the cursor on the error location.
- Your XML elements may have attributes set on them. Conventionally, attributes are used to contain metadata that is not displayed to the reader. By default, attributes are not displayed in the **Author** view (though there are some exceptions) and cannot be edited directly in the **Author** view (though in some cases the CSS that drives the display may use form controls to let you edit attributes directly). To edit the attributes of an element, place your cursor on the element and press **Alt+Enter** to bring up the attribute editor. Alternatively, you can use the ***Attributes** view* to edit attributes.

Tip: You can select **Hints** from the **Styles** drop-down menu (available on the **Author Styles** toolbar) to display tooltips throughout the DITA document that offers additional information to help you with the DITA structure. For more information, see the *Selecting and Combining Multiple CSS Styles* section.

Content Completion Assistance

Since it is a structured format, DITA only allows certain elements in certain places. The set of elements allowed differ from one DITA topic type to another (this is what makes one topic type different from another). To help you figure out which elements you can add in any given place and help you understand what they mean, Oxygen XML Editor plugin has a number of content completion assistance features.

- **The Enter key:** In **Author** mode, the Enter key does not create line breaks, it brings up the **Content Completion Assistant** to help you enter a new element. In XML, you do not use line breaks to separate paragraphs. You create paragraphs by creating paragraph elements (element `p` in DITA) and tools insert the line breaks in the output and on-screen.



The **Content Completion Assistant** not only suggests new elements you can add. If you press **Enter** at the end of a block element (such as a paragraph) it suggests creating a new element of the same type. If you press **Enter** in the middle of a block element, it suggests splitting that element into two elements.

A useful consequence of this behavior is that you can create a new paragraph simply by hitting **Enter** twice (just as you might in a text editor).

As you highlight an element name, a basic description of the element is displayed. Select the desired element and press **Enter** to create it.

To wrap an element around an existing element or piece of text, simply select it and press **Enter** and use the **Content Completion Assistant** to choose the wrapper element.

- **The Model view:** You can see the entire model of the current element by opening the **Model** view (**Window** > **Show View** > **Model**, if the view is not already open). The **Model** view shows you what type of content the current element can contain, all the child elements it can contain, all its permitted attributes, and their types.

Tip: You can also select **Inline actions** from the **Styles** drop-down menu (available on the **Author Styles** toolbar) to display possible elements that are allowed to be inserted at various locations throughout the DITA document. For more information, see the *Selecting and Combining Multiple CSS Styles* section.

The DITA Toolbar

The **DITA** toolbar contains buttons for inserting a number of common DITA elements (elements that are found in most DITA topic types).

If the **DITA** toolbar is not displayed, right-click anywhere on the toolbar area, select **Configure Toolbars**, and select it from the displayed dialog box.

Note: The **DITA** toolbar contains a list of the most common elements and actions for DITA, such as inserting an image, creating a link, inserting a content reference, or creating a table. It does not contain a button for every possible DITA element. For a complete list of elements you can currently create, press **Enter** to bring up the **Content Completion Assistant**.

The DITA Menu

Whenever the current document in the editor is a DITA document, the **DITA** menu is displayed in the menu bar. It contains a large number of commands for inserting elements, creating content references and keys, edit DITA documents, and controlling the display. These commands are specific to DITA and supplement the general editing commands available for all document types. As with the **DITA** Toolbar, the **DITA** menu does not list every possible DITA element.

Reusing DITA Content

Reusing content is one of the key features of DITA. DITA provides a number of different methods for reusing content. Oxygen XML Editor plugin provides support for each of these methods.

Reusing Topics in DITA Maps

A DITA topic does not belong to any one publication. You add a DITA topic to a publication by referencing it in a map. You can reference the same topic in more than one map.

Reusing Content with References and Keys

DITA allows you to reuse content by referencing it in another topic. DITA provides two mechanisms for including content by reference: `conref` and `conkeyref`. A `conref` creates a direct reference to a specific element of another topic. A `conkeyref` creates a reference to a key, which in turn points to a specific element of another topic. The advantage of using a `conkeyref` is that you can change the element that is included by changing the key reference. For example, since keys are defined in maps, a different key reference is used when you include your topic in a different map.

Oxygen XML Editor plugin provides support for both `conref` and `conkeyref` mechanisms.

While the `conref` and `conkeyref` mechanisms can be used to reference any content element, it is considered best practice to only *conref* or *conkeyref* content that is specifically set and managed as reusable content. This practice helps reduce expensive errors, such as an author accidentally deleting the source element that other topics are including by `conref`. Oxygen XML Editor plugin can help you create a reusable component from your current content.

Reusing Content with Reusable Components

DITA allows you to select content in a topic, create a reusable component from it and reference that component in other locations. Each reusable component is created as a separate file. Anytime the content needs to be edited, you only need to update it in the component file and all the locations in your topics that reference it will also be updated. This can help you to maintain continuity and accuracy throughout your documents.

Reusing Content with Variables

DITA allows you to replace the content of certain elements with the value pointed to by a key. This mechanism effectively means that you can create variables in your content, which you can then output with various different values by changing the value the key points to. This is done by profiling the definition of the key value, or by substituting another map with various different key values that are defined.

Reusing DITA Topics in Multiple Maps

You can reuse a DITA topic simply be including it in more than one map:

1. To create a new map, select **File** > **New** > **Other** > **Oxygen XML Editor plugin**, or click the  **New** button on the toolbar, select **New from Templates**, go to **Framework templates** > **DITA Map**, and choose the appropriate type of map.
2. Add existing topics to the new map by dragging and dropping them from the **Project** view or the file system (or right-click the map icon, or on a topic already in the map, and select **Append child** or **Insert After**).
3. If your topics use key references, set up the appropriate key definitions in your new map. You can set the keys when you add the topics, or afterwards by right-clicking the topic in the **DITA Maps Manager**, selecting  **Edit Properties**, and defining them in the **Keys** tab.
4. If you want to define relationships between topics, other than those defined in the topics themselves, *add a relationship table to your map* or to a separate map linked to your main map.
5. When you have finished adding topics, check that your map is complete and that all topic links and keys resolve correctly. To do this validation, click the  **Validate and Check for Completeness** action on the toolbar in the **DITA Maps Manager**.

Working with Content References

The DITA `conref` feature (short for *content reference*) lets you insert a piece of source content by referencing it from its source. When you need to update that content, you only need to do it in one place.

There are several 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 and you insert references from the content of the reusable component files. For example, suppose that you have a disclaimer that needs to be included in certain sections of your documentation. You can create a reusable component that contains

your disclaimer and reuse it as often as you need to. If the disclaimer ever needed to be updated, you only have to edit it in one file.

- *Single-source content references* - You may prefer to keep many pieces of reusable content in one file. For example, you might want to create a single file that contains all the actions that are available in various menus or toolbars for your software application. Then, wherever you need to describe or display an action in your documentation, you can reuse content from that single file by inserting content references. This strategy requires more setup than reusable components, but might make it easier to centrally managing the reused content and it allows for more flexibility in the XML structure of the reusable content.
- *Arbitrary content references* - Although it is not recommended, you can create content references amongst topics without storing the reusable content in components or a single file. This strategy might make it difficult to manage content that is reused and to maintain continuity and accuracy, since you may not have any indication that content you are editing is reused elsewhere.

Oxygen XML Editor plugin creates a reference to the external content by adding a `conref` attribute to an element in the local document. The `conref` attribute defines a link to the referenced content, made up of a path to the file and the topic ID within the file. The path may also reference a specific element ID within the topic. Referenced content is not physically copied to the referencing file. However, by default Oxygen XML Editor plugin displays it in **Author** mode as if it is there in the referencing file. If you do not want referenced content displayed, *open the **Preferences** dialog box*, go to **Editor** > **Edit modes** > **Author**, and disable the **Display referenced content** option.

Note: A reference also displays tracked changes and comments that are included in the source fragment. To edit these comments (or accept/reject changes) right-click the comment or tracked change and select **Edit Reference**.

Tip: To search for references made through a direct content references, use the **Search References** action from the contextual menu.

Creating a DITA Content Reference

DITA Content Reference

A DITA content reference, or `conref`, is one of the main *content reuse features of DITA*. It is a mechanism for re-using the same content in multiple topics (or even in multiple locations within the same topic).

In order for a `conref` to be created, the source content must have an `id` attribute that the `conref` can reference. Therefore, creating a `conref` requires that you add an `id` to the content to be reused before inserting a `conref` into the topic that reuses the referenced content.

Assigning an *ID* to the Referenced Content

To add an `id` to a DITA element in a topic, place the cursor on the element and select  **Edit Attributes** from the contextual menu to open the *in-place attribute editor*. Enter `id` as the **Name** of the attribute and a value of your choice in the **Value** field. You can also use *the **Attributes** view* to enter a value in the **id** attribute.

Note: The element may already have an `id`, since in some cases Oxygen XML Editor plugin automatically generates an *id* value when the `id` attribute is created.

Creating a Content Reference

To create a content reference (`conref`), follow these steps:

1. Make sure the element you want to reference has an *id assigned to it*.
2. In ***Author** mode*, place the cursor at the location where you want the reused content to be inserted.
3. Select the  **Reuse Content** action on the main toolbar (or from the **DITA** menu or **Reuse** submenu of the contextual menu). The ***Reuse Content** dialog box* is displayed.
4. In the **Location** field of the **Reuse Content** dialog box, select the topic that contains the element you want to reference. The elements that you can reference are presented in a table.

5. Select the **Element ID** of the element (or elements) from which you want to insert the content, and verify the content in the **Preview** pane. The `id` value of the element that you select is automatically added to the **Reference to (conref)** field.
6. Make any other selections you need in the ***Reuse Content*** *dialog box*. If you select multiple elements, the **Expand to (conrefend)** field is automatically filled with the `id` value of the last element in your selection.
7. Click **Insert** or **Insert and close** to create the content reference.

Other Ways to Reuse Content

An alternate way to reuse content is to use the Oxygen XML Editor plugin ***Create Reusable Component*** and ***Insert Reusable Component*** actions (available in the **DITA** menu and the **Reuse** submenu of the contextual menu). They handle the details of creating an *id* and *conref* and creates reusable component files, separate from your normal content files. This can help you manage your reusable content more effectively.

You can also *insert reusable content using content key references*. This may also make reusable content easier to manage, depending on your particular situation and needs.

Creating a DITA Content Key Reference

DITA Content Key Reference

A DITA content key reference, or `conkeyref`, is a mechanism for inserting a piece of content from one topic into another. It is a version of the *DITA content reference mechanism* that uses *keys* to locate the content to reuse rather than direct references to topics that contain reused content.

As with a *conref*, a *conkeyref* requires that the element to be reused has an `id` attribute. It also requires the topic that contains the reusable content to be assigned a *key* in a map. As with all uses of keys, you can substitute multiple maps or *use profiling* to create more than one definition of keys in a single map. This allows the same `conkeyref` to pull in content from various sources, depending on how your build is configured. This can make it easier to create and manage sophisticated content reuse scenarios.

Creating a Content Key Reference

To create a content key reference (`conkeyref`), follow these steps:

1. Make sure the topic that contains the reusable content is assigned a key in the DITA map and the element you want to reference has an *id* assigned to it.
2. In ***Author*** *mode*, place the cursor at the location where you want the reused content to be inserted.
3. Select  **Reuse Content** on the main toolbar (or from the **DITA** menu or **Reuse** submenu of the contextual menu). The ***Reuse Content*** *dialog box* is displayed.
4. Select the **Key** radio button for the content source and use the  **Choose Key Reference** button to select the key for the topic that contains the reusable content (you can also select one from the drop-down list in the **Key** field). The elements that you can reference from the source are presented in the table in the middle of the **Reuse Content** dialog box.
5. Select the **Element ID** of the element (or elements) that you want to insert, and verify the content in the **Preview** pane. The `id` value of the element that you select is automatically added to the **Reference to (conkeyref)** field.
6. Make any other selections you need in the ***Reuse Content*** *dialog box*. If you select multiple elements, the **Expand to (conrefend)** field is automatically filled with the `id` value of the last element in your selection.
7. Click **Insert** or **Insert and close** to create the content reference.

Editing DITA Content References

Oxygen XML Editor plugin also includes some actions that allows you to quickly edit existing content references. When the element that contains a content reference (`conref` or `conkeyref`) is selected, the following actions are available in the **DITA** menu and the **Reuse** submenu of the contextual menu:

Insert/Edit Content Reference

Opens the **Insert/Edit Content Reference** dialog box that allows you to select/change the source location (or key) and source element of a content reference (or content key reference), and the reference details (`conref`/`conkeyref` and `conrefend` attributes). See the *Reuse Content Dialog Box* on page 439 topic for more information.

Replace Reference with content

Replaces the referenced fragment (`conref` or `conkeyref`) at the cursor position with its content. This action is useful if you want to make changes to the content in the currently edited document without changing the referenced fragment in its source location.

Remove Content Reference

Removes the content reference (`conref` or `conkeyref`) inside the element at the cursor position.

Reuse Content Dialog Box

The **Reuse Content** dialog box provides a mechanism for reusing content fragments. DITA `conref`, `conkeyref`, and `keyref` attributes can be used to insert references to reusable content. The `conref` attribute stores a reference to another element and is processed to replace the referencing element with the referenced element. The `conkeyref` attribute uses *keys* to locate the content to reuse rather than direct references to the topic that contains the reusable content. The `keyref` attribute also uses *keys* and can be used to indirectly reference metadata that may have different values in various circumstances.

Note: For a `conref` or `conkeyref`, to reference the content inside a DITA element, the source element must have an `id` attribute assigned to it. The element containing the content reference acts as a placeholder for the referenced element. For more details about DITA `conref` and `conkeyref` attributes, go to *http://docs.oasis-open.org/dita/v1.2/os/spec/archSpec/conref.html*.

Note: For the purposes of using a `keyref`, keys are defined at map level and referenced afterwards. For more information about the DITA `keyref` attribute, go to *http://docs.oasis-open.org/dita/v1.2/os/spec/common/thekeyrefattribute.html*.

Oxygen XML Editor plugin *displays the referenced content* of a DITA content reference if it can resolve it to a valid resource. If you use URIs instead of local paths in your XML documents and your DITA OT transformation needs an XML catalog to map the URIs to local paths, you need to *add the catalog to Oxygen XML Editor plugin*. If the URIs can be resolved, the referenced content is displayed in **Author** mode and in the transformation output.

In **Author** mode, a references to reusable content (`conref`, `conkeyref`, or `keyref`) can easily be inserted at the cursor position by using the **Reuse Content** dialog box. It can be opened with any of the following methods:

- Go to **DITA** > **Reuse Content**.
- Click the **Reuse Content** action on the main toolbar.
- In the contextual menu of the editing area, go to **Reuse** > **Reuse Content**.

Your selection at the top of the dialog box for choosing the content source determines whether Oxygen XML Editor plugin will insert a `conref`, `conkeyref`, or `keyref`.

If you select **Location** for the content source, a *content reference* (`conref`) will be inserted. If you select **Key** for the content source, keys will be used to insert a *content key reference* (`conkeyref`) or a *key reference* (`keyref`).

Content Reference (`conref`) Options Using the Reuse Content Dialog Box

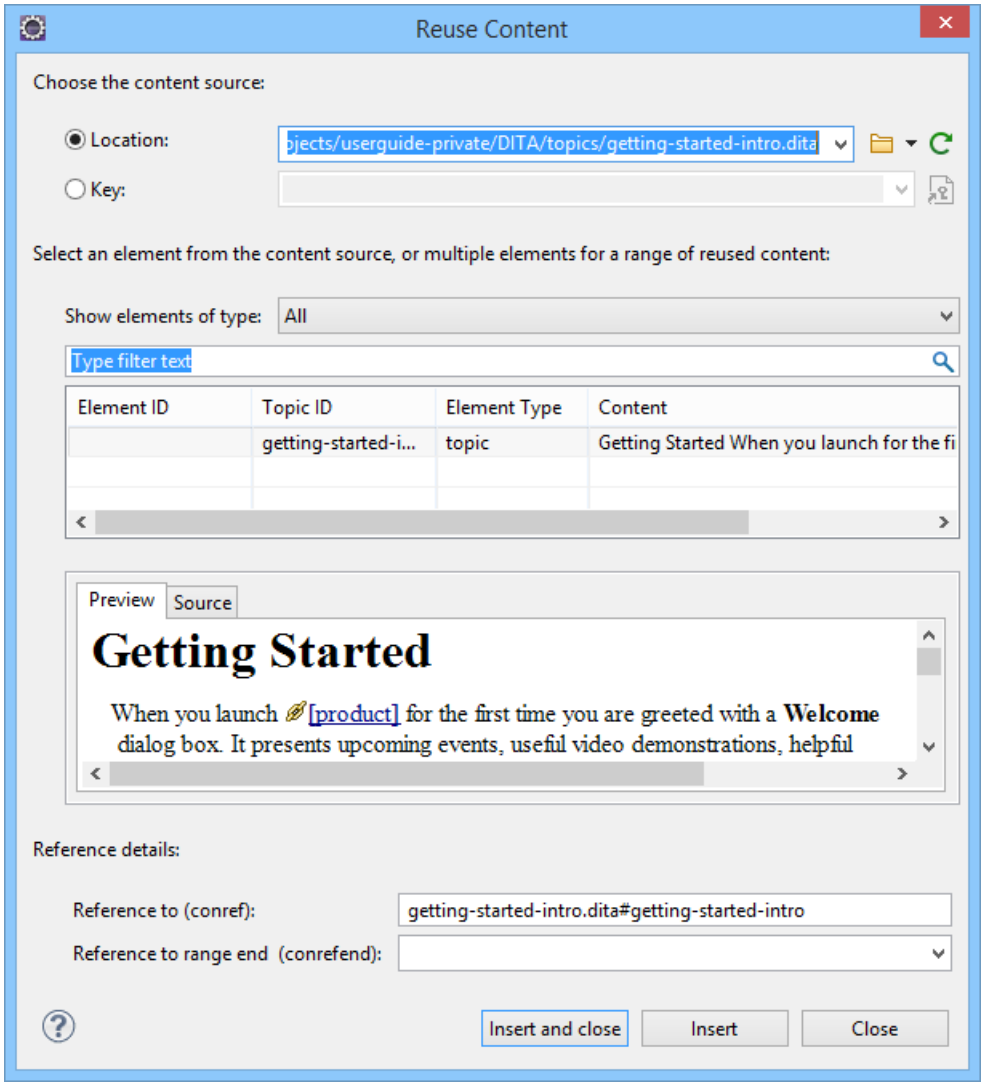


Figure 246: The Reuse Content Dialog Box (with the Default Insert Content Reference Options Displayed)

When **Location** is selected for the content source, a *content reference* (`conref`) will be inserted. Here you can specify the path of the topic that contains the content you want to reference.

The dialog box offers the following options:

Select an element from the content source Section

Show elements of type

You can use this drop-down list to select specific types of elements to be displayed in the subsequent table. This can help you narrow down the list of possible source elements that you can select.

Text Filter Field

You can also use the text filter field to narrow down the list of possible source elements to be displayed in the subsequent table.

Element Table

Present all the element IDs defined in the source topic. Use this table to select the **Element ID** of the element that you want to reference. You can select multiple contiguous elements to reference a block of content.

Preview Pane

Displays the content that will be references. If you select multiple elements in the element table, the content from all the selected elements is displayed.

Source Pane

Displays the source code of the element to be referenced.

Reference details Section

Reference to (conref)

Oxygen XML Editor plugin automatically fills this text field with the value of the `conref` attribute to be inserted. However, you can edit this value if need be.

Reference to range end (conrefend)

If you select multiple elements (of the same type) in the element table, Oxygen XML Editor plugin automatically fills this text field with the `id` value of the last element in your selection. This value will be inserted as a `conrefend` attribute, defining the end of the `conref` range.

Content Key Reference (`conkeyref`) Options Using the Reuse Content Dialog Box

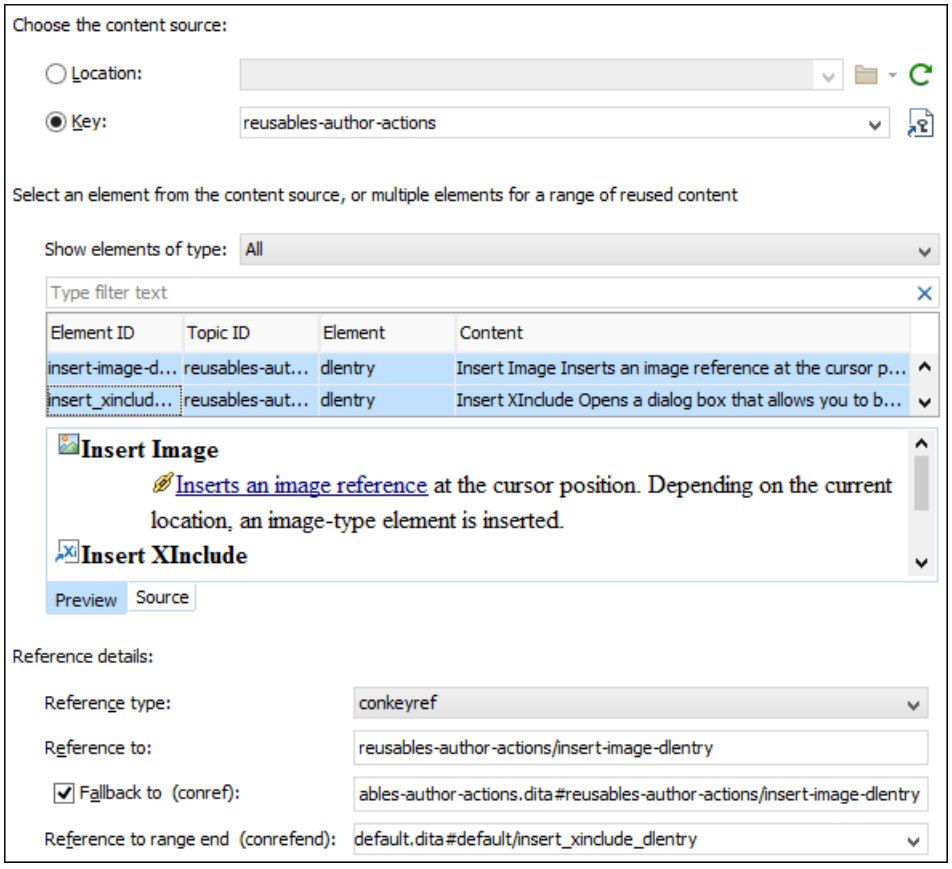


Figure 247: Insert Content Key Reference Options

Choose the content source Section

When **Key** is selected for the content source, you can use keys to reference content. You can use the **Choose Key Reference** button to open the **Choose Key** dialog box that allows you to select one from a list of all the keys that are gathered from the root map (you can also select one from the drop-down list in the **Key** field).

Note: If the current DITA map is not selected as the root map, no keys will be listed.

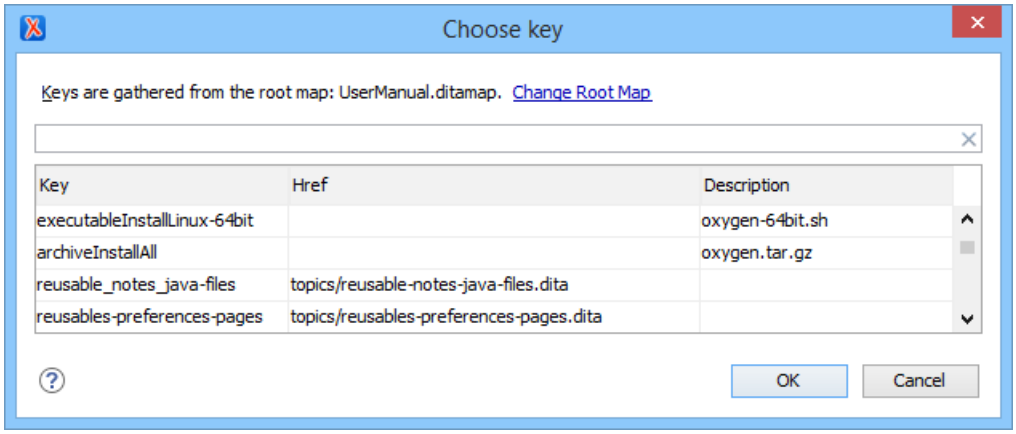


Figure 248: Choose Key Dialog Box

To insert a *content key reference* (`conkeyref`), select the key that contains the content you want to reference. Notice that the file path is shown in the **Href** column. Keys that do not have a value in the **Href** column are for referencing metadata with a `keyref` attribute. Therefore, to insert a `conkeyref`, you need to select a key that has a value (file path) in the **Href** column. After you select a key, click **OK** to return to the **Reuse Content** dialog box.

When a key that is defined as a *content key reference* has been selected, the **Reuse Content** dialog box offers the following additional options for inserting a `conkeyref`:

Select an element from the content source Section

Show elements of type

You can use this drop-down list to select specific types of elements to be displayed in the subsequent table. This can help you narrow down the list of possible source elements that you can select.

Text Filter Field

You can also use the text filter field to narrow down the list of possible source elements to be displayed in the subsequent table.

Element Table

Present all the element IDs defined in the source topic. Use this table to select the **Element ID** of the element that you want to reference. You can select multiple contiguous elements to reference a block of content.

Preview Pane

Displays the content that will be references. If you select multiple elements in the element table, the content from all the selected elements is displayed.

Source Pane

Displays the source code of the element to be referenced.

Reference details Section

Reference type

The type of reference that will be inserted. If you selected a key that references a DITA resource, you will notice that **conkeyref** value is automatically selected.

Reference to

Oxygen XML Editor plugin automatically fills this text field with the value of the `conkeyref` attribute to be inserted. However, you can edit this value if need be.

Fallback to (conref)

You can enable this option to define a `conref` attribute to be used as a fallback to determine the content reference relationship if the specified `conkeyref` cannot be resolved.

Reference to range end (conrefend)

If you select multiple elements (of the same type) in the element table, Oxygen XML Editor plugin automatically fills this text field with the `id` value of the last element in your selection. This value will be inserted as a `conrefend` attribute, defining the end of the `conkeyref` range.

Key Reference to Metadata (`keyref`) Options Using the Reuse Content Dialog Box

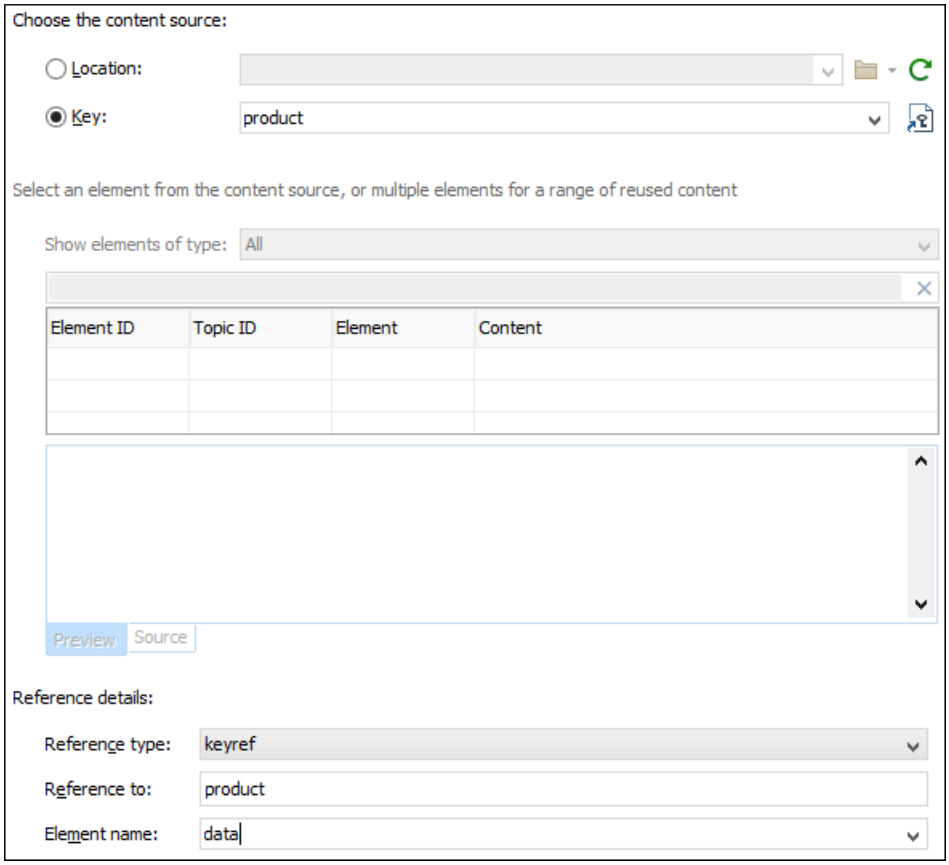


Figure 249: Insert Key Reference Options

Choose the content source Section

When **Key** is selected for the content source, you can use keys to reference content. You can use the **Choose Key Reference** button to open the **Choose Key** dialog box that allows you to select one from a list of all the keys that are gathered from the root map (you can also select one from the drop-down list in the **Key** field).

Note: If the current DITA map is not selected as the root map, no keys will be listed.

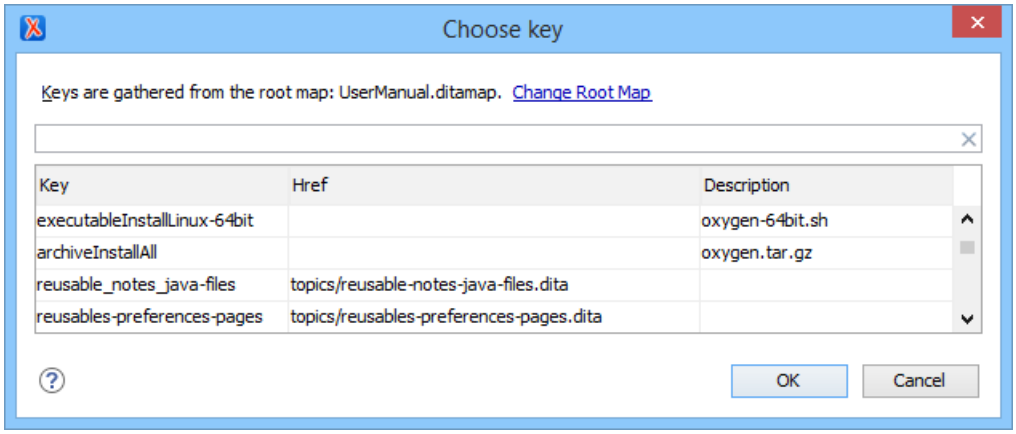


Figure 250: Choose Key Dialog Box

To insert a *key reference* to metadata (`keyref`), select the key you want to reference.

Note: Keys that do not have a value in the **Href** column are for referencing metadata with a `keyref` attribute, whereas keys for referencing a `conkeyref` have a value (file path) shown in the **Href** column.

Note: The **Description** column collects data from the definition of the key, either from the `navtitle` element or, if missing, from the `keyword` element. The following example shows two key definitions that will be collected in the keys table. Their corresponding information from the **Description** column will display `oxygen.sh` and `oxygen.tar.gz` respectively.

```
<keydef keys="executableInstallLinux">
  <topicmeta>
    <keywords>
      <keyword>oxygen.sh</keyword>
    </keywords>
  </topicmeta>
</keydef>

<keydef keys="archiveInstallAll">
  <topicmeta>
    <navtitle>
      oxygen.tar.gz
    </navtitle>
  </topicmeta>
</keydef>
```

After you select a key, click **OK** to return to the **Reuse Content** dialog box.

When a key that references metadata has been selected, the **Reuse Content** dialog box offers the following additional options for inserting a `keyref`:

Select an element from the content source Section

This section is not used when referencing metadata.

Reference details Section

Reference type

The type of reference that will be inserted. If you selected a key that does not reference a DITA resource, you will notice that **keyref** value is automatically selected.

Reference to

Oxygen XML Editor plugin automatically fills this text field with the value of the `keyref` attribute to be inserted.

Element name

Oxygen XML Editor plugin automatically selects the element that is most commonly used for the selected type of key reference, but you can use the drop-down list to choose another element to use for the reference.

Finalizing Your Content Reference Configuration

Once you click **Insert** or **Insert and close**, the configured content reference is inserted into your document.

Tip: You can easily insert multiple content references by keeping the **Reuse Content** dialog box opened, using the **Insert** button.

Working with the Conref Push Mechanism

Content Reference Push Mechanism

The usual method of using content references pulls element content from a source element and inserts it in the current topic. DITA 1.2 introduced an alternative method of content referencing, allowing element content to be pushed, or injected, from a source topic to another topic without any special coding in the topic where the content will be re-used. This technique is known as a content reference *push* mechanism (*conref push*).

The *conref push* mechanism requires elements in the target topic (the topic where the content is to be pushed) to have id elements, as the push mechanism inserts elements *before* or *after* a named element, or *replaces* the named element. Assuming the source topic is included in the DITA map, the *conref push* will be processed during publishing stage for the DITA map.

Example of a Conref Push Scenario

An example of a scenario in which a *conref push* would be useful is where a car manufacturer produces driver manuals that are distributed to various regions with their own specific regulations and certain sections need to be customized by the local car dealers before publishing. The local dealer could use a *conref push* technique to insert specific content without modifying the manufacturer-supplied content.

Push Current Element Action

Oxygen XML Editor plugin includes an action that allows you to easily reference content with a *conref push* mechanism. The **Push Current Element** action is available in the **DITA** menu and in the **Reuse** subfolder of the contextual menu when editing in **Author** mode. Selecting this action opens the **Push current element** dialog box that allows you to select a target resource and element, and where to insert the current element content.

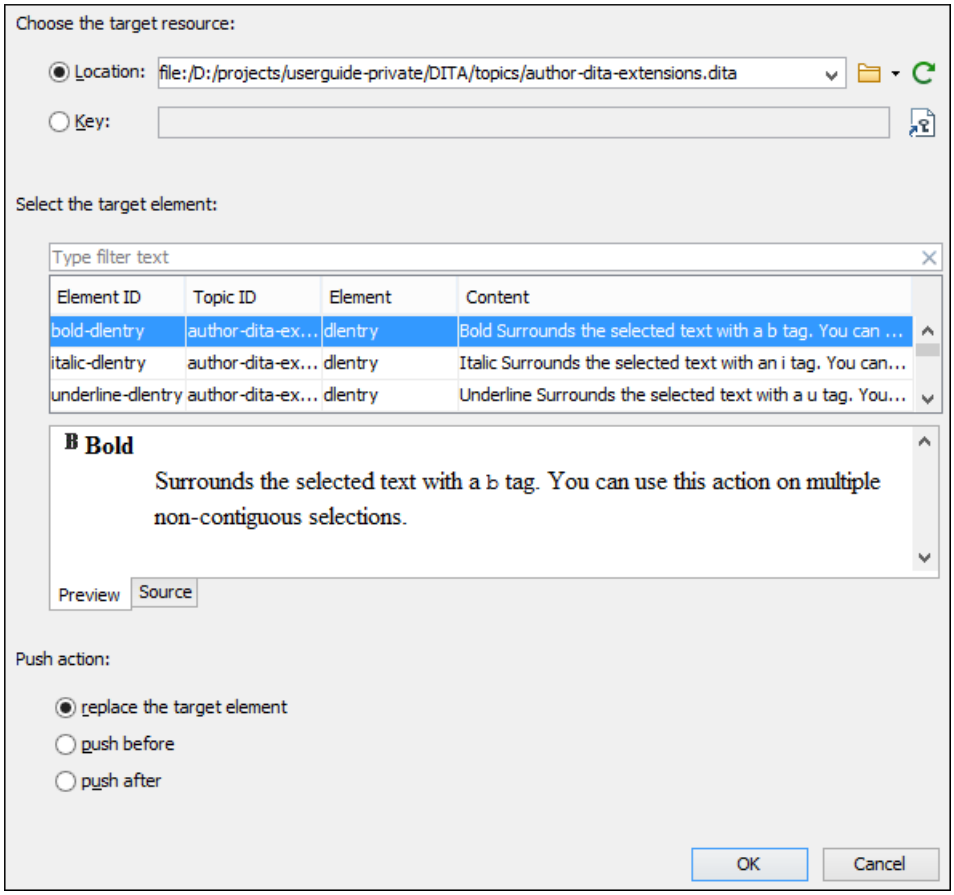


Figure 251: The Push Current Element Dialog Box

This dialog box allows you to configure the following options for the *conref push* action:

Choose the target resource

Allows you to select a **Location** URL or a **Key** for the target resource and the table in the next section of the dialog box will be populated using the information from the specified resource.

Select the target element

The table in this section contains the available elements (identified by their IDs) that belong to the same class as the current element on which the action was invoked.

Push action

Allows you to choose one of the following options for where you want to insert the current element content:

replace the target element

The target element will be replaced with the current element content.

On the technical side, the value of the `conaction` attribute in the current element will be set to `pushreplace` and the `conref` or `conkeyref` attribute will be set to the specified reference.

push before

The current element content will be inserted before the specified target element in the target resource.

On the technical side, the value of the `conaction` attribute in the current element will be set to `pushbefore`. Another element with the same name and class as the target element will be inserted in the document after the current element. The new element will have the `conaction` attribute set to `mark` and the `conref` or `conkeyref` attribute will be set to the specified reference.

push after

The current element content will be inserted after the specified target element in the target resource.

On the technical side, the value of the `conaction` attribute in the current element will be set to `pushafter`. Another element with the same name and class as the target element will be inserted in the document before the current element. The new element will have the `conaction` attribute set to `mark` and the `conref` or `conkeyref` attribute will be set to the specified reference.

You can also use the **Preview** panel to view the content that will be pushed and the **Source** panel to see the XML code for the content to be pushed. After you click **OK**, the *conref push* mechanism is inserted in the current document. The changes in the target resource will be processed when you transform the DITA map.

Working with Reusable Components

In DITA, the content of almost any element can be made reusable simply by adding an `id` attribute to the element. The DITA content reference mechanism can reuse any element with an *id*. However, it is not considered best practice to arbitrarily reuse pieces of text from random topics due to the difficulties this creates in trying to manage it. It also creates the possibility of authors deleting or changing content that is reused in other topics without being aware that the content is reused. To prevent these types of problems, you can create reusable components to manage a separate set of topics that contain topics designed specifically for reuse. Then, all of your reusable content can be referenced from the reusable components and if the content needs to be updated you only need to edit it in one place.

Oxygen XML Editor plugin allows you to select content in a topic, create a reusable component from it and reference that component in other locations by using the **Create Reusable Component** and **Insert Reusable Component** actions.

Creating a Reusable Content Component

Oxygen XML Editor plugin makes it easy to create reusable content components from existing topic content.

Note: To ensure that the topic file that contains the reusable component is a valid container for just the reusable content component, without having to include the other elements required by a standard topic type, Oxygen XML Editor plugin creates a specialized topic type on the fly. This specialization is designed to make sure that the content is compatible with the topic type from which it is created.

Follow these steps to create a reusable component:

1. In **Author** mode, select the content you want to make into a reusable component.
2. Select the **Create Reusable Component** action that is available in the **DITA** menu or the **Reuse** submenu of the contextual menu.
   The **Create Reusable Component** dialog box is displayed.
3. Use the **Reuse Content** drop-down list to select the scope of the content to be made reusable. It allows you to select how much of the current content you want to make reusable. The choices presented include the element at the current cursor position and its ancestor elements.
4. Add a description. This becomes the title of the topic that contains the reusable component, but is not part of the reusable content. It is just to help you identify the reusable content and will not become part of your output.
5. If you want to replace the extracted content with a reference to the new component you should leave the **Replace selection with content reference** option enabled. This is recommended, since the point of reuse is to maintain only one copy of the content.
6. Select a file name and location to save the topic containing the reusable component and click **Save**. It is considered best practice to save or store reusable components in an area set aside for that purpose.
   If the **Replace selection with content reference** option was enabled, Oxygen XML Editor plugin replaces the current content with a `conref` attribute. The content of the content reference will be displayed in your current topic with a gray background, but it will no longer be editable since it is stored in a separate file. To edit the source of the reusable component, click the **Edit Content** icon at the beginning of the inserted content.

You now have a reusable component that you can include in other topics by using the **Insert Reusable Component** action that is available in the **DITA** menu or the **Reuse** submenu of the contextual menu. You can also reference this new reusable component by using a *content reference* or *content key reference*.

Inserting a Reusable Content Component

Oxygen XML Editor plugin includes an **Insert Reusable Content** action that allows you to easily insert a reusable content component that you *created using the **Create Reusable Component** action*.

Caution: This action is only designed to insert reusable components created using the Oxygen XML Editor plugin **Create Reusable Component** action. It assumes certain things about the structure of the reusable content file that may not be true of reusable content created by other methods and it may not provide the expected results if used with content that does not have the same structure.

The **Insert Reusable Content** action creates a DITA `conref` to insert the content, and creates a parent element for the `conref` attribute based on the type of the reusable element in the reusable component file. This action ensures that the correct element is used to create the `conref`. However, that element must still be inserted at a point in the current topic where that element type is permitted.

Follow these steps to insert a reusable component that was created using the **Create Reusable Component** action:

1. Place the cursor at the insertion point where you want the reusable component to be inserted.
2. Select the **Insert Reusable Component** action that is available in the **DITA** menu or the **Reuse** submenu of the contextual menu.
   The **Insert Reusable Component** dialog box is displayed.
3. Locate the reusable content file in which you want to insert its content.
4. If you select **Content reference** in the **Insert as** drop-down list, the action will add a `conref` attribute to the DITA element at the current location. If you select **Copy** in the drop-down list, the content of the reusable component file will simply be pasted at the current location (assuming the content is valid at the current location).
5. Click **Insert** to perform the action.

Working with Variable Text in DITA

You may often find that you want a certain piece of text in a topic to have a different value in various circumstances. For example, if you are reusing a topic about a feature that is shared between several products, you might want to make the name of the product variable so that the correct product name is used in the manual for each product.

For example, you might have a sentence like this:

```
The quick-heat feature allows [product-name] to come up to temperature quickly.
```

You need a way to substitute the correct product name for each product.

One way to do this would be to use conditional profiling, as in this figure:

```
<p>The quick-heat feature allows
    <ph product="basic">Basic Widget</ph>
    <ph product="pro">Pro Widget</ph>
    <ph product="enterprise">Enterprise Widget</ph>
to come up to temperature quickly.</p>
```

Figure 252: Variable content using profiling

Creating Variable Text Using Keys

In DITA, you can create variable text using *keys*.

One way to do this would be to provide conditional values using the `product` profiling attribute.

However, this approach means that you are repeating the product names over and over again everywhere the product name is mentioned. This is time consuming for authors and will create a maintenance problem if the product names change.

Creating Variable Content Using a Key Reference

The alternative is to use a key reference, as in the following example:

```
<p>The quick-heat feature allows <ph keyref="product"/> to come up to temperature quickly.</p>
```

Figure 253: Variable content using a key reference

The key reference stands in for the name of the product. When the content is published, the current value of the key `product` will be inserted.

To insert a key reference into a document in Oxygen XML Editor plugin **Author** mode, follow these steps:

1. Press **Enter** and select any DITA element that supports the `keyref` attribute.
2. Select **Edit Attributes** from the contextual menu to bring up the attribute editor.
3. In the **Name** field, select `keyref`.
4. In the **Value** field, select or enter the name of the key.

Note: Additionally, if you have a need for reusing the key reference pattern while editing your documentation, you could add that pattern to a *code template* so that it appears in your list of content completion proposals.

Linking in DITA

DITA provides support for various types of linking between topics, some of which is automated, while others are specified by the author. Oxygen XML Editor plugin provides support for all forms of linking in DITA.

Linking Between Parent, Child, and Sibling Topics

A DITA map creates a hierarchical relationship between topics. That relationship map expresses a narrative flow from one topic to another, or it may be used as a classification system to help the reader find topics based on their classification, without creating a narrative flow. Since there may be various types of relationships between topics in a hierarchy, you may want to create links between topics in a variety of different ways. For instance, if your topics are supposed to be organized into a narrative flow, you may want to have links to the next and previous topics in that flow. If your topics are part of a hierarchical classification, you may want links from parent to child topics, and vice versa, but not to the next and previous topics.

Parent, child, and sibling links are created automatically by the DITA output transformations (and may differ between various output formats). The kinds of links that are created are determined by the DITA *`collection-type` attribute*.

In-Line Linking in the Content of a Topic

DITA supports linking within the text of a topic using the `xref` element. The destination of the link can be expressed directly using the `href` attribute or indirectly using the `keyref` attribute. If you use the `keyref` attribute, you link to a key rather than directly to a topic. That key is then assigned to a topic in a map that includes that topic. This means that you can change the destination that a key points to by editing the key definition in the map or by substituting a different map in the build.

Linking Between Related Topics

In addition to the relationships between topics that expressed by their place in the hierarchy of a map, a topic may be related to other topics in various ways. For instance, a task topic may be related to a concept topic that gives the background of the task, or to a reference topic that provides data needed to complete the task. Task topics may also be related to other tasks in a related area, or concepts to related concepts.

Typically, they are grouped in a list at the end of the topic, although this depends on the behavior of the output transformation. DITA provides two mechanisms for expressing relationships between topics at the topic level: the *related links* section of a topic and relationship tables in maps.

Managing Links

Links can break for a variety of reasons. The topic that a link points to may be renamed or removed. A topic may be used in a map that does not include a linked topic. A topic or a key may not exist in a map when a particular profile is applied. The **DITA Maps Manager** provides a way to *validate all the links in the documents that are included in the map*. This can include validating all the profiling conditions that are applied.

Hierarchical Linking in DITA Maps

To create hierarchical linking between the topics in a DITA map, you set the appropriate value of the `collection-type` attribute on the map. See the *DITA documentation* for the meaning of each of the values of the `collection-type` attribute.

Note: The decision for when and how to create hierarchical links is made by the publishing scripts. The `collection-type` attribute does not force a particular style of linking. Rather, it declares what the nature of the relationship is between the topics. The publishing scripts use that information to decide how to link topics. Scripts for different types of media may make different decisions depending on what is appropriate for the media. You can provide additional instructions to the scripts using the `linking` attribute.

To add the `collection-type` to an item in a map:

1. Right-click the topic and choose **Edit Properties**. The **Edit Properties** dialog box is displayed.
2. In the **Attributes** tab, select the appropriate value from the **Collection type** drop-down list.
3. You can use the **Other attributes** table to add a value to the `linking` attribute.

Linking in DITA Topics

Direct Links

You can create in-line links in the content of a DITA topic using the `xref` element. The destination of the link can be expressed directly by using the `href` attribute and the target can be another topic or a specific element within the other topic, another location within the same topic, a file, or a web link. You can also create direct *related links* to topics, files, or websites in a DITA topic using the `related-links` element.

Indirect Links Using Keys

The destination of the link can also be expressed indirectly by using *keys* to create either in-line links or *related links* (with the `keyref` attribute). By using keys, you avoid creating a direct dependency between topics. This makes links easier to manage and can make it easier to reuse topics in various publications. It can also be helpful in verifying the completeness of a publication, by ensuring that a publication map provides a key definition for every key reference used in the content.

Links based on keys require two pieces:

- Key Definition - Assigns a key to a topic so that other topics can link to it. For more information, see *Inserting and Defining Keys in DITA Maps* on page 422.
- Key Reference - Created in an `xref` element and specifies the key to link to.

The key reference points to a key definition, and the key definition points to a topic. Key definitions are created in maps, as an element on the `topicref` element that points to a topic. This allows you to assign a particular key to one topic in one map and to another topic in another map. When a topic that links to that key is used in each of these maps, the links work correctly in both maps.

Inserting a Link in Oxygen XML Editor plugin

To insert a link in ***Author*** *mode*, use the actions available in the **Link** drop-down menu from the toolbar (or the **Link** submenu in the contextual menu or **DITA** menu). You can choose between the following types of in-line links:

Cross Reference

Opens the **Cross Reference (xref)** dialog box that allows you to insert a link to a target resource at the current location within a document. The target resource can be the location of a file or a key that is already defined in your DITA map structure. Once the target resource has been selected, you can also target specific elements within that resource. Depending on the context where it is invoked, the action inserts one of the following two elements:

- `xref` - Used to link to other topics or another location within the same topic and points to the target using the `href` attribute.

- `fragref` - A logical reference to a fragment element within a syntax diagram and points to the target using the `href` attribute.

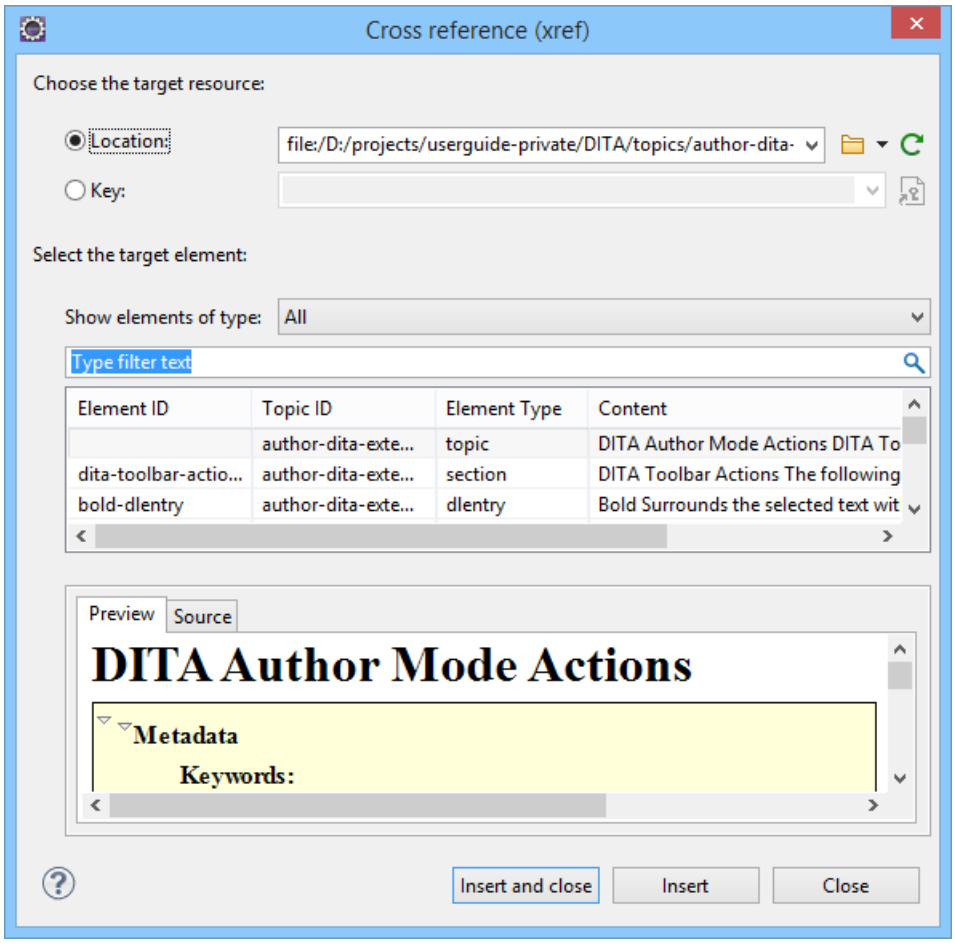


Figure 254: The Cross Reference (xref) Dialog Box

This dialog box includes the following sections and fields:

Choose the Target Resource Section

Location - If you select **Location** for the target, the link is expressed in an `href` attribute.

Key - If you select **Key** for the target, keys will be used to express the link in a `keyref` attribute. You can use the  **Choose Key Reference** button to open the **Choose Key** dialog box that allows you to select one from a list of all the keys that are gathered from the root map (you can also select one from the drop-down list in the **Key** field).

Select the Target Element Section

This section can be used to target a specific element inside the target resource.

Show elements of type

You can use this drop-down list to select specific types of elements to be displayed in the subsequent table. This can help you narrow down the list of possible source elements that you can select.

Text Filter Field

You can also use the text filter field to narrow down the list of possible source elements to be displayed in the subsequent table.

Element Table

Present all the element IDs defined in the source topic. Use this table to select the **Element ID** of the element that you want to reference.

Preview Pane

Displays the content that will be references.

Source Pane

Displays the XML source code of the element to be referenced.

Once you click **Insert** or **Insert and close**, the configured cross reference is inserted into your document.

Tip: You can easily insert multiple cross references by keeping the dialog box opened, using the **Insert** button.

File Reference

Opens the **File Reference** dialog box that allows you to insert a link to a target file resource at the current location within a document. The target resource can be the location of a file or a key that is already defined in your DITA map structure. It inserts an `xref` element with the value of the `format` attribute set to `xml`.

Choose the Target Resource

Location - If you select **Location** for the target file, the link is expressed in an `href` attribute.

Key - If you select **Key** for the target file, keys will be used to express the link in a `keyref` attribute. You can use the **Choose Key Reference** button to open the **Choose Key** dialog box that allows you to select one from a list of all the keys that are gathered from the root map and defined as a non-DITA resource (you can also select one from the drop-down list in the **Key** field).

Web Link

Opens the **Web Link** dialog box that allows you to insert a link to a target web-related resource at the current location within a document. The target resource can be a URL or a key that is already defined in your DITA map structure. It inserts an `xref` element with the value of the `format` attribute set to `html`, and `scope` set to `external`.

Choose the Target Web Resource

URL - If you select **URL** for the target resource, the link is expressed in an `href` attribute.

Key - If you select **Key** for the target resource, keys will be used to express the link in a `keyref` attribute. You can use the **Choose Key Reference** button to open the **Choose Key** dialog box that allows you to select one from a list of all the keys that are gathered from the root map and defined as a non-DITA resource (you can also select one from the drop-down list in the **Key** field).

Related Link to Topic

Opens the **Cross Reference (xref)** dialog box that allows you to insert a link to a target resource in a related links section that is typically at the bottom of your topic (although this depends on the behavior of the output transformation). The target resource can be the location of a file or a key that is already defined in your DITA map structure. Once the target resource has been selected, you can also target specific elements within that resource. If a related links section does not already exist, this action creates one. Specifically, it inserts a `link` element inside a *`related-links` element*.

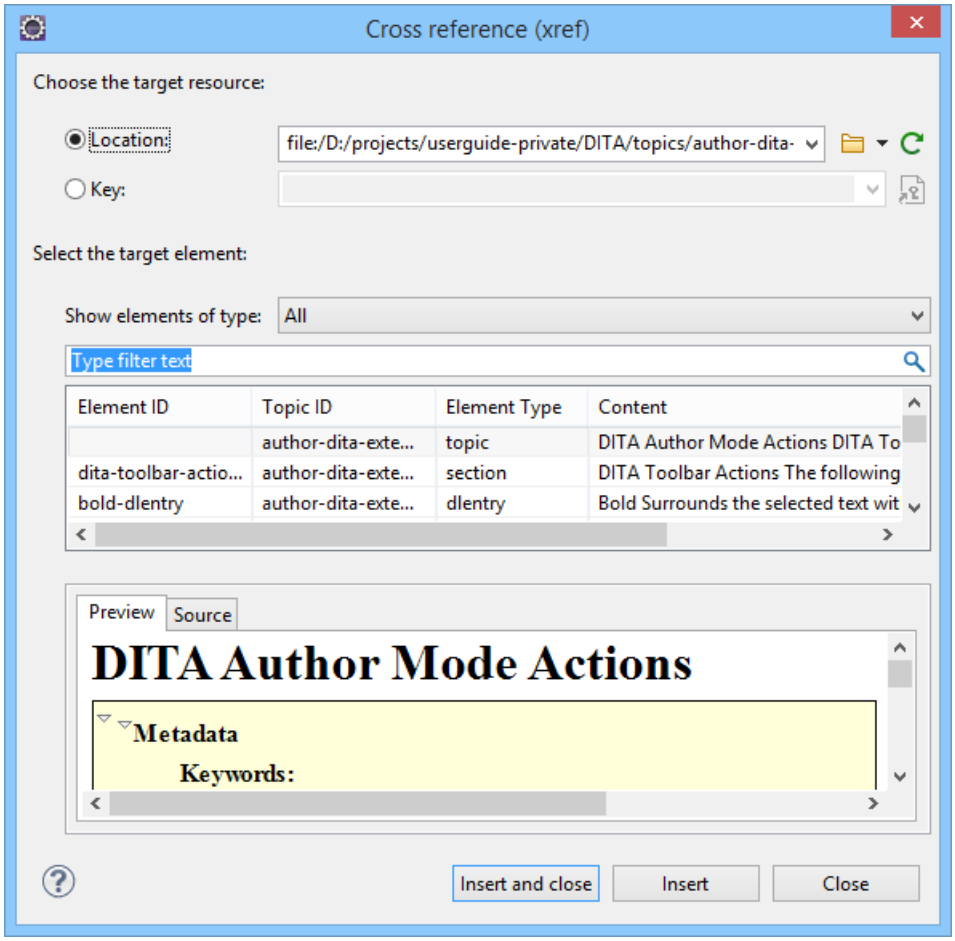


Figure 255: The Cross Reference (xref) Dialog Box

This dialog box includes the following sections and fields:

Choose the Target Resource Section

Location - If you select **Location** for the target, the link is expressed in an `href` attribute.

Key - If you select **Key** for the target, keys will be used to express the link in a `keyref` attribute. You can use the **Choose Key Reference** button to open the **Choose Key** dialog box that allows you to select one from a list of all the keys that are gathered from the root map (you can also select one from the drop-down list in the **Key** field).

Select the Target Element Section

This section can be used to target a specific element inside the target resource.

Show elements of type

You can use this drop-down list to select specific types of elements to be displayed in the subsequent table. This can help you narrow down the list of possible source elements that you can select.

Text Filter Field

You can also use the text filter field to narrow down the list of possible source elements to be displayed in the subsequent table.

Element Table

Present all the element IDs defined in the source topic. Use this table to select the **Element ID** of the element that you want to reference.

Preview Pane

Displays the content that will be references.

Source Pane

Displays the XML source code of the element to be referenced.

Once you click **Insert** or **Insert and close**, the configured cross reference is inserted into your document.

Tip: You can easily insert multiple cross references by keeping the dialog box opened, using the **Insert** button.

Related Link to File

Opens the **File Reference** dialog box that allows you to insert a link to a target file resource in a related links section that is typically at the bottom of your topic (although this depends on the behavior of the output transformation). The target resource can be the location of a file or a key that is already defined in your DITA map structure. If a related links section does not already exist, this action creates one. Specifically, it inserts a `link` element with the `format` attribute set to `xml` inside a *`related-links` element*.

Choose the Target Resource

Location - If you select **Location** for the target file, the link is expressed in an `href` attribute.

Key - If you select **Key** for the target file, keys will be used to express the link in a `keyref` attribute. You can use the **Choose Key Reference** button to open the **Choose Key** dialog box that allows you to select one from a list of all the keys that are gathered from the root map and defined as a non-DITA resource (you can also select one from the drop-down list in the **Key** field).

Related Link to Web Page

Opens the **Web Link** dialog box that allows you to insert a link to a target web-related resource in a related links section that is typically at the bottom of your topic (although this depends on the behavior of the output transformation). The target resource can be a URL or a key that is already defined in your DITA map structure. If a related links section does not already exist, this action creates one. Specifically, it inserts a `link` element with the attribute `format` set to `html` and `scope` set to `external` inside a *`related-links` element*.

Choose the Target Web Resource

URL - If you select **URL** for the target resource, the link is expressed in an `href` attribute.

Key - If you select **Key** for the target resource, keys will be used to express the link in a `keyref` attribute. You can use the **Choose Key Reference** button to open the **Choose Key** dialog box that allows you to select one from a list of all the keys that are gathered from the root map and defined as a non-DITA resource (you can also select one from the drop-down list in the **Key** field).

Linking with Relationship Tables in DITA

A relationship table is used to express relationships between topics outside of the topics themselves. The DITA publishing scripts can then create links between related topics when the content is published.

The reason for using a relationship table is to help make topics easier to reuse. If a topic links directly to another topic, this creates a dependency between the topics. If one topic is reused in a publication where the other is not used, the link is broken. By defining relationships between topics in a relationship table, you avoid creating this dependency.

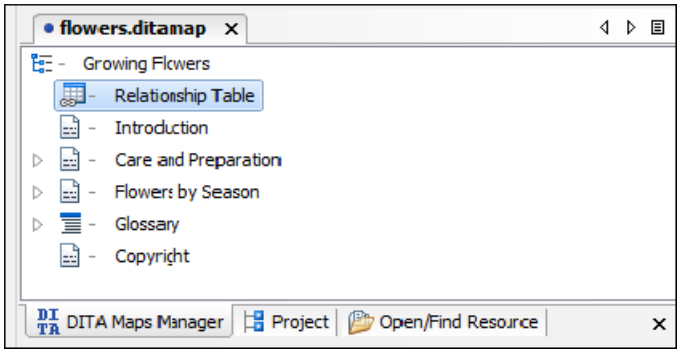
To create an appropriate set of links between topics in multiple publications, you can create a separate relationship table for each publication. If you are creating a variety of different publications by applying profiling conditions to a single map, you can also profile your relationship table.

Follow these steps to create a relationship table:

1. If the map is currently open in the **DITA Maps Manager**, double-click the map icon to open the map in **Author** mode. If it opens in **Text** mode, click **Author** at the bottom left to switch to **Author** mode.
2. Move the insertion point inside the *map* root element (usually `map`, but it might be `bookmap`, or another specialization of the `map` element). The easiest way to do this is to click below the title of the map in the editor and then press the up arrow once. Confirm that you are inside the *map* root element by checking the breadcrumbs at the top left of the editor window. You should only see the name of the *map* root element.

3. Select the **Insert Relationship Table** action on the toolbar or from the **Relationship Table** submenu of the contextual menu. The **Insert Relationship Table** dialog box is displayed.
4. Make the appropriate selections in the **Insert Relationship Table** dialog box. See the *DITA documentation* for a full explanation of the relationship table format and its options. Note that you can change all the selections that you make here later by using the actions on the toolbar (or in the **Relationship Table** submenu of the contextual menu) or by editing the underlying XML in **Text** mode.
5. To add topic references to your relationship table, drag and drop topics from the **DITA Maps Manager** or the **Project** view into the appropriate cell in the relationship table.

   Relationship tables are also displayed in the **DITA Maps Manager** view, along with the other elements in its DITA map.



   You can open the DITA map to edit the relationship table by doing one of the following:

   - Double-click the appropriate relationship table in the **DITA Maps Manager**.
   - Select the relationship table in the **DITA Maps Manager** and press **Enter**.
   - Select **Open** from the contextual menu of the relationship table in the **DITA Maps Manager**.

Creating Relationship Tables

You can define relationships between topics in a relationship table. A relationship table is created inside a *DITA map*.

1. If the map is currently open in the **DITA Maps Manager**, double-click the map icon to open the map in **Author** mode. If it opens in **Text** mode, click **Author** at the bottom left to switch to **Author** mode.
2. Go to **DITA** > **Relationship Table** > **Insert Relationship Table**.
   The **Insert Relationship Table** dialog box 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 select ***Insert Reference*** from the contextual menu or the **DITA Map** toolbar.
6. To add a new row to the table or remove an existing row use **Insert Relationship Row**/**Delete Relationship Row** from the contextual menu or the **DITA Map** toolbar.
7. To add a new column to the table or remove an existing column, use **Insert Relationship Column**/**Delete Relationship Column** contextual menu or the **DITA Map** toolbar. If you double-click the relationship table (or select it and press **Enter**, or choose **Open** from the contextual menu) the DITA map is opened in the editor with the cursor positioned inside the corresponding relationship table.

Note: When the map is open in the **DITA Maps Manager**, the newly created relationship table is also displayed there. If you double-click the relationship table (or select it and press **Enter**, or choose **Open** from the contextual menu) the DITA map will be opened in the editor with the cursor positioned inside the corresponding relationship table.

Adding Images to a DITA Topic

There are several ways to add images to a DITA topic, depending on if you want to create a figure element (with a title and caption) or just insert an image inline, and if you want to use different versions of a graphic in various different situations. For instance, you might want to use a specific image for each different product version or output media.

Adding an Image Inline

Use the following procedure to add an image inline:

1. Place the cursor in the position you want the graphic to be inserted.
2. Select the **Insert Image** action. The **Insert Image** dialog box appears.

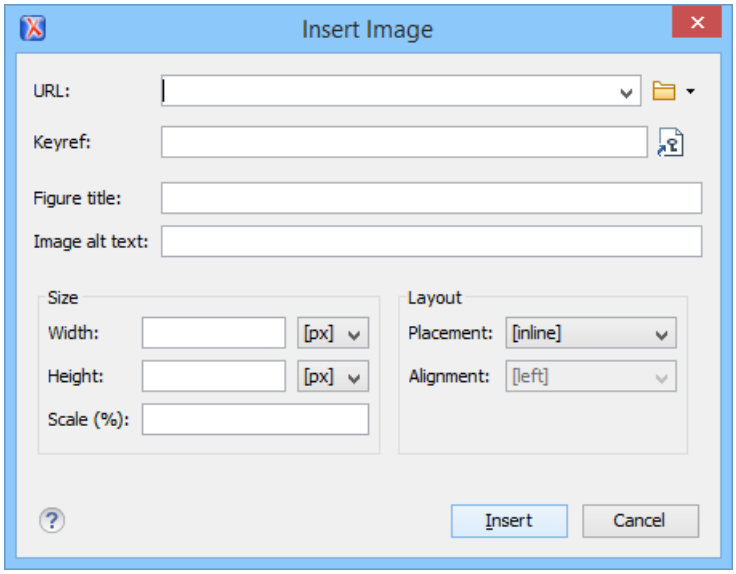


Figure 256: The Insert Image Dialog Box

3. Configure the options in this dialog box and click **Insert**.

   The **Insert Image** dialog box includes the following options for inserting images into a DITA document:

   URL

   Inserts an `image` element with an `href` attribute. You can type the URL of the image you want to insert or use the **Browse** drop-down menu to select an image using one of the following options:

   - **Browse for local file** - Displays the **Open** dialog box to select a local file.
   - **Browse workspace** - Opens a file browser dialog box that allows you to select a file from the local workspace.
   - **Browse for remote file** - Displays the **Open URL** dialog box to select a remote file.
   - **Browse for archived file** - Opens the **Archive Browser** to select a file from an archive.
   - **Browse Data Source Explorer** - Opens the **Data Source Explorer** to select a file from a connected data source.
   - **Search for file** - Displays the **Find Resource** dialog box to search for a file.

   Keyref

   You can use the **Choose Key Reference** button to open the **Choose Key** dialog box that presents the list of keys available in the selected root map. Use this dialog box to insert an `image` element with a `keyref` attribute. All keys that are presented in the dialog box are gathered from the root map of the current DITA map. Elements

that have the `keyref` attribute set are displayed as links. For more information, see the *Adding an Image Using a Key Reference* on page 457 section.

Note: If your defined keys are not listed in this dialog box, it is most likely trying to gather keys from the wrong root map. You can change the root map by using the **Change Root Map** link.

Figure title

Use this text box to insert a `title` and `image` element inside a `fig` element.

Alternate text

Use this text box to insert an `alt` element inside the `image` element.

Size

Use this section to configure the **Width** and **Height** of the image, or **Scale** the image. Specifying a value in these options inserts a `width`, `height`, and `scale` attribute, respectively.

Layout

Use the options in this section to insert `placement` and `align` attributes into the `image` element.

Adding an Image in a Figure Element

To add an image in a figure:

1. Add a `fig` element to your document at the appropriate place.
2. Add a `title` and/or `desc` element to the `fig` element, according to your needs.
3. *Add an `image` element* to the `fig` element.

Note: The `fig` element has a number of other child elements that may be appropriate to your content. See the *DITA documentation* for complete information about the `fig` element.

Note: The order in which the `image`, `title`, and `desc` content are presented in output is determined by the output transformation. If you want to change how they are output, you may have to modify the output transformation, rather than your source content.

Adding an Image Using a Key Reference

If you want to use a different version of the image in various situations, such as screenshots for multiple platforms or different formats for various types of output media, you can reference the image using a key reference:

1. Create a DITA map to hold your image keys. You can create one map for each use or create a single map and profile the key definitions for multiple uses. For instance, you might create one map of images to be used in PDF and one for images to be used in Web output, or you might use the platform profiling attribute to manage different versions of a screenshot (one for Macintosh and another for the Windows version of your product).
2. For each image, create a `keydef` element with the following structure:

```
<keydef keys="image.test" href="img/test.png" format="png">
```

Tip: You can easily create a `keydef` element that targets an image by using the ***Key Definition*** *action from the* ***Append Child*** *or* ***Insert After*** *submenus*.

3. If you are using profiling, add the alternative `keydef` elements and the appropriate profiling attributes:

```
<keydef keys="image.test" href="img/win/test.png" platform="windows" format="png">
<keydef keys="image.test" href="img/mac/test.png" platform="mac" format="png">
```

Tip: If you create the `keydef` element using the ***Key Definition*** *action*, you can use the ***Profiling*** *tab of the* ***Insert Reference*** *dialog box* to easily add profiling attributes to the target.

4. If you are using separate maps, repeat in each map. For instance, if you are using a separate map for images in PDF output, add a topic ref to that map like this:

```
<keydef keys="image.test" href="img/test.eps"  format="eps">
```

5. To insert an image by key, insert an `image` element and use a `keyref` attribute to point to the image:

```
<image keyref="image.test"/>
```

   Tip: You can also use the ***Keyref*** *section of the* ***Insert Image*** *dialog box* to insert a `keyref` attribute inside an `image` element.

   Oxygen XML Editor plugin displays the image in ***Author*** *mode*. Which image is displayed depends on the current profiling set that is applied and which *root map* is being used to resolve references.
6. Configure your build so that the appropriate image map is included for each output type and/or the appropriate profiling conditions are applied to each output.

Adding Tables to a DITA Topic

You can add a table to a DITA topic. By default, DITA supports two types of tables:

- DITA simple table model - This is the most commonly used model for basic tables.
- OASIS Exchange Table Model (a subset of the CALS table model) - This is used for more advanced functionality.

If you are using a specialized DITA vocabulary, it may contain specialized versions of these table models.

Since DITA is a structured format, you can only insert a table in places in the structure of a topic where tables are allowed. The Oxygen XML Editor plugin DITA toolbar provides support for entering and editing tables. It also helps to indicate where you are allowed to insert a table or its components by disabling the appropriate buttons.

The **DITA** toolbar showing the insert table button available and the table editing buttons are disabled:

Figure 257: The DITA Toolbar

Inserting a DITA Simple Table

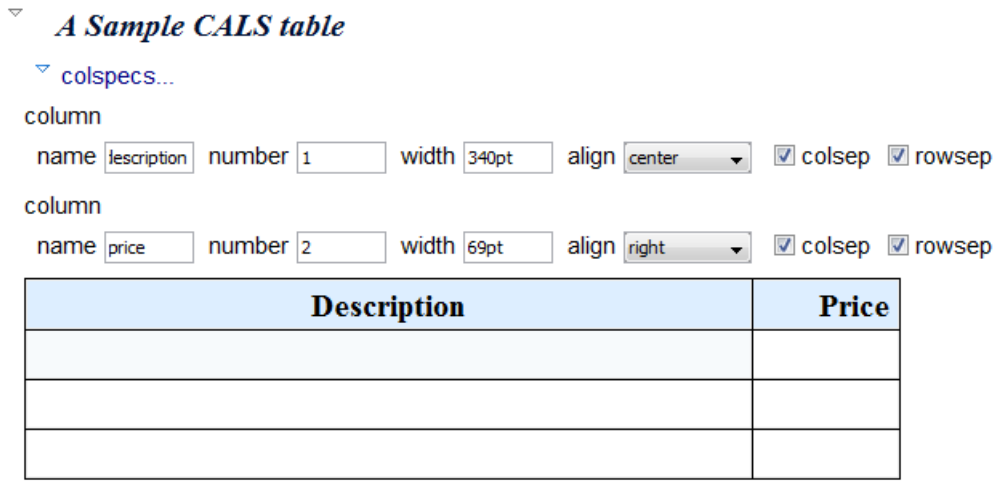
1. To insert a DITA simple table, select the **Insert Table** action on the **DITA** toolbar or in the **Insert** submenu from the contextual menu (or the **Table** submenu from the **DITA** menu). The **Insert Table** dialog box appears.
2. Select the **Simple** model.
3. Select the appropriate options for rows and columns, creating a header, column widths, and framing of cells.
4. Click **Insert**. The table is inserted.

Inserting an OASIS Exchange Table Model (CALS) Table

To insert an OASIS Exchange Table, follow the same procedure as above, but choose the CALS model. Additional options for specifying row and column separators and alignment will be enabled.

When you insert a CALS table, you see a link for setting the `colspecs` (column specifications) of your table. Click the link to open the `colspec` controls. Although they appear as part of the ***Author*** *mode*, the `colspec` link and the

`colspec` controls will not appear in your output. They are just there to make it easier to adjust how the columns of your table are formatted.



Editing an Existing Table

You can edit the structure of an existing table using the table buttons on the **DITA** toolbar (or in the **Table** submenu of the contextual menu or **DITA** menu) to add or remove cells, rows, or columns, and to set basic table properties. Additional attributes can be used to fine-tune the formatting of your tables by using the ***Attributes** view* (**Window** > **Show View** > **Attributes**). See the *DITA documentation* for a full explanation of these attributes.

Also, remember that underneath the visual representation, both table models are really just XML, and, if necessary, you can edit the XML directly by switching to ***Text** mode*.

Adding MathML Equations in DITA

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 from the **New** file action wizard.
- Reference an external MathML file as an image, using the **Insert Image** action that is available on the DITA toolbar (or from the **DITA** > **Insert** menu).

Note: 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, you must employ additional customizations for handling MathML content.

Working with Keys

DITA uses keys to insert content that may have different values in various circumstances. Keys provide a way to reference something indirectly. This can make it easier to manage and to reuse content in a number of ways.

You can think of keys like renting a post office box. Instead of the mail going directly from the sender to your house, it now goes to the post office box. You then go to the post office box and bring the mail back to your house. If you move to a new house, your mail still gets to you because it comes to the same post office box. You do not have to send change of address cards to all the people who send you mail. Your mailbox address is the key that makes sure your mail always reaches you, even if you move.

Similarly, if you use keys in your content to reference other content, you do not have to update the source content in order to change the value of the key or what it points to. You just change the definition of the key.

Keys are thus a very general mechanism. DITA uses keys for referencing different kinds of content for various purposes.

Using Keys for Values

You can use keys to represent *values that may vary in different outputs*. For instance, you may have several products that share a common feature. When you want to describe that feature, you need a way to insert the name of the product, even though that name is different depending on which product the feature description is being used for.

Assigning Keys to Topics

You can assign a key to a topic and use that key to reference that topic for various purposes, such as reuse or linking. As always, keys are defined in maps, so the key definition is done using the keys attribute of the `topicref` element:

```
<topicref href="quick-heat.dita" keys="feature.quick-heat"/>
```

You can also assign keys to a topic (and insert the `topicref` element in its DITA map) by using the **Keys** tab in the **Edit Properties** dialog box. In the **DITA Maps Manager**, invoke the contextual menu on the topic for which you want to assign a key and select **Edit Properties**. Go to the **Keys** tab and enter the name of the key in the **Define keys** field.

Once a key is assigned to a topic, you can use it to reference that topic for various purposes:

- You can *create a link* to it using `<xref keyref="feature.quick-heat"/>`. This allows you to change the target of the link by changing the topic that is pointed to by the key (for example, by profiling).
- You can use it *in a map to create a reference to a topic* by key: `<topicref keyref="feature.quick-heat"`. This allows you to change which topic is inserted in the map by the build, by changing the topic that is pointed to by the key.
- You can use it to *insert a content reference*. In this case, the content reference uses the key to locate the topic to pull content from. It uses a `conkeyref` attribute: `<procedure conkeyref="feature.quick-heat/preheat-procedure"/>`. In this example, `feature.quick-heat` is the key, and `preheat-procedure` is the id of a procedure within the topic for that key. Using this mechanism, you could have multiple versions of the preheat procedure in various topics and control which one is inserted by changing the topic that is pointed to by the key.

Assigning Keys to Graphics

You can assign a key to an image (using a map to point to the image file) and *insert the image using the key*.

Publishing DITA Output

In DITA, you create output by running a transformation on a DITA map. Transformations for various types of output are provided by the DITA Open Toolkit. Oxygen XML Editor plugin provides support for configuring and running transformation using transformation scenarios.

Note: Oxygen XML Editor plugin does not create any output formats itself. Oxygen XML Editor plugin runs externally defined transformations that produce output, and displays the result in the appropriate application, but the output itself is produced by the external transformation, not by Oxygen XML Editor plugin.

Customizing Outputs

You can customize the appearance of any of the output types by customizing the output transformations. There are two ways to do this:

1. Most transformations are configurable by passing parameters to the transformation script. Oxygen XML Editor plugin allows you to set parameters on a transformation scenario and save the parameters for future use. It can also allow you to prompt for a parameter whenever a transformation scenario is run. You can set up more than one transformation scenario for a given output type, which allows you to maintain several customized transformation scenarios for different types of output configurations.
2. If you want to customize an output in a way not supported by the customization options, you can create a modified version of the transformation code. Some transformation scripts export specific forms of extension or customization.

You should consult the SPFE Open Toolkit for the transformation type that you are interested in to see what customization options it supports. Oxygen XML Editor plugin provides full editing and debugging support from XSLT and CSS stylesheets, which you can use to modify transformation code.

You can also write your own output transformation scripts to produce a type of output not supported by the DITA Open Toolkit. Oxygen XML Editor plugin provides a full development environment for developing such transformations. You can create Oxygen XML Editor plugin transformation scenarios to run these scripts once they are complete.

Generating Output from DITA Content

As a structured writing format, DITA produces structured content (content that is annotated with specific structural and semantic information rather than with formatting information). To create a publication, your DITA map and its associated topics must be processed by a transformation script. That script is responsible for how the structural and semantic information in the DITA files is converted into formatting information for display.

This means that you can display the same DITA content in various different ways, media, or publications. It also means that you cannot control every aspect of the presentation of your content in your DITA files. The only way to change the formatting is to change the transformation routines that create it.

Therefore, to create output from your DITA content you have to run a transformation on your content. Oxygen XML Editor plugin provides a mechanism called transformation scenarios to help you configure and run transformations.

To select and run a transformation scenario on your map:

1. Click the **Configure Transformation Scenario(s)** button. The **Configure Transformation Scenario(s)** dialog box appears. This dialog box lists all the transformation scenarios that have been configured in your project. Oxygen XML Editor plugin provides a default set of transformation scenarios, but the people in charge of your DITA system may have provided others that are specifically configured for your needs.
2. Select the transformation scenario you want to run and click **Apply Associated**. The transformation scenario runs in the background. You can continue to work in Oxygen XML Editor plugin while the transformation is running. If there are errors or warnings, Oxygen XML Editor plugin displays them when the transformation is complete. If the transformation is successful, Oxygen XML Editor plugin opens the output in the appropriate application.
3. To rerun the same scenario again, click the **Apply Transformation Scenario(s)** button.

Transforming DITA Content

Oxygen XML Editor plugin uses the DITA Open Toolkit (DITA-OT) to transform DITA maps and topics into an output format. For this purpose, both the DITA Open Toolkit and Ant come bundled in Oxygen XML Editor plugin.

More information about the DITA Open Toolkit are available at *http://dita-ot.sourceforge.net/*.

DITA OT Transformation

To create a **DITA OT Transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **DITA OT Transformation**.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **DITA OT Transformation**.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **DITA OT Transformation**.

  Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **DITA Transformation Type** dialog box that presents the list of possible outputs.

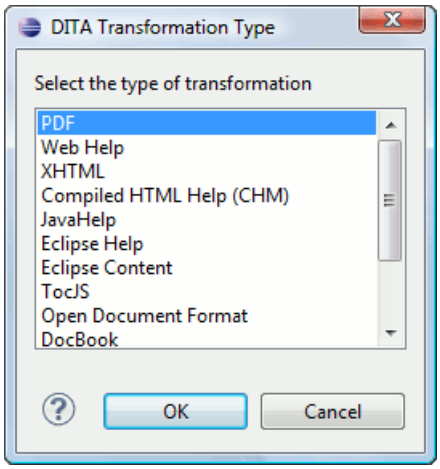


Figure 258: DITA Transformation Type Dialog Box

Select the desired type of output and click **OK**. This opens the **New Scenario** dialog box, which allows you to configure the options that control the transformation.

The lower part of the dialog box contains the following tabs (only those that are appropriate for the chosen output type will be displayed):

- **Skins** (Available for **WebHelp** and **WebHelp with Feedback** output types).
- **FO Processor** (Available for PDF output types).
- **Parameters**
- **Filters**
- **Advanced**
- **Output**

For information about creating an entirely new DITA OT transformation, see *Creating a DITA OT Customization Plugin* on page 478 and *Installing a Plugin in the DITA Open Toolkit* on page 480.

The FO Processor Tab

This tab allows you to select an FO Processor, when you choose to generate PDF output.

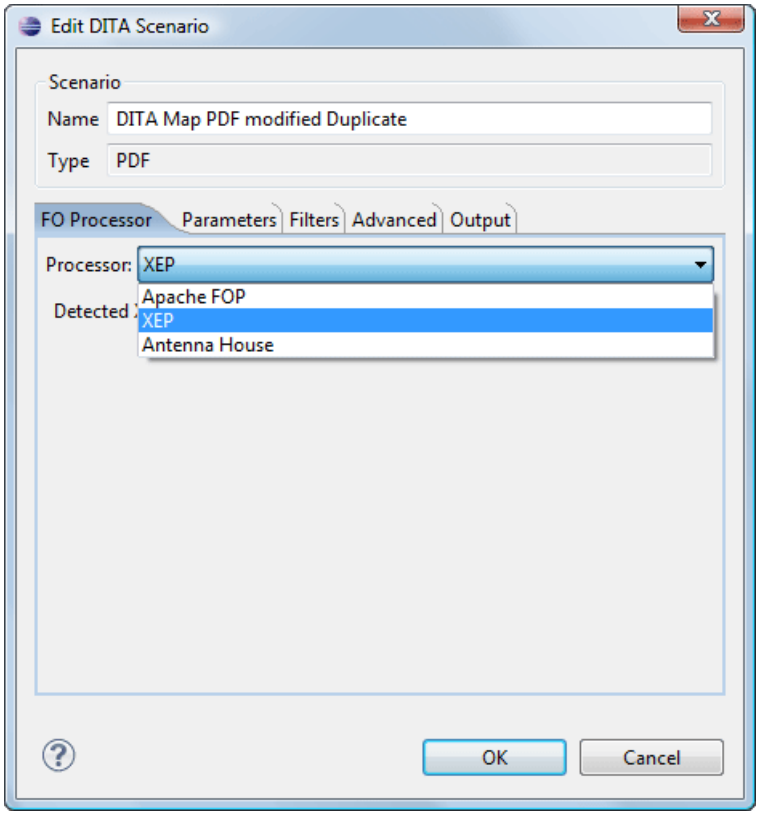


Figure 259: FO Processor Configuration Tab

You can choose the following processors:

- **Apache FOP** - The default processor that comes bundled with Oxygen XML Editor plugin.
- **XEP** - The *RenderX* XEP processor.

  If XEP is already installed, Oxygen XML Editor plugindisplays the detected installation path under the drop-down menu.

  XEP is considered installed if it was detected in one of the following sources:

  - XEP was configured as an external FO Processor in the ***FO Processors** option page*.
  - The system property *com.oxygenxml.xep.location* was set to point to the XEP executable file for the platform (for example: `xep.bat` on Windows).
  - XEP was installed in the *`DITA_OT_DIR`*`/plugins/org.dita.pdf2/lib` directory of the Oxygen XML Editor plugin installation directory.

- **Antenna House** - The *Antenna House* (AH Formatter) processor.

  If Antenna House is already installed, Oxygen XML Editor plugindisplays the detected installation path under the drop-down menu.

  Antenna House is considered installed if it was detected in one of the following sources:

  - Environment variable set by Antenna House installation (the newest installation version will be used).
  - Antenna House was added as an external FO Processor in the Oxygen XML Editor plugin preferences pages.

  To further customize the PDF output obtained from the Antenna House processor:

  - **Edit** the transformation scenario.
  - Open the ***Parameters** tab*.
  - Add the `env.AXF_OPT` parameter and point to Antenna House configuration file.

The Parameters Tab

The **Parameters**tab allows you to configure the parameters sent to the DITA-OT build file.

The table displays all the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (for example: XHTML or PDF), along with their description and current values. You can find more information about each parameter in the *DITA OT Documentation*. You can also add, edit, and remove parameters. Use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

Depending on the type of a parameter, its value can be one of the following:

- A simple text field for simple parameter values.
- A combo box with some predefined values.
- A file chooser and an *editor variable* selector to simplify setting a file path as the value of a parameter.

Note: To input parameter values at runtime, use the *ask editor variable* in the **Value** column.

The following actions are available for managing parameters:

New

Opens the **Add Parameter** dialog box that allows you to add a new parameter to the list. An *editor variable* can be inserted in the text box using the  **Insert Editor Variables** button.

Edit

Opens the **Edit Parameter** dialog box that allows you to change the value of the selected parameter by selecting it from a list of allowed values.

Unset

Resets the selected parameter to its default value. Available only for edited parameters with set values.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The Filters Tab

The **Filters** tab allows you to add filters to remove certain content elements from the generated output.

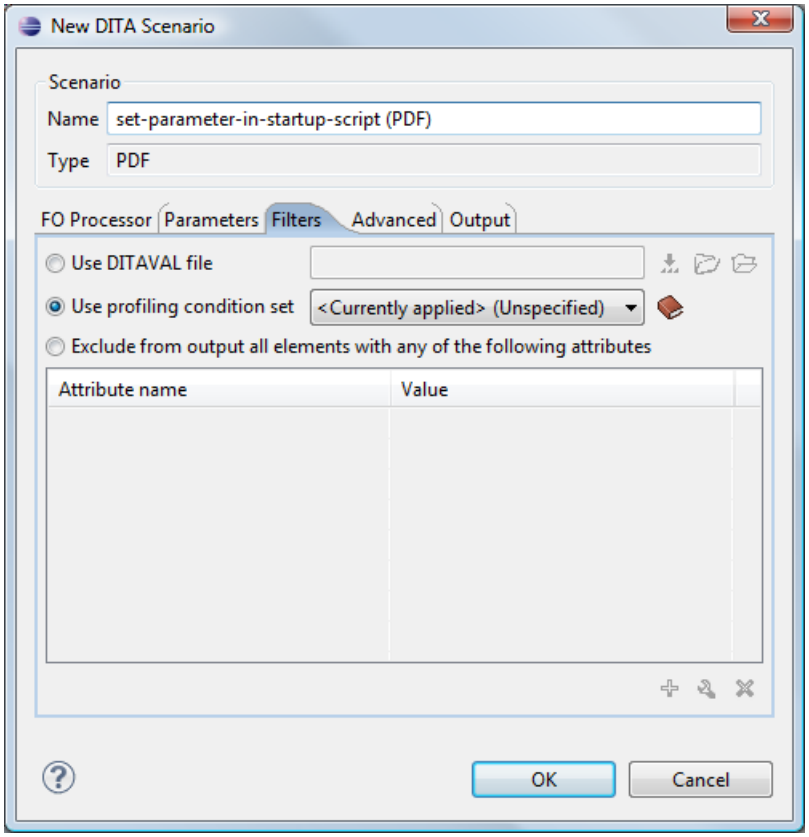


Figure 260: Edit Filters tab

There are three ways to define filters:

- **Use DITAVAL file** - If you already have a DITAVAL file associated with the DITA map, you can specify the file to be used when filtering content. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button. You can find out more about constructing a DITAVAL file in the *DITA OT Documentation*.
- **Use profiling condition set** - Sets the *profiling condition set* that will apply to your transformation.
- **Exclude from output all elements with any of the following attributes** - By using the **New**, **Edit**, or **Delete** buttons at the bottom of the pane, you can configure a list of attributes (name and value) to exclude all elements that contain any of these attributes from the output.

The Advanced Tab

The **Advanced** tab allows you to specify advanced options for the transformation scenario.

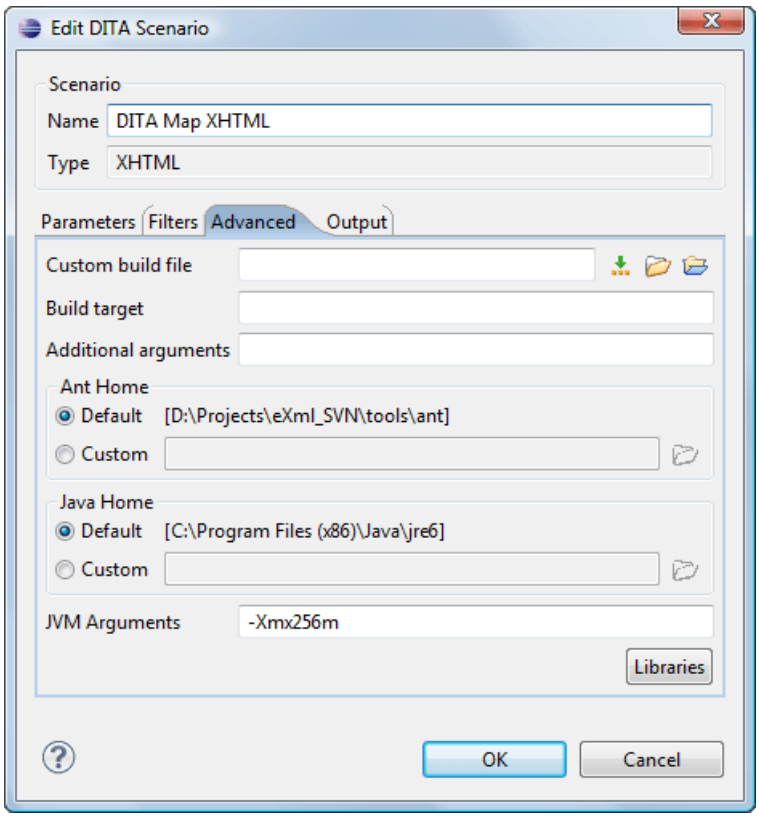


Figure 261: Advanced Settings Tab

You can specify the following parameters:

- **Custom build file** - If you use a custom DITA-OT build file, you can specify the path to the customized build file. If empty, the `build.xml` file from the *dita.dir* parameter that is configured in the **Parameters** tab is used. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button.
- **Build target** - Optionally, you can specify a build target for the build file. If no target is specified, the default `init` target is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the Ant transformation (such as -verbose).
- **Ant Home** - You can choose between the default or custom Ant installation to run the transformation.
- **Java Home** - You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by Oxygen XML Editor plugin.

  Note: It may be possible that the used Java version is incompatible with the DITA Open Toolkit engine. For example, DITA OT 1.8.5 and older requires Java 1.6 or later, while DITA OT 2.0 and newer requires Java 1.7 or newer. Thus, if you encounter related errors running the publishing, consider installing a Java VM that is supported by the DITA OT publishing engine and using it in the **Java Home** text field.

- **JVM Arguments** - This parameter allows you to set specific parameters for the Java Virtual Machine used by Ant. For example, if it is set to -Xmx384m, the transformation process is allowed to use 384 megabytes of memory. When performing a large transformation, you may want to increase the memory allocated to the Java Virtual Machine. This will help avoid Out of Memory error messages (**OutOfMemoryError**).
- **Libraries** - By default, Oxygen XML Editor plugin adds (as high priority) libraries that are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can use this button to specify additional libraries (jar files or additional class paths) to be used by the Ant transformer.

The Output Tab

The **Output** tab allows you to configure options that are related to the location where the output is generated.

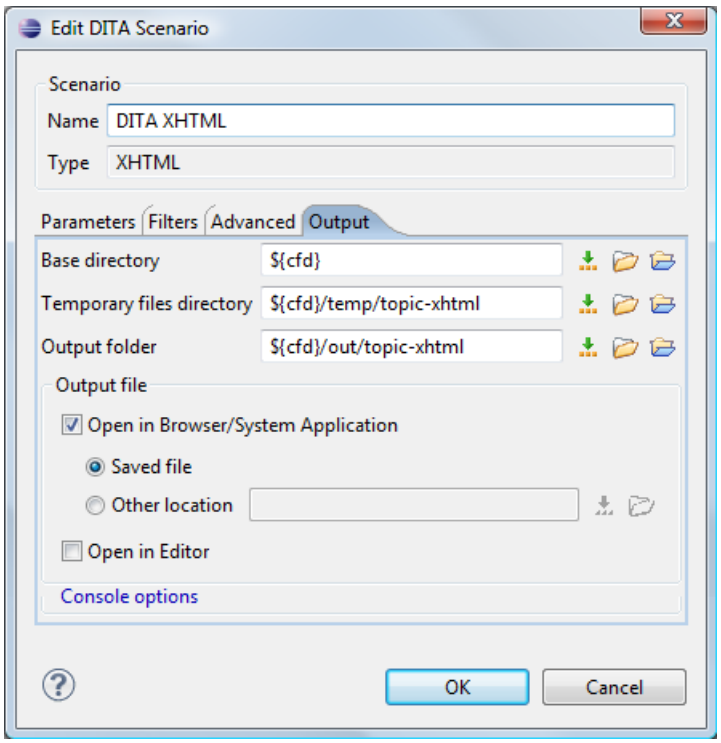


Figure 262: Output Settings Tab

You can specify the following parameters:

- **Base directory** - All the relative paths that appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.
- **Temporary files directory** - This directory is used to store pre-processed temporary files until the final output is obtained. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.
- **Output folder** - The folder where the content of the final output is stored. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.

  Note: If the DITA map or topic is opened from a remote location or a ZIP file, the parameters must specify absolute paths.

- **Open in Browser/System Application** - If enabled, Oxygen XML Editor plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, `.pdf` files are usually opened in the *Acrobat Reader* application).

  Note: To set the web browser that is used for displaying HTML/XHTML pages, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**.

  - **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin automatically opens the file specified in the **Save As** text field at the end of the transformation.
  - **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor plugin opens the file specified here. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.

The Skins Tab

A *skin* is a collection of CSS properties that can alter the look of the output by changing colors, font types, borders, margins, and paddings. This allows you to rapidly adapt the look and feel of the output for your organization.

Oxygen XML Editor plugin provides a set of predefined *skins* for the **DITA Map WebHelp** and **DITA Map WebHelp with Feedback** transformation scenarios.

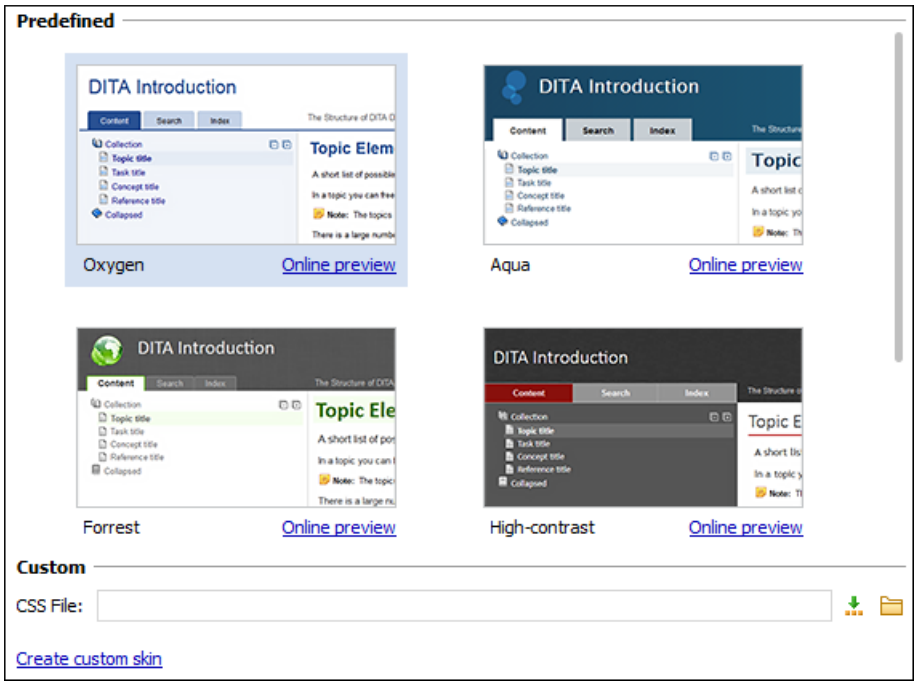


Figure 263: The Skins Tab

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 (notice the light blue border around the skin preview). If you want to obtain an output without any customization, deselect the currently selected skin.

To see how the *skin* looks when applied on a sample documentation project that is stored on the Oxygen XML Editor plugin 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 referenced in this parameter take precedence over the properties declared in the skin set in the **Skins tab**.

Customizing DITA Transformations

Oxygen XML Editor plugin includes a bundled copy of the DITA-OT as an Oxygen XML Editor plugin *framework*. That framework includes a number of transformation scenarios for common output formats. This section includes topics about customizing DITA transformations, such as using a custom build file, customizing PDF output, and using DITA OT plugins to customize your needs.

Using a Custom Build File

To use a custom build file in a DITA-OT transformation, follow these steps:

1. Use the **Configure Transformation Scenario(s)** action to open the **Configure Transformation Scenario(s)** dialog box.
2. Select the transformation scenario and click **Edit**.

3. Go to the ***Advanced*** tab and 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>
```

Note: If you use the built-in Ant 1.8.2 build tool that comes bundled with Oxygen XML Editor plugin, it is located in the `[OXYGEN_DIR]/tools/ant` directory. Any additional libraries for Ant must be copied to the Oxygen XML Editor plugin Ant `lib` directory.

DITA to PDF Output Customization

This section includes topics about customizing DITA to PDF output.

Creating a Customization Directory for PDF Output

DITA Open Toolkit PDF output can be customized in several ways:

- 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*.
- Creating a customization directory and using it from the PDF transformation scenario. A small example of this procedure can be found below.

The following procedure explains how to do a basic customization of the PDF output by setting up a customization directory. An example of a common use case is embedding a company logo image in the front matter of the book.

1. Copy the entire directory: *`DITA_OT_DIR`*`\plugins\org.dita.pdf2\Customization` to another location (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 Editor plugin 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 Editor plugin and create the template called createFrontMatter_1.0. This will override the same template from the *`DITA_OT_DIR`*`\plugins\org.dita.pdf2\xsl\fo\front-matter.xsl`. Now, `custom.xsl` has the following content:

```
<?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 the **DITA Map to PDF** transformation scenario and in the Parameters tab, set the customization.dir parameter to `C:\Customization`.

Header and Footer Customization in PDF Output

The XSLT stylesheet *`DITA_OT_DIR`*`/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 *`DITA_OT_DIR`*`/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 *Creating a Customization Directory for PDF Output* on page 469.

Adding a Watermark to PDF Output

To add a watermark to the PDF output of a DITA map transformation, follow these steps:

1. Create a custom XSL stylesheet named `custom.xsl` that sets the path of the watermark image, as in the following example:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:variable name="imageDir" select="'/common/artwork/'"/>
   <xsl:variable name="imageWatermarkPath"><xsl:value-of select="$imageDir"/>ExampleImage.png</xsl:variable>
</xsl:stylesheet>
```

2. Place the `custom.xsl` file in the following directory: *`DITA_OT_DIR`*`\plugins\org.dita.pdf2\Customization\fo\attrs`.
3. In the *`DITA_OT_DIR`*`\plugins\org.dita.pdf2\Customization` directory, locate the file named `catalog.xml.orig` and rename it as `catalog.xml`.

4. Edit the `catalog.xml` file and *uncomment* from the following line:

```
<!--uri name="cfg:fo/attrs/custom.xsl" uri="fo/attrs/custom.xsl"/-->
```

```
<uri name="cfg:fo/attrs/custom.xsl" uri="fo/attrs/custom.xsl"/>
```

5. Edit the `static-content.xsl` file that is located in the *DITA_OT_DIR*`\plugins\org.dita.pdf2\xsl\fo` directory and add the following template:

```
<xsl:template name="insertImage">
            <fo:block-container absolute-position="fixed" top="160mm">
                <fo:block>
                    <fo:external-graphic src="url({concat($customizationDir.url, $imageWatermarkPath)})"
xsl:use-attribute-sets="image" />
                </fo:block>
            </fo:block-container>
</xsl:template>
```

6. In the same file (`static-content.xsl`), add a call to this template before the `<fo:block>` tag in each of the header templates (`insertBodyOddHeader`, `insertBodyEvenHeader`, `insertBodyFirstHeader`, `insertBodyLastHeader`, `insertTocOddHeader`, `insertTocEvenHeader`, and `insertBodyFootnoteSeparator`), or in each of the footer templates (`insertBodyFirstFooter`, `insertBodyLastFooter`, `insertBodyOddFooter`, `insertBodyEvenFooter`, `insertTocOddFooter`, `insertTocEvenFooter`, and `insertBodyFootnoteSeparator`). The call would look like this:

```
<xsl:call-template name="insertImage"/>
```

Note: If you add the call in both the header and footer templates, the watermark image will be displayed twice in each page of the output.

7. Edit a *DITA Map PDF* transformation scenario and in the **Parameters** tab, set the value of the `customization.dir` parameter to something like: `${frameworksDir}/dita/DITA-OT/plugins/org.dita.pdf2/Customization`, where `${frameworksDir}/dita/DITA-OT` is pointing to your *DITA_OT_DIR*.
8. Apply the transformation scenario.

Force Page Breaks Between Two Block Elements

Suppose that at some point in your DITA content you have two block level elements, such as two paragraphs:

```
<p>First para</p>
<p>Second para</p>
```

and you want to force a page break between them in the PDF output.

Here is how you can implement a DITA Open Toolkit plugin that would achieve this:

1. Define your custom processing instruction that marks the place where a page break should be inserted in the PDF, for example:

```
<p>First para</p>
  <?pagebreak?>
<p>Second para</p>
```

2. Locate the **DITA Open Toolkit** distribution and in the `plugins` directory create a new plugin folder named `pdf-page-break`, for example.
3. In this new folder, create a new `plugin.xml` file with the content:

```
<plugin id="com.yourpackage.pagebreak">
  <feature extension="package.support.name" value="Force Page Break Plugin"/>
  <feature extension="package.support.email" value="support@youremail.com"/>
  <feature extension="package.version"value="1.0.0"/>
  <feature extension="dita.xsl.xslfo" value="pageBreak.xsl" type="file"/>
</plugin>
```

The most important feature in the plugin is that it will add a new XSLT stylesheet to the XSL processing that produces the PDF content.

4. In the same folder, create an XSLT stylesheet named `pageBreak.xsl` with the following content:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format"version="1.0">
  <xsl:template match="processing-instruction('pagebreak')">
    <fo:block break-after="page"/>
  </xsl:template>
</xsl:stylesheet>
```

5. *Install your plugin in the DITA Open Toolkit.*

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 *DITA_OT_DIR*`/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]\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.

Set a Font for PDF Output Generated with FO Processor

When a DITA map is transformed to PDF using an FO processor and it contains some Unicode characters that cannot be rendered by the default PDF fonts, a font that is capable of rendering these characters must be configured and embedded in the PDF result.

The settings that must be modified for configuring a font for the built-in FO processor are detailed in *this section*.

DITA OT PDF Font Mapping

The DITA OT contains a file *DITA_OT_DIR*`/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**.

Adding a Watermark to XHTML Output

To add a watermark to the XHTML output of a DITA map transformation, follow these steps:

1. Create a custom CSS stylesheet that includes the watermark image, as in the following example:

```
body {
 background-image: url(MyWatermarkImage.png);
}
```

2. Edit a *DITA Map XHTML* transformation scenario and in the **Parameters** tab set the value of the `args.css` parameter as the path to your watermark image.
3. Set the value of the `args.copycss` parameter to `yes`.
4. Apply the transformation scenario.
5. Copy the watermark image in the output directory of the transformation scenario, next to the CSS file created in step 1.

DITA Map to PDF WYSIWYG Transformation

Oxygen XML Editor plugin comes bundled with a DITA OT plugin that converts DITA maps to PDF using a CSS layout processor. Oxygen XML Editor plugin supports the following processors (not included in the Oxygen XML Editor plugin installation kit):

- **Prince Print with CSS** - A third-party component that needs to be purchased from *http://www.princexml.com*.
- **Antenna House Formatter** - A third-party component that needs to be purchased from *http://www.antennahouse.com/antenna1/formatter/*.

The DITA-OT plugin is located in the following directory: *`DITA_OT_DIR`*`/plugins/com.oxygenxml.pdf.css`.

Although it includes a set of CSS files in its `css` subfolder, when this plugin is used in Oxygen XML Editor plugin, the CSS files located in the `${frameworks}` directory take precedence.

Creating the Transformation Scenario

To create an experimental DITA map to PDF WYSIWYG transformation scenario, follow these steps:

1. Click the **Configure Transformation Scenario(s)** button from the **Dita Maps Manager** toolbar.
2. Select **DITA Map PDF - WYSIWYG - Experimental**.
3. When applied, this new transformation scenario uses the currently selected CSS files for the opened topic files. These CSS files can be selected from the **Styles** drop-down menu from the toolbar.

   Important: The author could open the map in the editor and change its style, but this is ignored in the publishing stage. Since authors usually edit topics instead of the map, Oxygen XML Editor plugin uses the styles selected in the opened topics.

4. In the **Parameters** tab, configure the following parameters:
   - `css.processor.path.prince` (if you are using the **Prince Print with CSS** processor) - Specifies the path to the Prince executable file that will be run to produce the PDF. If you installed Prince using its default settings, you can leave this blank.
   - `css.processor.path.antenna-house` (if you are using the **Antenna House Formatter** processor) - Specifies the path to the Antenna House executable file that will be run to produce the PDF. If you installed Antenna House using its default settings, you can leave this blank.

- `show.changes.and.comments` - When set to `yes`, the user comments and tracked changes are shown in the output. The default value is `no`.

Customizing the Styles (for Output and Editing)

If you need to change the styles, make sure you install Oxygen XML Editor plugin in a folder in which you have full read and write privileges (for instance, your user home directory). This is due to the fact that usually all the installed files are read-only (for instance, in Windows, Oxygen XML Editor plugin is installed in the `Program Files` folder where the users do not have change rights).

If you want to change the style of an element, open a document in the editor and select **Inspect Styles** from the contextual menu. *The **CSS Inspector** view* that shows all the CSS rules that apply to the selected element will be displayed. Click the link for the CSS selector that you need to change and Oxygen XML Editor plugin will open the CSS file and position the cursor at that selector. Simply add the properties you need and to see the changes in the editor, press **F5** to reload the document. Once you are satisfied with how it looks, use the transformation scenario and check for the changes in the PDF output.

Note: This experimental transformation scenario also allows you to present colored highlights in the PDF output.

DITA Profiling / Conditional Text

DITA offers support for conditionally profiling content by using profiling attributes. With Oxygen XML Editor plugin, you can define values for the DITA profiling attributes and they can be easily managed to filter content in the published output. You can switch between different profile sets to see how the edited content looks like before publishing. The profiling configuration can also be shared between content authors through the project file and there is no need for coding or editing configuration files.

Profiling Attributes

You can profile content elements or map elements by adding one or more of the default DITA profiling attributes (`product`, `platform`, `audience`, `rev`, `props`, and `otherprops`). You can also create your own *custom profiling attributes* and *custom profiling conditions sets*. The profiling attributes may contain one or more tokens that represent conditions to be applied to the content when a publication is built.

For example, you could define a section of a topic that would only be included for a publication related to the Windows platform by adding the `platform` profiling attribute:

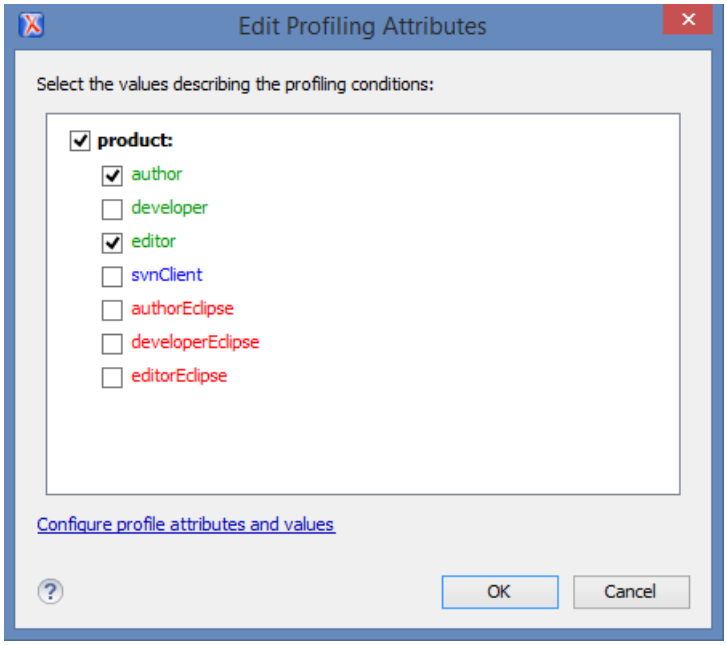
```
<section platform="windows">
```

Profiling Conditions

DITA allows you to conditionally profile parts of a topic so that certain parts of the topic are displayed when certain profiling conditions are set. Profiling conditions can be set both within topics and in maps. When set in a topic, they allow you to suppress an element (such as paragraph), step in a procedure, item in a list, or even a phrase within a sentence. When set in a map, they allow you to suppress an entire topic or group of topics. You can then create a variety of different publications from a single map by applying profiling conditions to the build.

Apply Profiling to DITA Content

To apply a profiling attribute to DITA content, highlight the content and select **Edit Profiling Attributes** from the contextual menu. To profile specific elements in a topic or map, right-click inside the element and select **Edit Profiling Attributes**. The **Edit Profiling Attributes** dialog box is displayed, allowing you to check each of the profiling tokens that apply for each attribute.

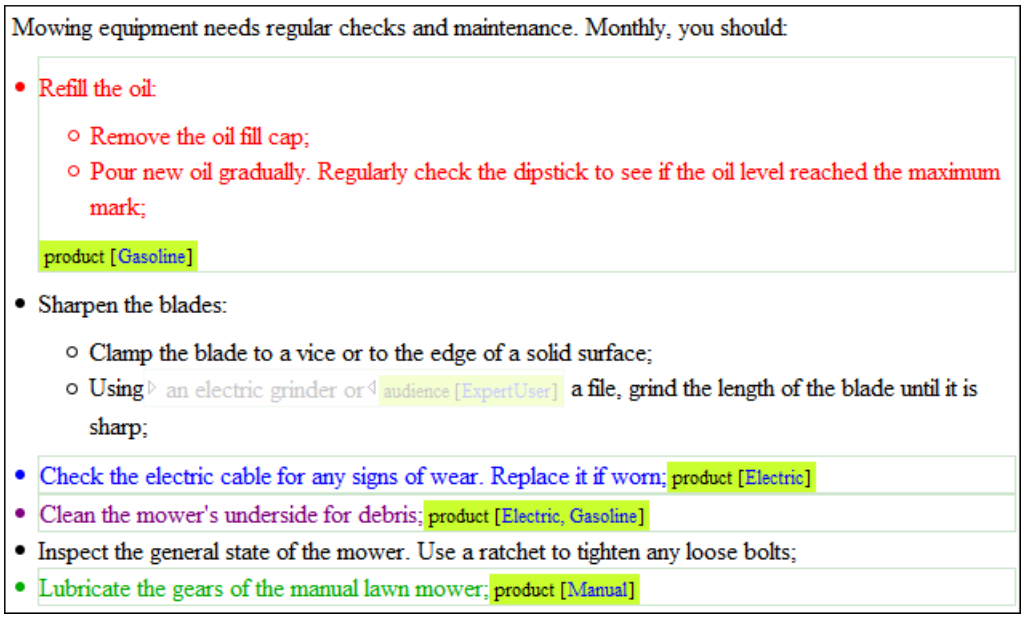


The profiling attributes, and their potential values, that appear in this dialog box depend on what has been configured in Oxygen XML Editor plugin. The content of the dialog box is determined as follows:

- If your root DITA map references a DITA subject scheme map that defines values for the profiling attributes, those values are used. In the image above, which is taken from the Oxygen XML Editor plugin documentation project, values are defined for seven different products, but none for other profiling attributes, which are therefore omitted from the dialog box.
- Otherwise, a basic default set of profiling attributes and values are available.

Visualizing Profiled Content

You can visualize the effect of profiling content by using the profiling tools in the **Profiling/Conditional Text** drop-down menu that is located on the **DITA Maps Manager** toolbar. You can select which profiles to show, or apply colors to text that is profiled in various ways, as shown in the following image:



To watch our video demonstration about DITA profiling, go to *http://oxygenxml.com/demo/DITA_Profiling.html*.

Profiling DITA Content

1. *Open the **Preferences** dialog box*, go to **Editor** > **Edit modes** > **Author** > **Profiling / Conditional Text**, and edit the **Profiling Attributes** table.

   Note that this table will be ignored if a *Subject Scheme Map* is in use.

2. For DITA documents, 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. You can use the profiling attributes set in the previous step with one of the following methods:

   a) To profile DITA topics, right-click a topic reference in the **DITA Maps Manager**, select **Edit Properties** from the contextual menu, and go to the **Profiling** tab.

   b) To profile DITA content in **Author** mode, highlight the content and select **Edit Profiling Attributes** from the contextual menu.

   c) To profile specific XML elements in **Author** mode, right-click inside the element and select **Edit Profiling Attributes** or use the **Attributes** view to set the profiling attributes on the element at the current cursor position.

   Enable the **Show Profiling Attributes** option to display the profiling markup in the **Author** editing mode.

Profiling / Conditional Text Markers

If the **Show Profiling Attributes** option (available in the ***Profiling / Conditional Text** toolbar menu*) is enabled, 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.

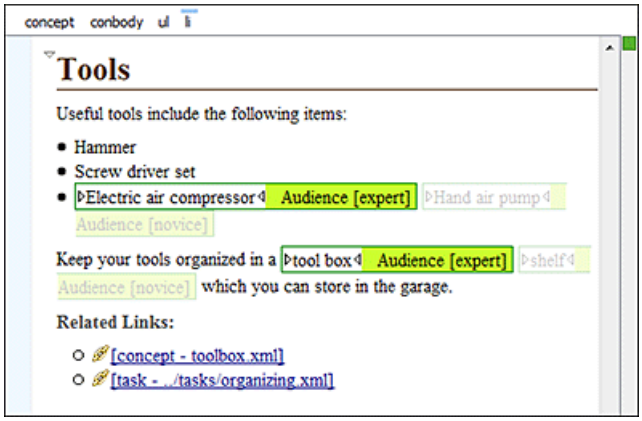


Figure 264: Profiling in Author

In the ***DITA Maps Manager** view*, the following icons are used 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.

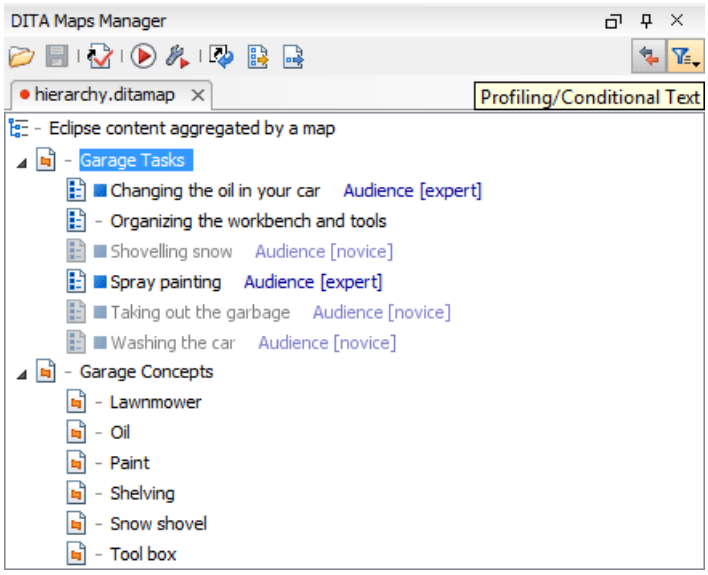


Figure 265: Profiling in DITA Maps Manager

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. A map that uses the set of profiling values must reference the subject scheme map in which the profiling values are defined at its highest level, as in the following example:

```
<topicref href="test.ditamap" format="ditamap" type="subjectScheme"/>
```

A profiled value should be 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.

The following is an example of content from a 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>
  <!-- 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,
       surgeon, neuro-surgeon and plastic-surgeon. -->
  <enumerationdef>
    <attributedef name="audience"/>
    <subjectdef keyref="users"/>
  </enumerationdef>
</subjectScheme>
```

When you edit a DITA topic in the **Text** or **Author** mode, Oxygen XML Editor plugin collects all the profiling values from the Subject Scheme Map that is referenced in the map that is 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 box*,

regardless of their mapping in the ***Profiling/Conditional Text*** *preferences page*. They are also available as proposals for values in the ***Content Completion Assistant***.

> In the example above, the values `therapist`, `oncologist`, `physicist`, and so on, are displayed in the content completion window as values for the `audience` attribute.

Consider that you have the following fragment in a topic:

```
<p audience="neuro-surgeon">Some text.. </p>
```

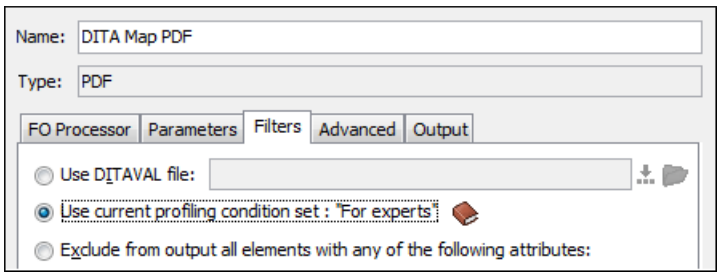
When you define a DITAVAL filter, you can, for instance, exclude anything that is profiled as `surgeon`:

```
<val>
     <prop action="exclude" att="audience" val="surgeon"/>
</val>
```

If you then transform the main DITA map specifying the DITAVAL filter file in the transformation scenario, the `p` element should be excluded from the output. Therefore, excluding the `surgeon` audience also excludes `neuro-surgeon` and `plastic-surgeon` from the output. More details about how hierarchical filtering and Subject Scheme Maps should work are found in the following 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*

Publishing Profiled Text

Oxygen XML Editor plugin comes with preconfigured transformation scenarios for DITA. By default, these scenarios take the current profiling condition set into account during the transformation, as defined in the **Filters** tab when *creating a DITA transformation*.



DITA Open Toolkit Support

Oxygen XML Editor plugin includes support for the DITA Open Toolkit.

Creating a DITA OT Customization Plugin

To describe the steps involved in creating a **DITA Open Toolkit** plugin, this section uses an example of creating an **XSLT** customization plugin that provides syntax highlighting when publishing DITA `codeblock` elements to **HTML** and **PDF** output formats. This plugin (**com.oxygenxml.highlight**) is available in the **DITA Open Toolkit** distribution that comes bundled with the latest version of Oxygen XML Editor plugin, but these instructions show you how to create it as if it were not included.

The steps to help you to create the plugin are as follows:

1. Create a folder for your plugin in the DITA OT **plugins** folder (*`DITA_OT_DIR`*`/plugins/`).

   For example, if you are using DITA 1.8 the path would look like this:

   `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.oxygenxml.highlight`

2. Create a **plugin.xml** file (in the same plugin folder) that contains the extension points of the plugin.

Note: You can easily create this file by using the **DITA OT Plugin** new file template that is included in Oxygen XML Editor plugin. From the **New** file wizard you can find this template in **Framework templates** > **DITA** > **plugin**.

For example, our syntax highlighting plugin example contains the following:

```
<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 most important extensions in it are the references to the XSLT stylesheets that will be used to style the HTML and PDF outputs.

You can find other **DITA OT** plugin extension points here:
http://dita-ot.sourceforge.net/1.5.3/dev_ref/extension-points.html

3. Create an XSLT stylesheet to customize the output types. In our example, to customize the HTML output we need to create an XSLT stylesheet called **xhtmlHighlight.xsl** (in the same plugin folder).

   Tip: You can use the **Find/Replace in Files** to find an XSLT stylesheet with content that is similar to the desired output and use it as a template to overwrite parts of your stylesheet. In our example we want to overwrite the creation of the HTML content from a DITA **codeblock** element. Since a DITA `codeblock` element has the `class` attribute value `+ topic/pre pr-d/codeblock` we can take part of the class attribute value (**topic**/**pre**) and search the **DITA OT** resources for a similar stylesheet.

   Our search found the XSLT stylesheet *DITA_OT_DIR*`/org.dita.xhtml/xsl/xslhtml/dita2htmlImpl.xsl`, which contains:

   ```
   <xsl:template match="*[contains(@class,' topic/pre ')]" name="topic.pre">
     <xsl:apply-templates select="."  mode="pre-fmt" />
   </xsl:template>
   ```

   We use it to overwrite our **xhtmlHighlight.xsl** stylesheet, which results in the following:

   ```
   <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>
   ```

   You could also use another XSLT template that applies the XSLTHL library as a Java extension to style the content.

4. Create additional XSLT stylesheets to customize all other desired output types. In our example, to customize the PDF output we need to create an XSLT stylesheet called **pdfHighlight.xsl** (in the same plugin folder).

   In this case we found an appropriate XSLT stylesheet *DITA_OT_DIR*`/plugins/legacypdf/xslfo/dita2fo-elems.xsl` to use as a template that we use to overwrite our **pdfHighlight.xsl** stylesheet, which results in the following:

   ```
   <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>
```

Note: You can edit the newly created stylesheets to customize different outputs in a variety of ways. For example, in our case you could edit the **xhtmlHighlight.xsl** or **pdfHighlight.xsl** stylesheets that we created to customize various colors for syntax highlighting.

5. To install the created plugin in the **DITA OT**, run the predefined transformation scenario called **Run DITA OT Integrator** by executing it from the **Apply Transformation Scenario(s)** dialog box. If the integrator is not visible, enable the **Show all scenarios** action that is available in the settings drop-down menu. For more information, see *Installing a Plugin in the DITA Open Toolkit* on page 480.

   Results of running the integrator using our example:

   XSLT content is applied with priority when publishing to both HTML and PDF outputs.

   a. For the HTML output, in the XSLT stylesheet *DITA_OT_DIR*`/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 thus has a higher priority. See more about imported template precedence in the XSLT specs: *http://www.w3.org/TR/xslt#import*

   b. Likewise, for the PDF output, in the top-level stylesheet *DITA_OT_DIR*`/plugins/org.dita.pdf2/xsl/fo/topic2fo_shell_fop.xsl` a new import statement appeared:

      ```
      <xsl:import href="../../../com.oxygenxml.highlight/pdfHighlight.xsl"/>
      ```

Now, you can distribute your plugin folder to anyone that has a DITA OT installation along with some simple installation notes. Your customization will work provided that the templates you are overwriting have not changed from one DITA OT distribution to the other.

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 *DITA_OT_DIR*`\plugins` directory.
2. Run the predefined transformation scenario called **Run DITA OT Integrator** by executing it from the **Apply Transformation Scenario(s)** dialog box. If the integrator is not visible, enable the **Show all scenarios** action that is available in the settings drop-down menu.

   Important: The folder where the **DITA OT** is located needs to have full write access permissions set to it. For example if you are integrating plugins in the **DITA OT** folder bundled with Oxygen XML Editor plugin and if you are running on **Windows** and your application is installed in the **Program Files** folder, you can start the Oxygen XML Editor plugin main executable with administrative rights in order for the integrator process to be able to modify resources in the **DITA OT** folder.

   Starting with version 17.0, Oxygen XML Editor plugin detects the transformation type (`transtype`) declarations from **DITA OT** plugins and presents descriptions, which are contributed in the `transtype` declarations, in the **DITA Transformation Type** dialog box. Oxygen XML Editor plugin also shows the contributed parameters from **DITA OT** plugins in the **Parameters** tab in the **Edit DITA Scenario** dialog box.

3. If the plugin contributed a new transformation type that is not detected (for instance, if you are using a previous version of Oxygen XML Editor plugin that does not detect the `transtype` declarations), you can create a new **DITA OT** transformation scenario with a predefined type that is similar to the new transformation type. Then edit the transformation scenario, and in the **Parameters** tab add a `transtype` parameter with the value of the new transformation type.

Note: A transformation type can also extend another `transtype`. For example, the *pdf-prince* `transtype` extends a `commons` transformation type that contains all the common DITA OT parameters.

Example:

```
<plugin id="com.oxygenxml.pdf.prince">
  <!-- extensions -->
  <feature extension="dita.conductor.transtype.check" value="pdf-prince" type="txt"/>
  <feature extension="dita.conductor.target.relative" value="integrator.xml" type="file"/>
  <feature extension="dita.transtype.print" value="pdf-prince"/>
  <transtype name="pdf-prince"extends="commons" desc="PDF (Prince XML - Experimental)">
    <param name="princeExecPath" type="file" desc="Path to the Prince executable file (eg:
&amp;quot;c:\path\to\prince.exe&amp;quot; on Windows) which should be run to produce the PDF"/>
  </ Transtype>
</plugin>
```

Use an External DITA Open Toolkit in Oxygen XML Editor plugin

Oxygen XML Editor plugin comes bundled with a DITA Open Toolkit, located in the *DITA_OT_DIR* directory. Starting with Oxygen XML Editor plugin version 17, if you want to use an external DITA OT for all transformations and validations, you can *open the **Preferences** dialog box* and go to *the **DITA** page*, where you can specify the DITA OT to be used. Otherwise, to use an external 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 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 DTDs and you want to use the new versions for content completion and validation, go to the Oxygen XML Editor plugin preferences in the **Document Type Association** page, edit the **DITA** and **DITA Map** document types and modify the catalog entry in the **Catalogs** tab to point to the custom catalog file `catalog-dita.xml`.

DITA Specialization Support

This section explains how you can integrate and edit a DITA specialization in Oxygen XML Editor plugin.

Integration of a DITA Specialization

A DITA specialization usually includes:

- DTD definitions for new elements as extensions of existing DITA elements.
- Optional 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 the 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 *DITA_OT_DIR*`\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 directory 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 DITA OT plugin, you have the following options:
   - If the DTD 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*** *preferences page*.

- In your specialization plugin directory, create a folder called `template_folders`, which would contain all the folders with new file templates. In Oxygen XML Editor plugin, the templates contributed by the plugin will be available in the **New** document wizard.

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 (such as insertion of topic refs, heads, properties editing) allow you 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 manually to insert the correct element name at cursor position. You can go to the **DITA Map** document type from the ***Document Type Association*** *preferences page* and edit the table actions to insert the element names as specified in your specialization. See the *Adding or Editing a Document Type Association (Framework)* on page 558 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** mode.

The content completion should work without additional modifications and you can choose the tags that are allowed at the cursor 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. You can go to the `DITA` document type from the ***Document Type Association*** *preferences page* and edit the actions to insert the element names, as specified in your specialization. See the *Adding or Editing a Document Type Association (Framework)* on page 558 section for more details.

Metadata

Metadata is a broad concept that describes data that explains or identifies other data. Metadata can be used for many purposes, from driving automation of document builds to enabling authors and readers to find content more easily. DITA provides a number of different types of metadata, each of which has different uses and is created in different places and ways. Some of the most important forms of metadata in DITA are topic and taxonomy.

Topic Metadata

Topic metadata describes the topic and what it is about. Topic metadata can be inserted in the `prolog` element of a topic or inside the `topicref` element that points to a topic from a map. In other words, metadata about the topic can be asserted by the topic itself, or can be assigned to it by the map that includes it in the build. This allows various different maps to assign metadata to the same topic. This may be appropriate where you want to describe a topic differently in various documents.

Taxonomy and Subject Scheme

A taxonomy is a controlled vocabulary. It can be used to standardize how many things in your content and metadata are named. This consistency in naming can help ensure that automated processes work correctly, and that consistent terminology is used in content, and in metadata. In DITA, taxonomies are created using subject scheme maps. When you are authoring, many of the values you choose from have been defined in subject scheme maps.

Creating an Index in DITA

In DITA, indexes are created from `indexterm` elements. You can insert index term elements:

- In the header of a topic. In paginated media, such as a printed book or a PDF, this results in an index entry that points to the page in which the topic starts, even if it is not the page in which the indexed term occurs.
- In the `topicref` element in a map that references the topic. This applies those index terms to that topic only when used in that map, allowing you to index topics differently in various publications. In paginated media, index entries point to the page in which the topic starts.
- In the body of a topic. In paginated media, this results in an index entry that points to the page in which the `indexterm` element occurs, even if that is not the page in which the topic starts.

To add index terms to the text of a topic of the topic header, *create the elements, as you normally would, in Oxygen XML Editor plugin*. To add index terms to a map, open the map in the editor and add the elements, as you normally would, in a topic.

In some media, indexes will be generated automatically when index entries are found in the source. For other media, such as books, you may need to tell DITA where to place the index. For instance, to add an index to a bookmap, you need to add an `indexlist` element to the `backmatter` of the book.

1. Open your *bookmap* in the **DITA Maps Manager**.
2. Right-click the book map and select **Append child** > **Backmatter**. The ***Insert Reference** dialog box* appears.
3. Click **Insert and Close** to insert the `backmatter` element.
4. Right-click the `backmatter` element and create a `booklists` element using **Append child** > **Book Lists**.
5. Use the same steps to create an `indexlist` element.

Caution: Adding `index` entries and an `indexlist` to your project creates an instruction to the DITA publishing routines to create an index. There is no guarantee that all DITA output types or third-party customizations obey that instruction or create the index the way you want it. Modifying the output may be necessary to get the result you want.

DITA 1.3 Experimental Support

Starting with version 17.1, Oxygen XML Editor plugin includes support for some DITA 1.3 features.

To enable DITA 1.3 support in Oxygen XML Editor plugin and use the DITA Open Toolkit 2.x for publishing, *open the **Preferences** dialog box*, go to **DITA**, and select the **Built-in DITA OT 2.x** radio button.

The Oxygen XML Editor plugin publication of the full DITA 1.3 specifications can be found at *http://www.oxygenxml.com/dita/1.3/specs/index.html#introduction/dita-release-overview.html*.

The following table is a list of DITA 1.3 features and their implementation status in Oxygen XML Editor plugin:

Table 7: DITA 1.3 Features Implementation Status

Feature	**Editing**	**Publishing [Latest DITA Open Toolkit 2.x is used.]**
DITA 1.3 DTD, XML Schema, and Relax NG-based maps/topics/tasks/references, etc.	New DITA 1.3 file templates. By default DITA topics and maps which do not specify version in the DOCTYPE declaration are also considered to be DITA 1.3 Specific annotations presented in the content completion window and documentation tooltips for all new DITA 1.3 elements	N/A
Learning Object and Group maps	New file templates	No specific support implemented
Troubleshooting specialization	Create and edit new troubleshooting topics	No specific support implemented
XML markup domain	Validation and Content Completion	Special rendering in PDF and XHTML-based outputs
Equation and MathML domain	Validation and content completion Display and Insert equations	Special rendering in PDF and XHTML-based outputs
SVG domain	Validation and content completion Display referenced SVG content	Special rendering in PDF and XHTML-based outputs
Other new DITA 1.3 elements (`div`, `strike-through`, `overline`, etc)	Validation and Content Completion	Special rendering in PDF and XHTML-based outputs
Release management domain	Validation and Content Completion	No specific support implemented
Scoped keys	Define key scopes Validate and check for completeness Resolve *keyrefs* and *conkeyrefs* taking key scopes into account Key scope information is displayed in a tooltip when hovering over an item in the **DITA Maps Manager**	Partially implemented (Various issues may still be encountered)
Branch filtering	Display, create, and edit *ditavalref* elements	Partially implemented (Various issues may still be encountered)
Key-based cross deliverable addressing	Special display for references to DITA maps with **scope="peer"** and a defined **keyscope** Gather and present keys from peer maps	Not implemented.

Feature	**Editing**	**Publishing [Latest DITA Open Toolkit 2.x is used.]**
Shorthand to address syntax that identifies elements in the same topic	Properly resolved for validation, links, and conrefs	Implemented
Various table attributes (`orientation`, `rotation`, `scope`, and `headers` on cells)	Not implemented in the Table Properties action support. But attributes can be changed from the Attributes view	Not implemented
New Map `topicref` attributes (`cascade`, `deliveryTarget`)	Allow setting new attributes, propose proper values for them	Implemented

Chapter 8

Document Types (Frameworks)

Topics:

A *document type* or *framework* is associated to an XML file according to a set of rules. It also includes a variety of settings that improve editing capabilities in the **Author** mode for its particular file type. These settings include:

- A default grammar used for validation and content completion in both **Author** mode and **Text** mode.
- CSS stylesheets for rendering XML documents in **Author** mode.
- User actions invoked from toolbars or menus in **Author** mode.
- Predefined scenarios used for transformations for the class of XML documents defined by the document type.
- XML catalogs.
- Directories with file templates.
- User-defined extensions for customizing the interaction with the content author in **Author** mode.

Oxygen XML Editor plugin comes with built-in support for many common document types. Each document type is defined in a framework. You can create new frameworks or make changes to existing frameworks to suit your individual requirements.

To see a video on configuring a framework in Oxygen XML Editor plugin, go to *http://oxygenxml.com/demo/FrameworkConfiguration.html*.

Predefined Document Types (Frameworks)

Predefined Document Types

The following predefined document types (frameworks) are fully supported in Oxygen XML Editor plugin and each of these document types include built-in transformation scenarios, validation, content completion, file templates , default CSS files for rendering content in **Author** mode, and default actions for editing in **Author** mode:

- *DocBook 4* - A document type standard for books, articles, and other prose documents (particularly technical documentation).
- *DocBook 5* - An enhanced (version 5) document type standard designed for a variety of documents (particularly technical documentation).
- *DITA* - An XML-based architecture designed for authoring, producing, and delivering technical information.
- *DITA Map* - A document type that collects and organizes references to DITA topics or other maps.
- *XHTML* - Extensible HyperText Markup Language includes the same depth of expression as HTML, but also conforms to XML syntax.
- *TEI ODD* - Text Encoding Initiative One Document Does it all is an XML-conformant specification that allows you to create TEI P5 schema in a literate programming style.
- *TEI P4* - The Text Encoding Initiative guidelines is a standard for the academic community that collectively define an XML format for text that is primarily semantic rather than presentational.
- *TEI P5* - The Text Encoding Initiative guidelines is a standard for the academic community that collectively define an XML format for text that is primarily semantic rather than presentational.
- *JATS* - The NISO Journal Article Tag Suite is a technical standard that defines an XML format for scientific literature.

Other Document Types

Oxygen XML Editor plugin also provides limited support (including file templates) for a variety of other document types, including:

- *EPUB (NCX, OCF, OPF 2.0 & 3.0)* - A standard for e-book files.
- *DocBook Targetset* - For resolving cross-references when using *olinks*.
- *XSLT Stylesheets* - A document type that provides a visual mode for editing XSLT stylesheets.
- *WSDL* - Web Services Description Language is an XML language for describing the functionality offered by a web service.
- *Schematron* - For making assertions about the presence or absence of patterns in XML documents. This document type applies to the ISO Schematron version.
- *Schematron Quick Fixes (SQF)* - An extension of the ISO standard Schematron, allows developers to define *QuickFixes* for Schematron errors.
- *StratML (Part 1 & 2)* - Part 1 and 2 of the Strategy Markup Language specification.
- *XProc* - A document type for processing XProc script files.
- *XML Schema* - Documents that provide support for editing annotations.
- *MathML* - Mathematical Markup Language (2.0 and 3.0) is an application of XML for describing mathematical notations.
- *XLIFF (1.2 & 2.0)* - XML Localization Interchange File Format is a standard for passing data between tools during a localization process.
- *XQuery* - The common query language for XML.
- *CSS* - Cascading Style Sheets is a language used for describing the look and formatting of a document.
- *LESS* - A dynamic style sheet language that can be compiled into CSS.
- *Relax NG Schema* - A schema language that specifies a pattern for the structure and content of an XML document.
- *NVDL Schema* - Namespace Validation Dispatching Language allows you to specify sections of XML documents to be validated against various schemas.
- *JSON* - JavaScript Object Notation is a lightweight data-interchange format.
- *JavaScript* - Programming language of HTML and the Web.
- XML Spec - A markup language for W3C specifications and other technical reports.

- DITAVAL - DITA conditional processing profile to identify the values you want to conditionally process for a particular output, build, or other purpose.
- Daisy - A technical standard for digital audio books, periodicals, and computerized text. It is designed to be an audio substitute for print material.
- EAD - Encoded Archival Description is an XML standard for encoding archival finding aids.
- KML - Keyhole Markup Language is an XML notation for expressing geographic visualization in maps and browsers.
- Maven Project & Settings - Project or settings file for Maven build automation tool that is primarily used for Java projects.
- Oasis XML Catalog - A document that describes a mapping between external entity references and locally-cached equivalents.

The DocBook 4 Document Type

DocBook is a very popular set of tags for describing books, articles, and other prose documents, particularly technical documentation.

A file is considered to be a *DocBook 4* document when one of the following conditions are true:

- The root element name is `book` or `article`.
- The PUBLIC ID of the document contains the string `-//OASIS//DTD DocBook XML`.

The default schema that is used if one is not detected in the DocBook 4 file is `docbookxi.dtd` and it is stored in `[OXYGEN_DIR]/frameworks/docbook/4.5/dtd/`.

The default CSS files used for rendering DocBook content in **Author** mode are stored in `[OXYGEN_DIR]/frameworks/docbook/css/`.

The default XML catalog, `catalog.xml`, is stored in `[OXYGEN_DIR]/frameworks/docbook/`.

To watch our video demonstration about editing DocBook documents, go to *http://oxygenxml.com/demo/DocBook_Editing_in_Author.html*.

DocBook 4 Author Mode Actions

A variety of actions are available in the DocBook 4 framework that can be added to the **DocBook4** menu, the **Author Custom Actions** toolbar, the contextual menu, and the **Content Completion Assistant**.

DocBook 4 Toolbar Actions

The following default actions are readily available on the **DocBook (Author Custom Actions)** toolbar when editing in **Author** mode (by default, most of them are also available in the **DocBook4** menu and in various submenus of the contextual menu):

Bold

Emphasizes the selected text by surrounding it with `<emphasis role="bold">` tag. You can use this action on multiple non-contiguous selections.

Italic

Emphasizes the selected text by surrounding it with `<emphasis role="italic">` tag. You can use this action on multiple non-contiguous selections.

Underline

Emphasizes the selected text by surrounding it with `<emphasis role="underline">` tag. You can use this action on multiple non-contiguous selections.

Link Actions Drop-Down Menu

The following link actions are available from this menu:

Cross reference (link)

Opens a dialog box that allows you to select a URL to insert as a hypertext link.

Cross reference (xref)

Inserts a cross reference to other parts of the document.

Web Link (ulink)

Inserts a link that addresses its target with a URL (Universal Resource Locator).

Insert OLink

Opens an **OLink** dialog box that allows you to insert a link that addresses its target indirectly, using the `targetdoc` and `targetptr` values that are present in a ***Targetset*** file.

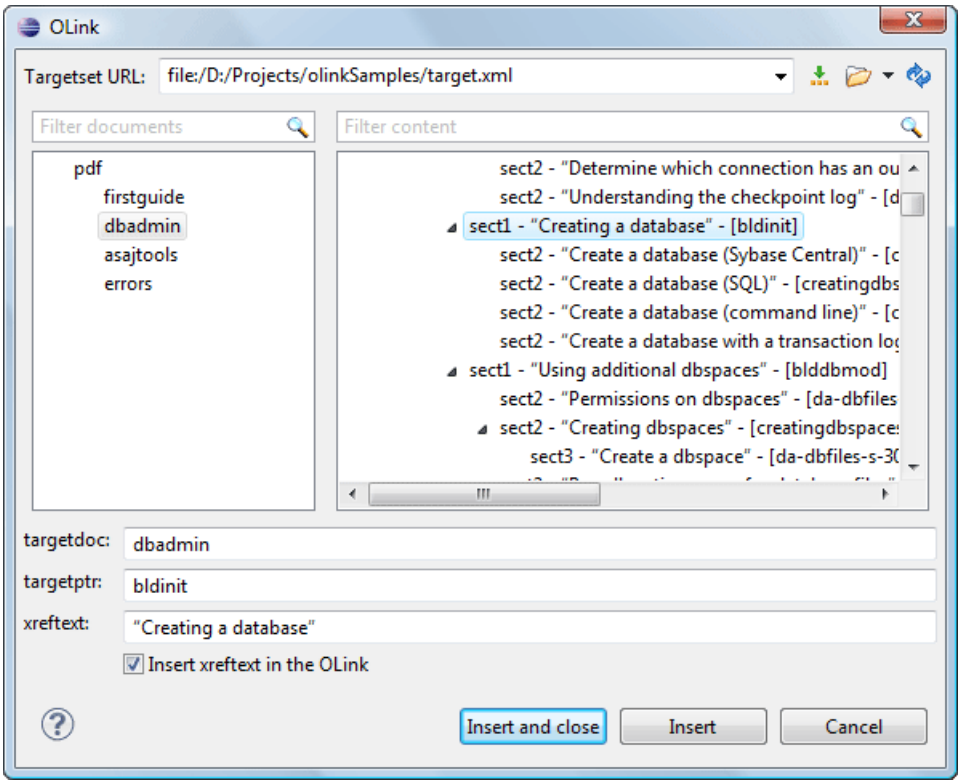


Figure 266: Insert OLink Dialog Box

After you choose the **Targetset URL**, the structure of the target documents is presented. For each target document (`targetdoc`), its content is displayed allowing you to easily identify the `targetptr` for the `olink` element that will be inserted. You can also use the search fields to quickly identify a target. If you already know the values for **targetdoc** and **targetptr**, you can insert them directly in the corresponding fields. You can also edit an `olink` using the **Edit OLink** action that is available on the contextual menu. The last used **Targetset** URL will be used to identify the edited target.

To insert XREF text into the `olink`, enter the text in the **xreftext** field and make sure the **Insert xreftext in the OLink** option is enabled.

Insert URI

Inserts an URI element. The URI identifies a Uniform Resource Identifier (URI) in content.

Insert Image

Inserts an image reference at the cursor position. Depending on the current location, an image-type element is inserted.

Insert XInclude

Opens a dialog box that allows you to browse and select content to be included and automatically generates the corresponding XInclude instruction.

Insert Section Drop-Down Menu

The following actions are available from this menu:

Insert Section

Inserts a new section or subsection in the document, depending on the current context. For example, if the current context is `sect1`, then a `sect2` is inserted. By default, this action also inserts a `para` element as a child node. The `para` element can be deleted if it is not needed.

Promote Section

Inserts the current node as a sibling of the parent node.

Demote Section

Inserts the current node a child of the previous node.

Insert Paragraph

Insert a new paragraph element at current cursor position.

Insert Equation

Opens the **XML Fragment Editor** that allows you to insert and *edit MathML notations*.

Insert List Item

Inserts a list item in the current list type.

Insert Ordered List

Inserts an ordered list at the cursor position. A child list item is also automatically inserted by default.

Insert Itemized List

Inserts an itemized list at the cursor position. A child list item is also automatically inserted by default.

Insert Variable List

Inserts a DocBook variable list. A child list item is also inserted automatically by default.

Insert Procedure List

Inserts a DocBook `procedure` element. A step `child` item is also inserted automatically.

Sort

Sorts cells or list items in a table.

Insert Table

Opens a dialog box 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 Below

Inserts a new table row with empty cells below the current row. This action is available when the cursor is positioned inside a table.

Insert Column After

Inserts a new table column with empty cells after the current column. This action is available when the cursor is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the cursor is positioned between two cells, Oxygen XML Editor plugin a new cell at cursor position. If the cursor is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at cursor position.

Delete Row

Deletes the table row located at cursor position.

Table Properties

Opens the **Table properties** dialog box that allows you to configure properties of a table (such as frame borders).

Table Join/Split Drop-Down Menu

The following join and split actions are available from this menu:

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 cursor is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current cursor 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 cursor 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 Editor plugin 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 cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

DocBook4 Menu Actions

In addition, the following default actions are available in the **DocBook4** menu when editing in **Author** mode:

Paste special submenu

This submenu includes the following special paste actions that are specific to the DocBook 4 framework:

Paste As XInclude

Allows you to create an `xi:include` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Paste as link

Allows you to create a `link` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Paste as xref

Allows you to create an `xref` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu:

Insert Row Above

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

Insert Column Before

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

ID Options

Opens the **ID Options** dialog box that allows you to configure options for generating IDs in **Author** mode. The dialog box includes the following:

ID Pattern

The pattern for the ID values that will be generated. This text field can be customized using constant strings and most of the supported *Editor Variables* on page 594.

Element name or class value to generate ID for

The elements for which ID values will be generated, specified using class attribute values. To customize the list, use the **Add**, **Edit**, or **Remove** buttons.

Auto generate IDs for elements

If enabled, Oxygen XML Editor plugin will automatically generate unique IDs for the elements listed in this dialog box when they are created in **Author** mode.

Remove IDs when copying content in the same document

Allows you to control whether or not pasted elements that are listed in this dialog box should retain their existing IDs when copying content in the same document. To retain the element IDs, disable this option.

Generate IDs

Oxygen XML Editor plugin generates unique IDs for the current element (or elements), depending on how the action is invoked:

- When invoked on a single selection, an ID is generated for the selected element at the cursor position.
- When invoked on a block of selected content, IDs are generated for all top-level elements and elements from the list in the **ID Options** dialog box that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

Refresh References

You can use this action to manually trigger a refresh and update of all referenced resources.

Tags display mode Submenu

Full Tags with Attributes - Displays full tag names with attributes for both block level and in-line level elements.

Full Tags - Displays full tag names without attributes for both block level and in-line level elements.

Block Tags - Displays full tag names for block level elements and simple tags without names for in-line level elements.

Inline Tags - Displays full tag names 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 and is as close as possible to a word-processor view.

Profiling/Conditional Text Submenu

Edit Profiling Attributes - Allows you to configure the *profiling attributes* and their values.

Show Profiling Colors and Styles - Select this option to turn on conditional styling.

Show Profiling Attributes - Select 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 - When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

List of all profiling condition sets that match the current document type - You can click a listed condition set to activate it.

Profiling Settings - Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

DocBook 4 Contextual Menu Actions

In addition to many of the *DocBook 4 toolbar actions* and the *general **Author** mode contextual menu actions*, the following DocBook 4 framework-specific actions are also available in the contextual menu when editing in **Author** mode:

Paste special submenu

This submenu includes the following special paste actions that are specific to the DocBook 4 framework:

Paste As XInclude

Allows you to create an `xi:include` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Paste as link

Allows you to create a `link` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Paste as xref

Allows you to create an `xref` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu when the contextual menu is invoked on a table:

Insert Row Above

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

Insert Column Before

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

Link > Edit OLink

Opens a dialog box that allows you edit an existing OLink. See the **Insert OLink** action for more information.

Generate IDs

Oxygen XML Editor plugin generates unique IDs for the current element (or elements), depending on how the action is invoked:

- When invoked on a single selection, an ID is generated for the selected element at the cursor position.
- When invoked on a block of selected content, IDs are generated for all top-level elements and elements from the list in the **ID Options** dialog box that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

DocBook 4 Drag/Drop Actions

Dragging a file from *the **Project** view* or *__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 (for example, the `inlinegraphic` DocBook element with a `fileref` attribute) at the drop location, similar to the **Insert Image** toolbar action.

DocBook 4 Transformation Scenarios

Default transformation scenarios allow you to convert DocBook 4 to DocBook 5 documents and transform DocBook documents to WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, EPUB and EPUB 3.

DocBook4 to WebHelp Output

DocBook 4 documents can be transformed into WebHelp systems, such as:

WebHelp Output

To publish DocBook 4 to WebHelp, follow these steps:

1. Click the **Configure Transformation Scenario(s)** action from the toolbar.
2. Select the **DocBook WebHelp** scenario from the **DocBook 4** section.
3. Click **Apply associated**.

When the **DocBook WebHelp** transformation is complete, the output is automatically opened in your default browser.

WebHelp With Feedback Output

To publish DocBook 4 to WebHelp With Feedback, follow these steps:

1. Click **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp with Feedback** scenario from the **DocBook 4** section.
3. Click **Apply associated**.
4. Enter the documentation product ID and the documentation version.

When the **DocBook WebHelp with Feedback** transformation is complete, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

For further information about all the DocBook transformation parameters, go to *http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html*.

To watch our video demonstration about the feedback-enabled WebHelp system, go to *http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html*.

WebHelp Mobile Output

To generate a mobile WebHelp system from your DocBook 4 document, follow these steps:

1. Click **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp - Mobile** scenario from the **DocBook 4** section.
3. Click **Apply associated**.

When the **DocBook WebHelp - Mobile** transformation is complete, the output is automatically opened in your default browser.

To further customize these out-of-the-box transformation, you can edit the following most commonly used parameters:

`webhelp.copyright`

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

`webhelp.footer.file`

Path to an XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, Google Analytics, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
    <div id="fb-root"/>
    <script>
            <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
 return;
             js = d.createElement(s); js.id = id; js.src =
"//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
             fjs.parentNode.insertBefore(js, fjs); }(document, 'script', 'facebook-jssdk')); -->
    </script>
   <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

`webhelp.footer.include`

Specifies whether or not to include footer in each WebHelp page. Possible values: `yes`, `no`. If set to `no`, no footer is added to the WebHelp pages. If set to `yes` and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen XML Editor plugin footer is inserted in each WebHelp page.

`webhelp.logo.image` (not available for the WebHelp Mobile transformation)

Specifies a path to an image displayed as a logo in the left side of the output header.

`webhelp.logo.image.target.url` (not available for the WebHelp Mobile transformation)

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

`webhelp.search.ranking` (not available for the WebHelp Mobile transformation)

If this parameter is set to `false` then the *5-star rating mechanism* is no longer included in the search results that are displayed on the **Search** tab (default setting is `true`).

`webhelp.search.japanese.dictionary`

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Editor plugin uses for indexing Japanese content in the WebHelp pages. This indexer does not come bundled with Oxygen XML Editor plugin or the Oxygen XML WebHelp plugin. To use it, you need to download it from *http://mvnrepository.com/artifact/org.apache.lucene/lucene-analyzers-kuromoji/4.0.0* and add it in the *DITA_OT_DIR*`/plugins/com.oxygenxml.webhelp/lib` directory.

`use.stemming`

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

`clean.output`

Deletes all files from the output folder before the transformation is performed (only `no` and `yes` values are valid and the default value is `no`).

`args.default.language`

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

`l10n.gentext.default.language`

This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.

`webhelp.product.id` (available only for the WebHelp with Feedback transformation)

This parameter specifies a short name for the documentation target, or product (for example, `mobile-phone-user-guide`, `hvac-installation-guide`).

Note: You can deploy documentation for multiple products on the same server.

Note: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

`webhelp.product.version` (available only for the WebHelp with Feedback transformation)

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.

Note: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

`webhelp.skin.css` (not available for the WebHelp Mobile transformation)

Path to a CSS file that sets the style theme in the output WebHelp pages. It can be one of the predefined skin CSS from the `OXYGEN_INSTALL_DIR\frameworks\docbook\xsl\com.oxygenxml.webhelp\predefined-skins` directory, or it can be a custom skin CSS generated with the *Oxygen Skin Builder* web application.

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 (for example, `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 XML Editor plugin and set the customization layer in *the **XSL URL** property of the scenario*.

DocBook to EPUB Transformation

The EPUB specification recommends the use of *OpenType* fonts (recognized by their `.otf` file extension) when possible. To use a specific font, follow these steps:

1. Declare it in your CSS file, as in the following example:

```
@font-face {
font-family: "MyFont";
font-weight: bold;
font-style: normal;
src: url(fonts/MyFont.otf);
}
```

2. In the CSS, specify 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;
}
```

3. In your DocBook to EPUB transformation, set the `epub.embedded.fonts` parameter to `fonts/MyFont.otf`. If you need to provide more files, use commas to separate their file paths.

Note: The `html.stylesheet` parameter allows you to include a custom CSS in the output EPUB.

DocBook 4 Templates

Default templates are available in the *New File* wizard. You and can use them to create a skeletal form of a DocBook 4 book or article. These templates are stored in the `[OXYGEN_DIR]/frameworks/docbook/templates/DocBook 4` folder.

Here are some of the DocBook 4 templates available when creating *new documents from templates*:

- **Article**
- **Article with MathML**
- **Article with SVG**
- **Article with XInclude**
- **Book**
- **Book with XInclude**
- **Chapter**
- **Section**
- **Set of Books**

Inserting an `olink` in DocBook Documents

An `olink` is a type of link between two DocBook XML documents.

The `olink` element is used 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, a *Mail Administrator Guide* with the document ID `MailAdminGuide` might contain a chapter about user accounts, like this:

```
<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`, as in the following example:

```
You may need to update your
<olink targetdoc="MailAdminGuide" targetptr="user_accounts">user accounts
</olink>
when you get a new machine.
```

To use an `olink` to create links between documents, follow these steps:

1. Decide what documents are to be included in the domain for cross referencing.

   A unique ID must be assigned to each document that will be referenced with an `olink`. It is usually added as an `id` (or `xml:id` for DocBook5) attribute to the root element of the document.

2. Decide on your output hierarchy.

   For creating links between documents, the relative locations of the output documents must be known. Before going further you must decide the names and locations of the 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.

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.

   The following is an example target database document. It structures the documents in the collection into a `sitemap` element that provides the relative locations of the outputs (HTML in this example). Then it pulls in the individual target data using system entity references to target data files that will be created in the next step.

```
<?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>
```

4. Generate the target data files by executing a DocBook transformation scenario.

   Before applying the transformation, you need to edit the transformation scenario, go to **Parameters**, and make sure the value of the `collect.xref.targets` parameter is set to `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 `targets.filename` parameter.

   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>
```

5. Insert `olink` elements in the DocBook documents.

   When editing a DocBook XML document in **Author** mode, the **Insert OLink** action is available in the **Link** drop-down menu from the toolbar. This action opens the **Insert OLink** dialog box that allows you to select the target of an `olink` from the list of all possible targets from a specified target database document (specified in the **Targetset URL** field). Once a **Targetset URL** is selected, the structure of the target documents is presented. For each target document (`targetdoc`), its content is displayed allowing you to easily identify the appropriate `targetptr`. You can also 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.

   In the following image, the target database document is called `target.xml`, *dbadmin* is selected for the target document (`targetdoc`), and *bldinit* is selected as the value for the `targetptr` attribute. Notice that you can also add XREF text into the `olink` by using the `xreftext` field.

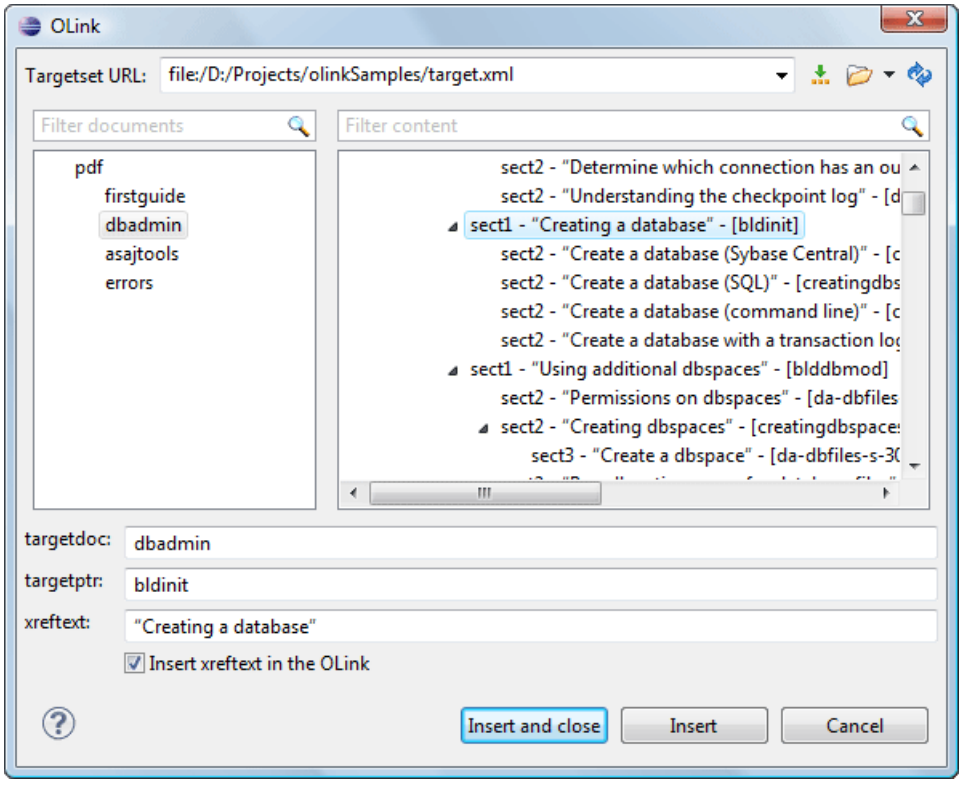


Figure 267: Insert OLink Dialog Box

6. Process a DocBook transformation each document to generate its output.
   a) Edit the transformation scenario and set the value of the `target.database.document` parameter to be the URL of the target database document.
   b) Apply the transformation scenario.

The DocBook 5 Document Type

A file is considered to be a DocBook 5 document when the namespace is *http://docbook.org/ns/docbook*.

The default schema that is used if one is not detected in the DocBook 5 file is `docbookxi.rng` and it is stored in `[OXYGEN_DIR]/frameworks/docbook/5.0/rng/`. Other types of schemas are also located in various folders inside the `[OXYGEN_DIR]/frameworks/docbook/5.0/` directory (for example, XML Schema files are located in the `xsd` folder).

The default CSS files used for rendering DocBook content in **Author** mode is stored in `[OXYGEN_DIR]/frameworks/docbook/css/`.

The default XML catalog, `catalog.xml`, is stored in `[OXYGEN_DIR]/frameworks/docbook/5.0/`.

Note: A default XML catalog and schema files for experimental DocBook 5.1 documents are also available in the `[OXYGEN_DIR]/frameworks/docbook/5.1/` directory.

To watch our video demonstration about editing DocBook documents, go to *http://oxygenxml.com/demo/DocBook_Editing_in_Author.html*.

DocBook 5 Author Mode Actions

A variety of actions are available in the DocBook 5 framework that can be added to the **DocBook5** menu, the **Author Custom Actions** toolbar, the contextual menu, and the **Content Completion Assistant**.

DocBook 5 Toolbar Actions

The following default actions are readily available on the **DocBook (Author Custom Actions)** toolbar when editing in **Author** mode (by default, most of them are also available in the **DocBook5** menu and in various submenus of the contextual menu):

Bold

Emphasizes the selected text by surrounding it with `<emphasis role="bold">` tag. You can use this action on multiple non-contiguous selections.

Italic

Emphasizes the selected text by surrounding it with `<emphasis role="italic">` tag. You can use this action on multiple non-contiguous selections.

Underline

Emphasizes the selected text by surrounding it with `<emphasis role="underline">` tag. You can use this action on multiple non-contiguous selections.

Link Actions Drop-Down Menu

The following link actions are available from this menu:

Cross reference (link)

Opens a dialog box that allows you to select a URL to insert as a hypertext link.

Cross reference (xref)

Inserts a cross reference to other parts of the document.

Web Link (ulink)

Inserts a link that addresses its target with a URL (Universal Resource Locator).

Insert OLink

Opens an **OLink** dialog box that allows you to insert a link that addresses its target indirectly, using the `targetdoc` and `targetptr` values that are present in a ***Targetset*** file.

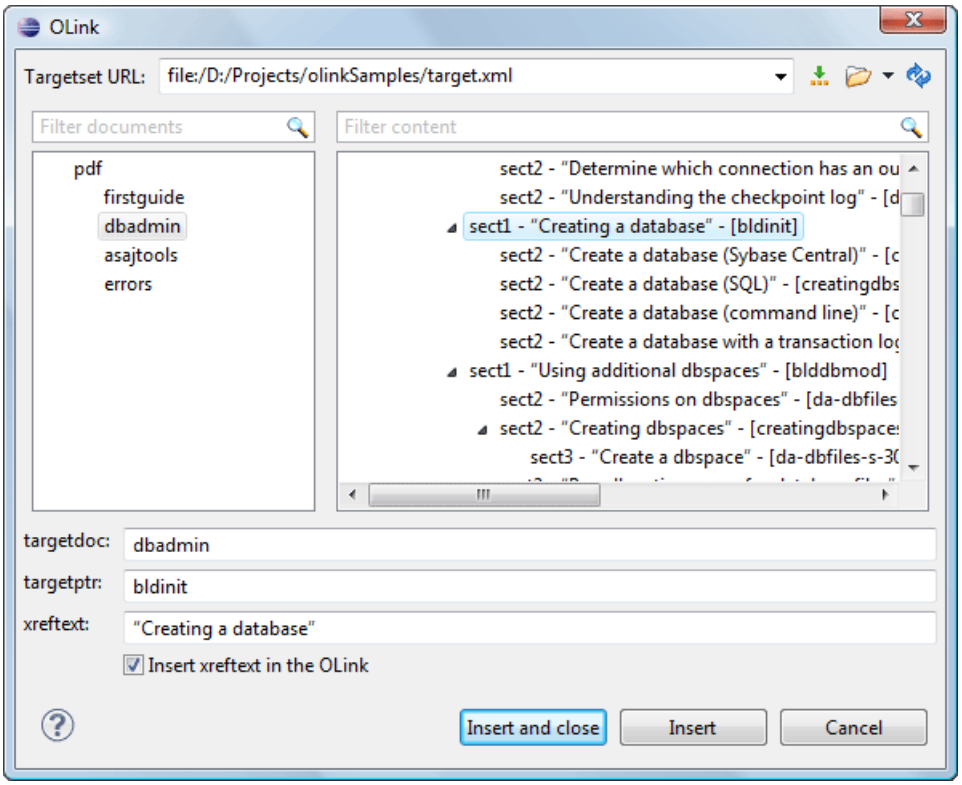


Figure 268: Insert OLink Dialog Box

After you choose the **Targetset URL**, the structure of the target documents is presented. For each target document (`targetdoc`), its content is displayed allowing you to easily identify the `targetptr` for the `olink` element that will be inserted. You can also use the search fields to quickly identify a target. If you already know the values for **targetdoc** and **targetptr**, you can insert them directly in the corresponding fields. You can also edit an `olink` using the **Edit OLink** action that is available on the contextual menu. The last used **Targetset** URL will be used to identify the edited target.

To insert XREF text into the `olink`, enter the text in the **xreftext** field and make sure the **Insert xreftext in the OLink** option is enabled.

Insert URI

Inserts an URI element. The URI identifies a Uniform Resource Identifier (URI) in content.

Insert Image

Inserts an image reference at the cursor position. Depending on the current location, an image-type element is inserted.

Insert XInclude

Opens a dialog box that allows you to browse and select content to be included and automatically generates the corresponding XInclude instruction.

Insert Section Drop-Down Menu

The following actions are available from this menu:

Insert Section

Inserts a new section or subsection in the document, depending on the current context. For example, if the current context is `sect1`, then a `sect2` is inserted. By default, this action also inserts a `para` element as a child node. The `para` element can be deleted if it is not needed.

Promote Section

Inserts the current node as a sibling of the parent node.

Demote Section

Inserts the current node a child of the previous node.

Insert Paragraph

Insert a new paragraph element at current cursor position.

Insert Equation

Opens the **XML Fragment Editor** that allows you to insert and *edit MathML notations*.

Insert List Item

Inserts a list item in the current list type.

Insert Ordered List

Inserts an ordered list at the cursor position. A child list item is also automatically inserted by default.

Insert Itemized List

Inserts an itemized list at the cursor position. A child list item is also automatically inserted by default.

Insert Variable List

Inserts a DocBook variable list. A child list item is also inserted automatically by default.

Insert Procedure List

Inserts a DocBook `procedure` element. A step `child` item is also inserted automatically.

Sort

Sorts cells or list items in a table.

Insert Table

Opens a dialog box 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 Below

Inserts a new table row with empty cells below the current row. This action is available when the cursor is positioned inside a table.

Insert Column After

Inserts a new table column with empty cells after the current column. This action is available when the cursor is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the cursor is positioned between two cells, Oxygen XML Editor plugin a new cell at cursor position. If the cursor is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at cursor position.

Delete Row

Deletes the table row located at cursor position.

Table Properties

Opens the **Table properties** dialog box that allows you to configure properties of a table (such as frame borders).

Table Join/Split Drop-Down Menu

The following join and split actions are available from this menu:

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 cursor is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current cursor 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 cursor 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 Editor plugin 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 cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

DocBook5 Menu Actions

In addition, the following default actions are available in the **DocBook5** menu when editing in **Author** mode:

Paste special submenu

This submenu includes the following special paste actions that are specific to the DocBook 5 framework:

Paste As XInclude

Allows you to create an `xi:include` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Paste as link

Allows you to create a `link` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Paste as xref

Allows you to create an `xref` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu:

Insert Row Above

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

Insert Column Before

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

ID Options

Opens the **ID Options** dialog box that allows you to configure options for generating IDs in **Author** mode. The dialog box includes the following:

ID Pattern

The pattern for the ID values that will be generated. This text field can be customized using constant strings and most of the supported *Editor Variables* on page 594.

Element name or class value to generate ID for

The elements for which ID values will be generated, specified using class attribute values. To customize the list, use the **Add**, **Edit**, or **Remove** buttons.

Auto generate IDs for elements

If enabled, Oxygen XML Editor plugin will automatically generate unique IDs for the elements listed in this dialog box when they are created in **Author** mode.

Remove IDs when copying content in the same document

Allows you to control whether or not pasted elements that are listed in this dialog box should retain their existing IDs when copying content in the same document. To retain the element IDs, disable this option.

Generate IDs

Oxygen XML Editor plugin generates unique IDs for the current element (or elements), depending on how the action is invoked:

- When invoked on a single selection, an ID is generated for the selected element at the cursor position.
- When invoked on a block of selected content, IDs are generated for all top-level elements and elements from the list in the **ID Options** dialog box that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

Refresh References

You can use this action to manually trigger a refresh and update of all referenced resources.

Tags display mode Submenu

Full Tags with Attributes - Displays full tag names with attributes for both block level and in-line level elements.

Full Tags - Displays full tag names without attributes for both block level and in-line level elements.

Block Tags - Displays full tag names for block level elements and simple tags without names for in-line level elements.

Inline Tags - Displays full tag names 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 and is as close as possible to a word-processor view.

Profiling/Conditional Text Submenu

Edit Profiling Attributes - Allows you to configure the *profiling attributes* and their values.

Show Profiling Colors and Styles - Select this option to turn on conditional styling.

Show Profiling Attributes - Select 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 - When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

List of all profiling condition sets that match the current document type - You can click a listed condition set to activate it.

Profiling Settings - Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

DocBook 5 Contextual Menu Actions

In addition to many of the *DocBook 5 toolbar actions* and the *general **Author** mode contextual menu actions*, the following DocBook 5 framework-specific actions are also available in the contextual menu when editing in **Author** mode:

Paste special submenu

This submenu includes the following special paste actions that are specific to the DocBook 5 framework:

Paste As XInclude

Allows you to create an `xi:include` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Paste as link

Allows you to create a `link` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Paste as xref

Allows you to create an `xref` element that references a DocBook element copied from **Author** mode. The operation fails if the copied element does not have a declared ID.

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu when the contextual menu is invoked on a table:

Insert Row Above

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

Insert Column Before

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

Link > Edit OLink

Opens a dialog box that allows you edit an existing OLink. See the **Insert OLink** action for more information.

Generate IDs

Oxygen XML Editor plugin generates unique IDs for the current element (or elements), depending on how the action is invoked:

- When invoked on a single selection, an ID is generated for the selected element at the cursor position.
- When invoked on a block of selected content, IDs are generated for all top-level elements and elements from the list in the **ID Options** dialog box that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

DocBook 5 Drag/Drop Actions

Dragging a file from *the **Project** view* or *__DITA Maps Manager__ view* and dropping it into a DocBook 5 document that is edited in **Author** mode, creates a link to the dragged file (the `link` 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 5 document inserts an image element (the `imageobject` DocBook element with an `imagedata` child element and a `fileref` attribute) at the drop location, similar to the **Insert Image** toolbar action.

DocBook 5 Transformation Scenarios

Default transformation scenarios allow you to transform DocBook 5 documents to WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, EPUB, and EPUB 3.

DocBook 5 to WebHelp Output

DocBook 5 documents can be transformed into WebHelp systems, such as:

WebHelp Output

To publish DocBook 5 to WebHelp, follow these steps:

1. Click the **Configure Transformation Scenario(s)** action from the toolbar.

2. Select the **DocBook WebHelp** scenario from the **DocBook 5** section.
3. Click **Apply associated**.

When the **DocBook WebHelp** transformation is complete, the output is automatically opened in your default browser.

WebHelp With Feedback Output

To publish DocBook 5 to WebHelp With Feedback, follow these steps:

1. Click **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp with Feedback** scenario from the **DocBook 5** section.
3. Click **Apply associated**.
4. Enter the documentation product ID and the documentation version.

When the **DocBook WebHelp with Feedback** transformation is complete, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

For further information about all the DocBook transformation parameters, go to *http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html*.

To watch our video demonstration about the feedback-enabled WebHelp system, go to *http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html*.

WebHelp Mobile Output

To generate a mobile WebHelp system from your DocBook 5 document, follow these steps:

1. Click **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp - Mobile** scenario from the **DocBook 5** section.
3. Click **Apply associated**.

When the **DocBook WebHelp - Mobile** transformation is complete, the output is automatically opened in your default browser.

To further customize these out-of-the-box transformation, you can edit the following most commonly used parameters:

`webhelp.copyright`

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

`webhelp.footer.file`

Path to an XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, Google Analytics, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
    <div id="fb-root"/>
    <script>
            <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
 return;
             js = d.createElement(s); js.id = id; js.src =
"//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
             fjs.parentNode.insertBefore(js, fjs); }(document, 'script', 'facebook-jssdk')); -->
    </script>
   <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

`webhelp.footer.include`

Specifies whether or not to include footer in each WebHelp page. Possible values: `yes`, `no`. If set to `no`, no footer is added to the WebHelp pages. If set to `yes` and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen XML Editor plugin footer is inserted in each WebHelp page.

webhelp.logo.image (not available for the WebHelp Mobile transformation)

Specifies a path to an image displayed as a logo in the left side of the output header.

webhelp.logo.image.target.url (not available for the WebHelp Mobile transformation)

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

webhelp.search.ranking (not available for the WebHelp Mobile transformation)

If this parameter is set to `false` then the *5-star rating mechanism* is no longer included in the search results that are displayed on the **Search** tab (default setting is `true`).

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Editor plugin uses for indexing Japanese content in the WebHelp pages. This indexer does not come bundled with Oxygen XML Editor plugin or the Oxygen XML WebHelp plugin. To use it, you need to download it from *http://mvnrepository.com/artifact/org.apache.lucene/lucene-analyzers-kuromoji/4.0.0* and add it in the *DITA_OT_DIR*`/plugins/com.oxygenxml.webhelp/lib` directory.

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

clean.output

Deletes all files from the output folder before the transformation is performed (only `no` and `yes` values are valid and the default value is `no`).

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

l10n.gentext.default.language

This parameter is used to identify the correct stemmer that differs from language to language. For example, for English the value of this parameter is `en` or for French it is `fr`, and so on.

webhelp.product.id (available only for the WebHelp with Feedback transformation)

This parameter specifies a short name for the documentation target, or product (for example, `mobile-phone-user-guide`, `hvac-installation-guide`).

Note: You can deploy documentation for multiple products on the same server.

Note: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

webhelp.product.version (available only for the WebHelp with Feedback transformation)

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.

Note: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

webhelp.skin.css (not available for the WebHelp Mobile transformation)

Path to a CSS file that sets the style theme in the output WebHelp pages. It can be one of the predefined skin CSS from the `OXYGEN_INSTALL_DIR\frameworks\docbook\xsl\com.oxygenxml.webhelp\predefined-skins` directory, or it can be a custom skin CSS generated with the *Oxygen Skin Builder* web application.

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 (for example, `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 XML Editor plugin and set the customization layer in *the **XSL URL** property of the scenario*.

DocBook to EPUB Transformation

The EPUB specification recommends the use of *OpenType* fonts (recognized by their `.otf` file extension) when possible. To use a specific font, follow these steps:

1. Declare it in your CSS file, as in the following example:

```
@font-face {
font-family: "MyFont";
font-weight: bold;
font-style: normal;
src: url(fonts/MyFont.otf);
}
```

2. In the CSS, specify 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;
}
```

3. In your DocBook to EPUB transformation, set the `epub.embedded.fonts` parameter to `fonts/MyFont.otf`. If you need to provide more files, use commas to separate their file paths.

Note: The `html.stylesheet` parameter allows you to include a custom CSS in the output EPUB.

DocBook 5 Templates

Default templates are available in the *New File wizard* and can be used for easily creating a skeletal form of a DocBook 5 book or article. These templates are stored in the `[OXYGEN_DIR]/frameworks/docbook/templates/DocBook 5` folder.

Here are some of the DocBook 5 templates available when creating *new documents from templates*:

- **Article**
- **Article with MathML**
- **Article with SVG**
- **Article with XInclude**
- **Book**
- **Book with XInclude**
- **Chapter**
- **Section**
- **Set of Books**

Some experimental DocBook 5.1 templates are also available and are stored in the `[OXYGEN_DIR]/frameworks/docbook/templates/DocBook 5.1 (Experimental)` folder. They include the following:

- **Assembly**
- **Topic**

Inserting an `olink` in DocBook Documents

An `olink` is a type of link between two DocBook XML documents.

The `olink` element is used 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, a *Mail Administrator Guide* with the document ID `MailAdminGuide` might contain a chapter about user accounts, like this:

```
<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`, as in the following example:

```
You may need to update your
<olink targetdoc="MailAdminGuide" targetptr="user_accounts">user accounts
</olink>
when you get a new machine.
```

To use an `olink` to create links between documents, follow these steps:

1. Decide what documents are to be included in the domain for cross referencing.

   A unique ID must be assigned to each document that will be referenced with an `olink`. It is usually added as an `id` (or `xml:id` for DocBook5) attribute to the root element of the document.

2. Decide on your output hierarchy.

   For creating links between documents, the relative locations of the output documents must be known. Before going further you must decide the names and locations of the 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.

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.

The following is an example target database document. It structures the documents in the collection into a `sitemap` element that provides the relative locations of the outputs (HTML in this example). Then it pulls in the individual target data using system entity references to target data files that will be created in the next step.

```
<?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>
```

4. Generate the target data files by executing a DocBook transformation scenario.

   Before applying the transformation, you need to edit the transformation scenario, go to **Parameters**, and make sure the value of the `collect.xref.targets` parameter is set to `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 `targets.filename` parameter.

   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>
```

5. Insert `olink` elements in the DocBook documents.

When editing a DocBook XML document in **Author** mode, the **Insert OLink** action is available in the **Link** drop-down menu from the toolbar. This action opens the **Insert OLink** dialog box that allows you to select the target of an `olink` from the list of all possible targets from a specified target database document (specified in the **Targetset URL** field). Once a **Targetset URL** is selected, the structure of the target documents is presented. For each target document (`targetdoc`), its content is displayed allowing you to easily identify the appropriate `targetptr`. You can also 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.

In the following image, the target database document is called `target.xml`, *dbadmin* is selected for the target document (`targetdoc`), and *bldinit* is selected as the value for the `targetptr` attribute. Notice that you can also add XREF text into the `olink` by using the `xreftext` field.

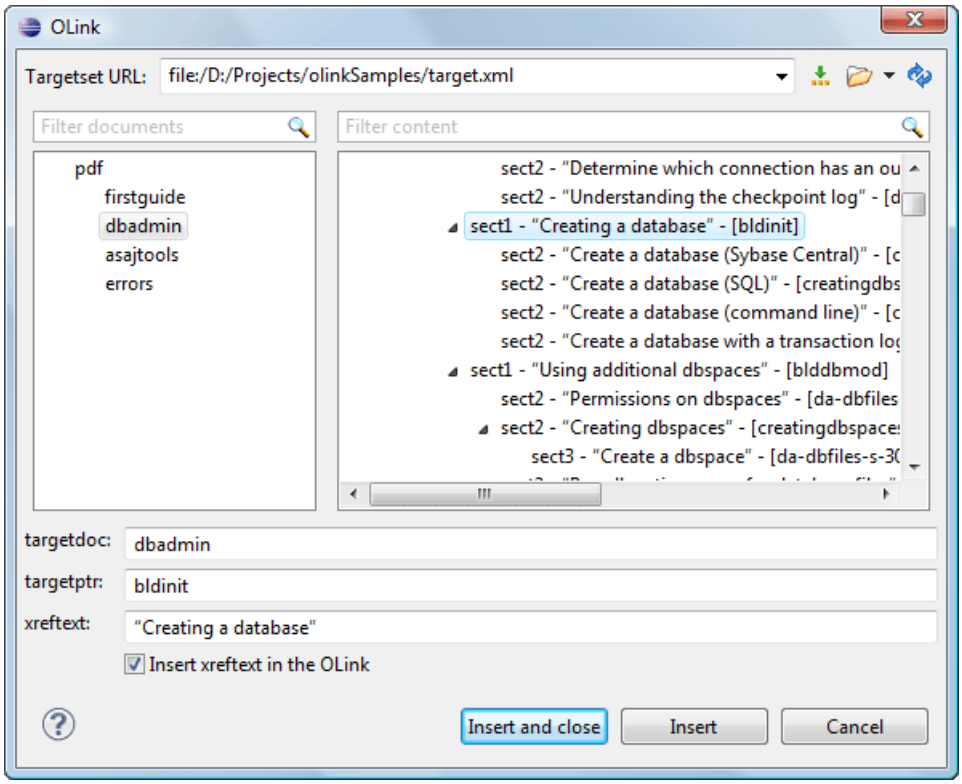


Figure 269: Insert OLink Dialog Box

6. Process a DocBook transformation each document to generate its output.
   a) Edit the transformation scenario and set the value of the `target.database.document` parameter to be the URL of the target database document.
   b) Apply the transformation scenario.

The DITA Topics Document Type

The Darwin Information Typing Architecture (DITA) is an XML-based architecture oriented to authoring, producing, and delivering technical information. It divides content into small, self-contained topics that you can reuse in various deliverables. The extensibility of DITA permits organizations to define specific information structures while still using standard tools to work with them. Oxygen XML Editor plugin provides schema-driven (DTD, RNG, XSD) templates for DITA documents.

A file is considered to be a DITA topic document when one of the following conditions are true:

- The root element name is one of the following: `concept`, `task`, `reference`, `dita`, or `topic`.
- The PUBLIC ID of the document is a PUBLIC ID for the elements listed above.
- The root element of the file has an attribute named `DITAArchVersion` for the "http://dita.oasis-open.org/architecture/2005/" namespace. This enhanced case of matching is only applied when the

Enable DTD/XML Schema processing in document type detection option is enabled from the ***Document Type Association*** *preferences page*.

Default schemas that are used if one is not detected in the DITA documents are stored in the various folders inside `[OXYGEN_DIR]/frameworks/dita/DITA-OT/dtd/` or `[OXYGEN_DIR]/frameworks/dita/DITA-OT/schema/` (or if you are using DITA 2.x, inside `[OXYGEN_DIR]/frameworks/dita/DITA-OT2.x/dtd/` or `[OXYGEN_DIR]/frameworks/dita/DITA-OT2.x/schema/`).

The default CSS files used for rendering DITA content in **Author** mode are stored in `[OXYGEN_DIR]/frameworks/dita/css/`.

The default catalogs for the DITA topic document type are as follows:

- `[OXYGEN_DIR]/frameworks/dita/catalog.xml`
- `[OXYGEN_DIR]/frameworks/dita/DITA-OT/catalog-dita.xml` (or if you are using DITA 2.x, `[OXYGEN_DIR]/frameworks/dita/DITA-OT2.x/catalog-dita.xml`)
- `[OXYGEN_DIR]/frameworks/dita/plugin/catalog.xml`
- `[OXYGEN_DIR]/frameworks/dita/styleguide/catalog.xml`

DITA Author Mode Actions

A variety of actions are available in the DITA framework that can be added to the **DITA** menu, the **Author Custom Actions** toolbar, the contextual menu, and the **Content Completion Assistant**.

DITA Toolbar Actions

The following default actions are readily available on the **DITA (Author Custom Actions)** toolbar when editing in **Author** mode (by default, most of them are also available in the **DITA** menu and in various submenus of the contextual menu):

Bold

Surrounds the selected text with a `b` tag. You can use this action on multiple non-contiguous selections.

Italic

Surrounds the selected text with an `i` tag. You can use this action on multiple non-contiguous selections.

Underline

Surrounds the selected text with a `u` tag. You can use this action on multiple non-contiguous selections.

Link Actions Drop-Down Menu

The following link actions are available from this menu:

Cross Reference

Opens the ***Cross Reference (xref)*** *dialog box* that allows you to insert a link to a target resource at the current location within a document. The target resource can be the location of a file or a key that is already defined in your DITA map structure. Once the target resource has been selected, you can also target specific elements within that resource. For more information, see the *Linking in DITA Topics* on page 450 topic.

File Reference

Opens the ***File Reference*** *dialog box* that allows you to insert a link to a target file resource at the current location within a document. The target resource can be the location of a file or a key that is already defined in your DITA map structure. For more information, see the *Linking in DITA Topics* on page 450 topic.

Web Link

Opens the ***Web Link*** *dialog box* that allows you to insert a link to a target web-related resource at the current location within a document. The target resource can be a URL or a key that is already defined in your DITA map structure. For more information, see the *Linking in DITA Topics* on page 450 topic.

Related Link to Topic

Opens the ***Cross Reference (xref)*** *dialog box* that allows you to insert a link to a target resource in a related links section at the bottom of the current document. The target resource can be the location of a file or a key that

is already defined in your DITA map structure. Once the target resource has been selected, you can also target specific elements within that resource. If a related links section does not already exist, this action creates one. For more information, see the *Linking in DITA Topics* on page 450 topic.

Related Link to File

Opens the ***File Reference** dialog box* that allows you to insert a link to a target file resource in a related links section at the bottom of the current document. The target resource can be the location of a file or a key that is already defined in your DITA map structure. If a related links section does not already exist, this action creates one. For more information, see the *Linking in DITA Topics* on page 450 topic.

Related Link to Web Page

Opens the ***Web Link** dialog box* that allows you to insert a link to a target web-related resource in a related links section at the bottom of the current document. The target resource can be a URL or a key that is already defined in your DITA map structure. If a related links section does not already exist, this action creates one. For more information, see the *Linking in DITA Topics* on page 450 topic.

Insert Image

Opens the ***Insert Image** dialog box* that allows you to configure the properties of an image to be inserted into a DITA document at the cursor position.

Insert Section Drop-Down Menu

The following insert actions are available from this menu:

Insert Section

Inserts a new `section` element in the document, depending on the current context.

Insert Concept

Inserts a new `concept` element, depending on the current context. Concepts provide background information that users must know before they can successfully work with a product or interface.

Insert Task

Inserts a new `task` element, depending on the current context. 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.

Insert Topic

Inserts a new `topic` element, depending on the current context. Topics are the basic units of DITA content and are usually organized around a single subject.

Insert Reference

Inserts a new `reference` element, depending on the current context. A reference is a top-level container for a reference topic.

Insert Paragraph

Inserts a new paragraph at current cursor position.

Reuse Content

This action provides a mechanism for reusing content fragments. It opens the ***Reuse Content** dialog box* that allows you to insert several types of references to reusable content at the cursor position. The types of references that you can insert using this dialog box include *content references (`conref`)*, *content key references (`conkeyref`)*, or *key references to metadata (`keyref`)*.

Insert step or list item

Inserts a new list or step item in the current list type.

Insert Unordered List

Inserts an unordered list at the cursor position. A child list item is also automatically inserted by default.

Insert Ordered List

Inserts an ordered list at the cursor position. A child list item is also automatically inserted by default.

Sort

Sorts cells or list items in a table.

Insert Table

Opens a dialog box 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 Below

Inserts a new table row with empty cells below the current row. This action is available when the cursor is positioned inside a table.

Insert Column After

Inserts a new table column with empty cells after the current column. This action is available when the cursor is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the cursor is positioned between two cells, Oxygen XML Editor plugin a new cell at cursor position. If the cursor is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at cursor position.

Delete Row

Deletes the table row located at cursor position.

Table Properties

Opens the **Table properties** dialog box that allows you to configure properties of a table (such as frame borders).

Table Join/Split Drop-Down Menu

The following join and split actions are available from this menu:

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 cursor is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current cursor 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 cursor 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 Editor plugin 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 cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

DITA Menu Actions

In addition to the *DITA toolbar actions*, the following default actions are available in the **DITA** menu when editing in **Author** mode:

Reuse Content

This action provides a mechanism for reusing content fragments. It opens the ***Reuse Content** dialog box* that allows you to insert several types of references to reusable content at the cursor position. The types of references that you can insert using this dialog box include *content references (`conref`)*, *content key references (`conkeyref`)*, or *key references to metadata (`keyref`)*.

Push Current Element

Opens the ***Push current element** dialog box* that allows content from a source topic to be inserted into another topic without any special coding in the topic where the content will be re-used.

Insert/Edit Content Reference

Opens the **Insert/Edit Content Reference** dialog box that allows you to select/change the source location (or key) and source element of a content reference (or content key reference), and the reference details (`conref`/`conkeyref` and `conrefend` attributes). See the *Reuse Content Dialog Box* on page 439 topic for more information.

Replace Reference with content

Replaces the referenced fragment (`conref` or `conkeyref`) at the cursor position with its content. This action is useful if you want to make changes to the content in the currently edited document without changing the referenced fragment in its source location.

Remove Content Reference

Removes the content reference (`conref` or `conkeyref`) inside the element at the cursor position.

Create Reusable Component

Creates a reusable component from the selected fragment of text. For more information, see *Creating a Reusable Content Component* on page 447.

Insert Reusable Component

Inserts a reusable component at cursor location. For more information, see *Inserting a Reusable Content Component* on page 447.

Paste special submenu

This submenu includes the following special paste actions that are specific to the DITA framework:

Paste as content reference

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 reference 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:

1. Set the `id` attribute of the element holding the content you want to reference.
2. Open the DITA map in the **DITA Maps Manager** view and make sure that the **Root map** combo box points to the correct map that stores the keys.

3. Right click the topic that holds the content you want to reference, select **Edit Properties**, and enter a value in the **Keys** field.

Paste as link

Inserts a `link` element or an `xref` (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 reference. To use this action, do the following:

1. Set the `id` attribute of the element that you want to reference.
2. Open the DITA map in the **DITA Maps Manager** view and make sure that the **Root map** combo box points to the correct map that stores the keys.
3. Right click the topic that holds the content you want to reference, select **Edit Properties**, and enter a value in the **Keys** field.

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu:

 **Insert Row Above**

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

 **Insert Column Before**

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

Insert > Σ Insert Equation

Opens the **XML Fragment Editor** that allows you to insert and *edit MathML notations*.

ID Options

Opens the **ID Options** dialog box that allows you to configure options for generating IDs in **Author** mode. The dialog box includes the following:

ID Pattern

The pattern for the ID values that will be generated. This text field can be customized using constant strings and most of the supported *Editor Variables* on page 594.

Element name or class value to generate ID for

The elements for which ID values will be generated, specified using class attribute values. To customize the list, use the **Add**, **Edit**, or **Remove** buttons.

Auto generate IDs for elements

If enabled, Oxygen XML Editor plugin will automatically generate unique IDs for the elements listed in this dialog box when they are created in **Author** mode.

Remove IDs when copying content in the same document

Allows you to control whether or not pasted elements that are listed in this dialog box should retain their existing IDs when copying content in the same document. To retain the element IDs, disable this option.

Generate IDs

Oxygen XML Editor plugin generates unique IDs for the current element (or elements), depending on how the action is invoked:

- When invoked on a single selection, an ID is generated for the selected element at the cursor position.
- When invoked on a block of selected content, IDs are generated for all top-level elements and elements from the list in the **ID Options** dialog box that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

Style Guide

Opens the **DITA Style Guide Best Practices for Authors** in your browser and displays a topic that is relevant to the element at the cursor 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 that are displayed by the **Content Completion Assistant**.

Refresh References

You can use this action to manually trigger a refresh and update of all referenced resources.

Tags display mode Submenu

Full Tags with Attributes - Displays full tag names with attributes for both block level and in-line level elements.

Full Tags - Displays full tag names without attributes for both block level and in-line level elements.

Block Tags - Displays full tag names for block level elements and simple tags without names for in-line level elements.

Inline Tags - Displays full tag names 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 and is as close as possible to a word-processor view.

Profiling/Conditional Text Submenu

Edit Profiling Attributes - Allows you to configure the *profiling attributes* and their values.

Show Profiling Colors and Styles - Select this option to turn on conditional styling.

Show Profiling Attributes - Select 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 - When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

List of all profiling condition sets that match the current document type - You can click a listed condition set to activate it.

Profiling Settings - Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

DITA Contextual Menu Actions

In addition to many of the *DITA toolbar actions* and the *general **Author** mode contextual menu actions*, the following DITA framework-specific actions are also available in the contextual menu when editing in **Author** mode:

Paste special submenu

This submenu includes the following special paste actions that are specific to the DITA framework:

Paste as content reference

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 reference 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:

1. Set the `id` attribute of the element holding the content you want to reference.
2. Open the DITA map in the **DITA Maps Manager** view and make sure that the **Root map** combo box points to the correct map that stores the keys.
3. Right click the topic that holds the content you want to reference, select **Edit Properties**, and enter a value in the **Keys** field.

Paste as link

Inserts a `link` element or an `xref` (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 reference. To use this action, do the following:

1. Set the `id` attribute of the element that you want to reference.
2. Open the DITA map in the **DITA Maps Manager** view and make sure that the **Root map** combo box points to the correct map that stores the keys.
3. Right click the topic that holds the content you want to reference, select **Edit Properties**, and enter a value in the **Keys** field.

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu when the contextual menu is invoked on a table:

 **Insert Row Above**

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

 **Insert Column Before**

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

Insert > Σ Insert Equation

Opens the **XML Fragment Editor** that allows you to insert and *edit MathML notations*.

Generate IDs

Oxygen XML Editor plugin generates unique IDs for the current element (or elements), depending on how the action is invoked:

- When invoked on a single selection, an ID is generated for the selected element at the cursor position.
- When invoked on a block of selected content, IDs are generated for all top-level elements and elements from the list in the **ID Options** dialog box that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

Reuse submenu

This submenu includes the following actions in regards to reusing content in DITA:

Push Current Element

Opens the ***Push current element** dialog box* that allows content from a source topic to be inserted into another topic without any special coding in the topic where the content will be re-used.

Insert/Edit Content Reference

Opens the **Insert/Edit Content Reference** dialog box that allows you to select/change the source location (or key) and source element of a content reference (or content key reference), and the reference details (`conref`/`conkeyref` and `conrefend` attributes). See the *Reuse Content Dialog Box* on page 439 topic for more information.

Replace Reference with content

Replaces the referenced fragment (`conref` or `conkeyref`) at the cursor position with its content. This action is useful if you want to make changes to the content in the currently edited document without changing the referenced fragment in its source location.

Remove Content Reference

Removes the content reference (`conref` or `conkeyref`) inside the element at the cursor position.

Create Reusable Component

Creates a reusable component from the selected fragment of text. For more information, see *Creating a Reusable Content Component* on page 447.

Insert Reusable Component

Inserts a reusable component at cursor location. For more information, see *Inserting a Reusable Content Component* on page 447.

Search References

Finds the references to the `id` attribute value of the selected element in all the topics from the current DITA map (opened in the **DITA Maps Manager** view). The default shortcut of the action is **Ctrl Shift G** and can be changed in the **DITA Topic** document type.

About Element submenu

This submenu includes the following actions:

Style Guide

Opens the **DITA Style Guide Best Practices for Authors** in your browser and displays a topic that is relevant to the element at the cursor 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 that are displayed by the **Content Completion Assistant**.

Browse reference manual

Opens a reference to the documentation of the XML element closest to the cursor position in a web browser.

Show Definition

Moves the cursor to the definition of the current element.

DITA Drag/Drop Actions

Dragging a file from *the **Project** view* or ***DITA Maps Manager** view* and dropping it into a DITA document that is edited in **Author** mode, creates a link to the dragged file (the `xref` DITA element with the `href` attribute) 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 DITA document inserts an image element (the `image` DITA element with the `href` attribute) at the drop location.

DITA Transformation Scenarios

The following default transformation scenarios are available for DITA Topics:

- **DITA XHTML** - Transforms a DITA topic to XHTML using DITA Open Toolkit.
- **DITA PDF** - Transforms a DITA topic to PDF using the DITA Open Toolkit and the Apache FOP engine.

DITA Templates

The default templates available for DITA topics are stored in various sub-folders inside the `[OXYGEN_DIR]/frameworks/dita/templates/` folder. They can be used for easily creating a DITA `concept`, `reference`, `task`, `topic`, or other DITA documents.

Here are some of the DITA templates available when creating *new documents from templates*:

- **Composite** - New DITA Composite
- **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 (available when using the *built-in DITA-OT 1.8 version*):

- **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

Note: If you select the ***built-in DITA-OT 2.x (with experimental DITA 1.3 support)*** *option in the DITA Preferences Page*, you will also have access to some experimental templates for DITA 1.3.

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 Association*** *preferences page* is enabled.

Default schemas that are used if one is not detected in the DITA map document are stored in the various folders inside `[OXYGEN_DIR]/frameworks/dita/DITA-OT/dtd/` or `[OXYGEN_DIR]/frameworks/dita/DITA-OT/schema/` (or if you are using DITA 2.x, inside `[OXYGEN_DIR]/frameworks/dita/DITA-OT2.x/dtd/` or `[OXYGEN_DIR]/frameworks/dita/DITA-OT2.x/schema/`).

The default CSS files used for rendering DITA content in **Author** mode are stored in `[OXYGEN_DIR]/frameworks/dita/css/`.

The default catalogs for the *DITA Map* document type are as follows:

- `[OXYGEN_DIR]/frameworks/dita/catalog.xml`
- `[OXYGEN_DIR]/frameworks/dita/DITA-OT/catalog-dita.xml` (or if you are using DITA 2.x, `[OXYGEN_DIR]/frameworks/dita/DITA-OT2.x/catalog-dita.xml`)

DITA Map Author Mode Actions

A variety of actions are available in the DITA Map framework that can be added to the **DITA** menu, the **Author Custom Actions** toolbar, the contextual menu, and the **Content Completion Assistant**.

DITA Map Toolbar and Menu Actions

When a DITA map is opened in **Author** mode, the following default actions are available on the **DITA Map Author Custom Actions** toolbar (by default, they are also available in the **DITA** menu and in various submenus of the contextual menu):

 **Insert New Topic**

Opens a ***New** DITA topic dialog box* that allows you to create a new topic and inserts a reference to it at the cursor position.

 **Insert Topic Reference**

Opens the ***Insert Reference** dialog box* that allows you to insert and configure a reference to a topic at the cursor position.

 **Reuse Content**

Opens the ***Reuse Content** dialog box* that allows you to insert and configure a content reference (`conref`), or a content key reference (`conkeyref`) at the cursor position.

 **Insert Topic Heading**

Opens the ***Insert Reference** dialog box* that allows you to insert a topic heading at the cursor position.

 **Insert Topic Group**

Opens the ***Insert Reference** dialog box* that allows you to insert a topic group at the cursor position.

 **Insert Relationship Table**

Opens a dialog box that allows you to configure the relationship table to be inserted. The dialog box allows you 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.

 **Relationship Table Properties**

Allows you to change the properties of rows in relationship tables.

 **Insert Relationship Row**

Inserts a new table row with empty cells. The action is available when the cursor position is inside a table.

 **Insert Relationship Column**

Inserts a new table column with empty cells after the current column. The action is available when the cursor position is inside a table.

 **Delete Relationship Column**

Deletes the table column where the cursor is located.

Delete Relationship Row

Deletes the table row where the cursor is located.

DITA Menu Actions

In addition, the following default actions are available in the **DITA** menu when editing a DITA map in **Author** mode:

Refresh References

You can use this action to manually trigger a refresh and update of all referenced resources.

Tags display mode Submenu

Full Tags with Attributes - Displays full tag names with attributes for both block level and in-line level elements.

Full Tags - Displays full tag names without attributes for both block level and in-line level elements.

Block Tags - Displays full tag names for block level elements and simple tags without names for in-line level elements.

Inline Tags - Displays full tag names 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 and is as close as possible to a word-processor view.

Profiling/Conditional Text Submenu

Edit Profiling Attributes - Allows you to configure the *profiling attributes* and their values.

Show Profiling Colors and Styles - Select this option to turn on conditional styling.

Show Profiling Attributes - Select 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 - When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

List of all profiling condition sets that match the current document type - You can click a listed condition set to activate it.

Profiling Settings - Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

DITA Map Contextual Menu Actions

In addition to many of the *DITA Map toolbar actions* and the *general **Author** mode contextual menu actions*, the following DITA map framework-specific action is also available in the contextual menu when editing in **Author** mode:

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).

DITA Map Drag/Drop Actions

Dragging a file from *the **Project** view* or *__DITA Maps Manager__ view* and dropping it into a DITA map document that is edited in **Author** mode creates a link to the dragged file (a `topicref` element, `chapter`, `part`, 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 the DITA-OT/plugins directory.
- **DITA Map Metrics Report** - Use this transformation scenario if you want to generate a DITA map statistics report containing information such as:
  - The number of processed maps and topics.
  - Content reuse percentage.
  - Number of elements, attributes, words, and characters used in the entire DITA map structure.
  - DITA conditional processing attributes used in the DITA maps.
  - Words count.
  - Information types such as number of containing maps, bookmaps, or topics.

Many more output formats are available by clicking the **New** button. The transformation process relies on the DITA Open Toolkit.

DITA Map to WebHelp Output

DITA maps can be transformed into WebHelp systems.

WebHelp Output

To publish a DITA map to WebHelp, follow these steps:

1. Click the **Configure Transformation Scenario(s)** action from the toolbar.
2. Select the **DITA Map WebHelp** scenario from the **DITA Map** section.
3. Click **Apply associated**.

   When the **DITA Map WebHelp** transformation is complete, the output is automatically opened in your default browser.

WebHelp With Feedback Output

To publish a DITA map as WebHelp with Feedback:

1. Click **Configure Transformation Scenarios**.
2. Select the **DITA Map WebHelp with Feedback** scenario from the **DITA Map** section.
3. Click **Apply associated**.
4. Enter the documentation product ID and the documentation version.

   When the **DITA Map WebHelp with Feedback** transformation is complete, your default browser opens the `installation.html` file. This file contains information about the output location, system requirements, installation instructions, and deployment of the output.

To watch our video demonstration about the feedback-enabled WebHelp system, go to *http://oxygenxml.com/demo/Feedback_Enabled_WebHelp.html*.

WebHelp Mobile Output

To generate a mobile WebHelp system from your DITA map:

1. From the **DITA Maps Manager** view click **Configure Transformation Scenarios**.
2. Select the **DITA Map WebHelp - Mobile** transformation scenario.
3. Click **Apply associated**.

When the **DITA Map WebHelp - Mobile** transformation is complete, the output is automatically opened in your default browser.

Once Oxygen XML Editor plugin finishes the transformation process, the output is automatically opened in your default browser.

To further customize these out-of-the-box transformation, you can edit the following most commonly used parameters:

`webhelp.sitemap.base.url`

Base URL for all the `loc` elements in the generated `sitemap.xml` file. The value of a `loc` element is computed as the relative file path from the `href` attribute of a `topicref` element from the DITA map, appended to this base URL value. The `loc` element is mandatory in `sitemap.xml`. If you leave this parameter set to its default empty value, then the `sitemap.xml` file is not generated.

`webhelp.sitemap.change.frequency`

The value of the `changefreq` element in the generated `sitemap.xml` file. The `changefreq` element is optional in `sitemap.xml`. If you leave this parameter set to its default empty value, then the `changefreq` element is not added in `sitemap.xml`. Allowed values: <empty string> (default), `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, `never`.

`webhelp.sitemap.priority`

The value of the `priority` element in the generated `sitemap.xml` file. It can be set to any fractional number between 0.0 (least important priority) and 1.0 (most important priority), such as: 0.3, 0.5, 0.8, etc. The `priority` element is optional in `sitemap.xml`. If you leave this parameter set to its default empty value, then the `priority` element is not added in `sitemap.xml`.

`webhelp.copyright`

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

`webhelp.footer.file`

Path to an XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, Google Analytics, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
    <div id="fb-root"/>
    <script>
            <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
 return;
             js = d.createElement(s); js.id = id; js.src =
"//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
             fjs.parentNode.insertBefore(js, fjs); }(document, 'script', 'facebook-jssdk')); -->
    </script>
   <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

`webhelp.footer.include`

Specifies whether or not to include footer in each WebHelp page. Possible values: `yes`, `no`. If set to `no`, no footer is added to the WebHelp pages. If set to `yes` and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen XML Editor plugin footer is inserted in each WebHelp page.

`webhelp.logo.image` (not available for the WebHelp Mobile transformation)

Specifies a path to an image displayed as a logo in the left side of the output header.

`webhelp.logo.image.target.url` (not available for the WebHelp Mobile transformation)

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

`webhelp.search.ranking` (not available for the WebHelp Mobile transformation)

If this parameter is set to `false` then the *5-star rating mechanism* is no longer included in the search results that are displayed on the **Search** tab (default setting is `true`).

`webhelp.search.japanese.dictionary`

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Editor plugin uses for indexing Japanese content in the WebHelp pages. This indexer does not come bundled with Oxygen XML Editor plugin or the Oxygen XML WebHelp plugin. To use it, you need to download it from *http://mvnrepository.com/artifact/org.apache.lucene/lucene-analyzers-kuromoji/4.0.0* and add it in the *DITA_OT_DIR*`/plugins/com.oxygenxml.webhelp/lib` directory.

webhelp.google.search.script

Specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google. The value must be an URL.

webhelp.google.search.results

URL value that specifies the location of a well-formed XHTML file containing the Google Custom Search Engine element `gcse:searchresults-only`. You can use all supported attributes for this element. It is recommend to set the `linkTarget` attribute to `frm` for frameless (*iframe*) version of WebHelp or to `contentWin` for the frameset version of WebHelp. The default value for this attribute is `_blank` and the search results will be loaded in a new window. If this parameter is not specified, the following code will be used

```
<gcse:searchresults-only linkTarget="frm"></gcse:searchresults-only>
```

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

clean.output

Deletes all files from the output folder before the transformation is performed (only `no` and `yes` values are valid and the default value is `no`).

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

webhelp.head.script

URL value that specifies the location of a well-formed XHTML file containing the custom script that will be copied in the `<head>` section of every WebHelp page.

webhelp.body.script

URL value that specifies the location of a well-formed XHTML file containing the custom script that will be copied in the `<body>` section of every WebHelp page.

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Editor plugin)

The DITA Open Toolkit usually has problems processing references that point to locations outside of the directory of the processed DITA map. This parameter is used to specify whether or not the application should try to fix such references in a temporary files folder before the DITA Open Toolkit is invoked on the fixed references. The fix has no impact on your edited DITA content. Allowed values: `true` or `false` (default).

Support for Right-to-Left (RTL) Oriented Languages

To activate support for RTL languages in WebHelp output, edit the DITA map and set the `xml:lang` attribute on its root element (`map`). The corresponding attribute value can be set for following RTL languages:

- `ar-eg` - Arabic
- `he-il` - Hebrew
- `ur-pk` - Urdu

WebHelp Search Engine Optimization

A **DITA Map WebHelp** transformation scenario can be configured to produce a `sitemap.xml` file that is used by search engines to aid crawling and indexing mechanisms. A *sitemap* lists all pages of a WebHelp system and allows webmasters to provide additional information about each page, such as the date it was last updated, change frequency, and importance of each page in relation to other pages in your WebHelp deployment.

The structure of the `sitemap.xml` file looks like this:

```
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/topics/introduction.html</loc>
    <lastmod>2014-10-24</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.5</priority>
  </url>
  <url>
    <loc>http://www.example.com/topics/care.html#care</loc>
```

```
    <lastmod>2014-10-24</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.5</priority>
  </url>
   .  .  .
</urlset>
```

Each page has a `<url>` element structure containing additional information, such as:

- `loc` - the URL of the page. This URL must begin with the protocol (such as `http`), if required by your web server. It is constructed from the value of the `webhelp.sitemap.base.url` parameter from the transformation scenario and the relative path to the page (collected from the `href` attribute of a `topicref` element in the DITA map).

  Note: The value must have less than 2,048 characters.

- `lastmod` - the date when the page was last modified. The date format is `YYYY-MM-DD`.
- `changefreq` - indicates how frequently the page is likely to change. This value provides general information to assist search engines, but may not correlate exactly to how often they crawl the page. Valid values are: `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, and `never`. The first time the `sitemap.xml` file is generated, the value is set based upon the value of the `webhelp.sitemap.change.frequency` parameter in the DITA WebHelp transformation scenario. You can change the value in each `url` element by editing the `sitemap.xml` file.

  Note: The value `always` should be used to describe documents that change each time they are accessed. The value `never` should be used to describe archived URLs.

- `priority` - the priority of this page relative to other pages on your site. Valid values range from 0.0 to 1.0. This value does not affect how your pages are compared to pages on other sites. It only lets the search engines know which pages you deem most important for the crawlers. The first time the `sitemap.xml` file is generated, the value is set based upon the value of the `webhelp.sitemap.priority` parameter in the DITA WebHelp transformation scenario. You can change the value in each `url` element by editing the `sitemap.xml` file.

Note: `lastmod`, `changefreq`, and `priority` are optional elements.

Creating and Editing the `sitemap.xml` File

Follow these steps to produce a `sitemap.xml` file for your WebHelp system, which can then be edited to fine-tune search engine optimization:

1. **Edit** the transformation scenario you currently use for obtaining your WebHelp output. This opens the **Edit DITA Scenario** dialog box.
2. Open the **Parameters** tab and set a value for the following parameters:
   - `webhelp.sitemap.base.url` - the URL of the location where your WebHelp system is deployed

     Note: This parameter is required in order for Oxygen XML Editor plugin to generate the `sitemap.xml` file.

   - `webhelp.sitemap.change.frequency` - how frequently the WebHelp pages are likely to change (accepted values are: `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, and `never`)
   - `webhelp.sitemap.priority` - the priority of each page (value ranging from 0.0 to 1.0)
3. Run the transformation scenario.
4. Look for the `sitemap.xml` file in the transformation's output folder. Edit the file to fine-tune the parameters of each page, according to your needs.

Indexing Japanese Content in WebHelp Pages

To optimize the indexing of Japanese content in WebHelp pages generated from DITA map transformations, the Kuromoji analyzer can be used. This analyzer in not included in the Oxygen XML Editor plugin installation kit and must be downloaded and added.

To use the Kuromoji analyzer to index Japanese content in your WebHelp system, follow these steps:

1. Download the analyzer jar file from *http://mvnrepository.com/artifact/org.apache.lucene/lucene-analyzers-kuromoji/4.0.0*.
2. Place the Kuromoji analyzer jar file in the following directory: *DITA_OT_DIR*`/plugins/com.oxygenxml.webhelp/lib`.
3. For the analyzer to work properly, search terms that are entered into your WebHelp pages must be separated by spaces.

Optionally a Japanese user dictionary can be set with *the* `webhelp.search.japanese.dictionary` *parameter*.

Compiled HTML Help (CHM) Output Format

To perform a **Compiled HTML Help (CHM)** transformation Oxygen XML Editor plugin needs `Microsoft HTML Help Workshop` to be installed on your computer. Oxygen XML Editor plugin automatically detects `HTML Help Workshop` and uses it.

Note: `HTML Help Workshop` might fail if the files used for transformation contain accents in their names, due to different encodings used when writing the *.hhp* and *.hhc* files. If the transformation fails to produce the CHM output but the *.hhp* (HTML Help Project) file is already generated, you can manually try to build the CHM output using `HTML Help Workshop`.

Changing the Output Encoding

Oxygen XML Editor plugin uses the `htmlhelp.locale` parameter to correctly display specific characters of different languages in the output of the **Compiled HTML Help (CHM)** transformation. The **Compiled HTML Help (CHM)** default scenario that comes bundled with Oxygen XML Editor plugin has the `htmlhelp.locale` parameter set to `en-US`.

The default value of the `htmlhelp.locale` is `en-US`. To customize this parameter, go to **Configure Transformation Scenarios** and click the **Edit** button. In the parameter tab search for the `htmlhelp.locale` parameter and change its value to the desired language tag.

The format of the `htmlhelp.locale` parameter is `LL-CC`, where `LL` represents the language code (`en` for example) and `CC` represents the country code (`US` for example). The language codes are contained in the `ISO 639-1` standard and the country codes are contained in the `ISO 3166-1` standard. For further details about language tags, go to *http://www.rfc-editor.org/rfc/rfc5646.txt*.

Kindle Output Format

Oxygen XML Editor plugin requires KindleGento generate Kindle output from DITA maps. To install KindleGen for use by Oxygen XML Editor plugin, follow these steps:

1. Go to *www.amazon.com/kindleformat/kindlegen* and download the zip file that matches your operating system.
2. Unzip the file on your local disk.
3. Start Oxygen XML Editor plugin and open a DITA map in the **DITA Maps Manager** view.
4. In the **DITA Maps Manager** view, open the **Configure Transformation Scenario(s)** dialog box.
5. Select the **DITA Map Kindle** transformation and press the **Edit** button to edit it.
6. Go to **Parameters** tab and set the **kindlegen.executable** parameter as the path to the *KindleGen* directory.
7. Accept the changes.

Migrating OOXML Documents to DITA

Oxygen XML Editor plugin integrates the entire DITA for Publishers plugins suite, enabling you to migrate content from Open Office XML documents to DITA:

- Open an OOXML document in Oxygen XML Editor plugin. The document is opened in the **Archive Browser** view.
- From the **Archive Browser**, open `document.xml`.

Note: `document.xml` holds the content of the document.

- Click **Configure Transformation Scenario(s)** on the toolbar and apply the **DOCX DITA** scenario. If you encounter any issues with the transformation, click the link below for further details about the Word to DITA Transformation Framework.

DITA Map Templates

The default templates available for DITA maps are stored in `[OXYGEN_DIR]/frameworks/dita/templates/map` folder.

Here are some of the DITA map templates available when creating *new documents from templates*:

- **DITA Map - Bookmap** - New DITA Bookmap.
- **DITA Map - Map** - New DITA map.
- **DITA Map - Learning Map** - New DITA learning and training content specialization map.
- **DITA Map - Learning Bookmap** - New DITA learning and training content specialization bookmap.
- **DITA Map - Eclipse Map** - IBM specialization of DITA map used for producing Eclipse Help plugins.

DITA for Publishers Map specialization templates:

- **D4P Map** - New DITA for Publishers map.
- **D4P Pub-component-map** - New DITA for Publishers pub-component-map.
- **D4P Pubmap** - New DITA for Publishers pubmap.

The XHTML Document Type

The Extensible HyperText Markup Language (XHTML), is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax.

A file is considered to be a XHTML document when the root element name is `html`.

Default schemas that are used if one is not detected in the XHTML file are stored in the following locations:

- XHTML 1.0 - `[OXYGEN_DIR]/frameworks/xhtml/dtd/` or `[OXYGEN_DIR]/frameworks/xhtml/nvdl/`.
- XHTML 1.1 - `[OXYGEN_DIR]/frameworks/xhtml11/dtd/` or `[OXYGEN_DIR]/frameworks/xhtml11/schema/`.
- XHTML 5 - `[OXYGEN_DIR]/frameworks/xhtml/xhtml5 (epub3)/`.

The 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 default CSS files used for rendering XHTML content in **Author** mode are stored in `[OXYGEN_DIR]/frameworks/xhtml/css/`.

The default catalogs for the XHTML document type are as follows:

- `[OXYGEN_DIR]/frameworks/xhtml/dtd/xhtmlcatalog.xml`
- `[OXYGEN_DIR]/frameworks/relaxng/catalog.xml`
- `[OXYGEN_DIR]/frameworks/nvdl/catalog.xml`
- `[OXYGEN_DIR]/frameworks/xhtml11/dtd/xhtmlcatalog.xml`
- `[OXYGEN_DIR]/frameworks/xhtml11/schema/xhtmlcatalog.xml`
- `[OXYGEN_DIR]/xhtml5 (epub3)/catalog-compat.xml`

XHTML Author Mode Actions

A variety of actions are available in the XHTML framework that can be added to the **XHTML** menu, the **Author Custom Actions** toolbar, the contextual menu, and the **Content Completion Assistant**.

XHTML Toolbar Actions

The following default actions are readily available on the **XHTML (Author Custom Actions)** toolbar when editing in **Author** mode (by default, they are also available in the **XHTML** menu and some of them are in various submenus of the contextual menu):

Bold

Changes the style of the selected text to `bold` by surrounding it with `b` tag. You can use this action on multiple non-contiguous selections.

Italic

Changes the style of the selected text to `italic` by surrounding it with `i` tag. You can use this action on multiple non-contiguous selections.

Underline

Changes the style of the selected text to `underline` by surrounding it with `u` tag. You can use this action on multiple non-contiguous selections.

Link

Inserts an `a` element with an `href` attribute at the cursor position. You can type the URL of the reference you want to insert or use the **Browse** drop-down menu to select it using one of the following options:

- **Browse for local file** - Displays the **Open** dialog box to select a local file.
- **Browse for remote file** - Displays the **Open URL** dialog box to select a remote file.
- **Browse for archived file** - Opens the **Archive Browser** to select a file from an archive.
- **Browse Data Source Explorer** - Open the **Data Source Explorer** to select a file from a connected data source.
- **Search for file** - Opens the **Find Resource** dialog box to search for a file.

Insert Image

Inserts a graphic object at the cursor 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.

Headings Drop-down Menu

A drop-down menu that includes actions for inserting `h1`, `h2`, `h3`, `h4`, `h5`, `h6` elements.

Insert Paragraph

Insert a new paragraph element at current cursor position.

Insert Equation

Opens the **XML Fragment Editor** that allows you to insert and *edit MathML notations*.

Insert List Item

Inserts a list item in the current list type.

Insert Unordered List

Inserts an unordered list at the cursor position. A child list item is also automatically inserted by default.

Insert Ordered List

Inserts an ordered list at the cursor position. A child list item is also automatically inserted by default.

Insert a definition list at the cursor position

Inserts a definition list (`dl` element) with one list item (a `dt` child element and a `dd` child element).

Sort

Sorts cells or list items in a table.

Insert Table

Opens a dialog box 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 Below

Inserts a new table row with empty cells below the current row. This action is available when the cursor is positioned inside a table.

Insert Row Above

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Column After

Inserts a new table column with empty cells after the current column. This action is available when the cursor is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the cursor is positioned between two cells, Oxygen XML Editor plugin a new cell at cursor position. If the cursor is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at cursor position.

Delete Row

Deletes the table row located at cursor position.

Table Join/Split Drop-Down Menu

The following join and split actions are available from this menu:

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 cursor is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current cursor 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 cursor 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 Editor plugin 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 cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

XHTML Menu Actions

In addition, the following default actions are available in the **XHTML** menu when editing in **Author** mode (some of them are also available in the contextual menu):

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu:

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

Insert Column Before

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

Refresh References

You can use this action to manually trigger a refresh and update of all referenced resources.

Tags display mode Submenu

Full Tags with Attributes - Displays full tag names with attributes for both block level and in-line level elements.

Full Tags - Displays full tag names without attributes for both block level and in-line level elements.

Block Tags - Displays full tag names for block level elements and simple tags without names for in-line level elements.

Inline Tags - Displays full tag names 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 and is as close as possible to a word-processor view.

Profiling/Conditional Text Submenu

Edit Profiling Attributes - Allows you to configure the *profiling attributes* and their values.

Show Profiling Colors and Styles - Select this option to turn on conditional styling.

Show Profiling Attributes - Select 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 - When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

List of all profiling condition sets that match the current document type - You can click a listed condition set to activate it.

Profiling Settings - Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

XHTML Drag/Drop Actions

Dragging a file from *the **Project** view* or ***DITA Maps Manager** view* and dropping it into an XHTML document that is edited in **Author** mode creates a link to the dragged file (the `a` element with the `href` attribute) 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 an XHTML document inserts an image element (the `img` element with the `src` attribute) at the drop location, similar to the **Insert Image** 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 you to create a custom TEI P5 schema in a literate programming fashion. A system of XSLT stylesheets called *Roma* was created by the TEI Consortium for manipulating the ODD files.

A file is considered to be a TEI ODD document when the following conditions are true:

- The file extension is `.odd`.
- The document namespace is *http://www.tei-c.org/ns/1.0*.

The default schema that is used if one is not detected in the TEI ODD document is `tei_odds.rng` and it is stored in `[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/`.

The default CSS files used for rendering TEI ODD content in **Author** mode are stored in `[OXYGEN_DIR]/frameworks/tei/xml/tei/css/`.

The default catalogs for the TEI ODD document type are as follows:

- `[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 Mode Actions

A variety of actions are available in the TEI ODD framework that can be added to the **TEI ODD** menu, the **Author Custom Actions** toolbar, the contextual menu, and the **Content Completion Assistant**.

TEI ODD Toolbar Actions

The following default actions are readily available on the **TEI ODD (Author Custom Actions)** toolbar when editing in **Author** mode (by default, they are also available in the **TEI ODD** menu and some of them are in various submenus of the contextual menu):

B **Bold**

Changes the style of the selected text to `bold` by surrounding it with `hi` tag and setting the `rend` attribute to `bold`. You can use this action on multiple non-contiguous selections.

Italic

Changes the style of the selected text to `italic` by surrounding it with `hi` tag and setting the `rend` attribute to `italic`. You can use this action on multiple non-contiguous selections.

Underline

Changes the style of the selected text to `underline` by surrounding it with `hi` tag and setting the `rend` attribute to `ul`. You can use this action on multiple non-contiguous selections.

Insert Section

Inserts a new section or subsection, depending on the current context. For example, if the current context is `div1`, then a `div2` is inserted. By default, this action also inserts a paragraph element as a child node.

Insert Paragraph

Insert a new paragraph element at current cursor position.

Insert Image

Inserts an image reference at the cursor position. Depending on the current location, an image-type element is inserted.

Insert List Item

Inserts a list item in the current list type.

Insert Ordered List

Inserts an ordered list at the cursor position. A child list item is also automatically inserted by default.

Insert Itemized List

Inserts an itemized list at the cursor position. A child list item is also automatically inserted by default.

Insert Table

Opens a dialog box 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 Below

Inserts a new table row with empty cells below the current row. This action is available when the cursor is positioned inside a table.

Insert Column After

Inserts a new table column with empty cells after the current column. This action is available when the cursor is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the cursor is positioned between two cells, Oxygen XML Editor plugin a new cell at cursor position. If the cursor is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at cursor position.

Delete Row

Deletes the table row located at cursor position.

Table Join/Split Drop-Down Menu

The following join and split actions are available from this menu:

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 cursor is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current cursor 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 cursor 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 Editor plugin 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 cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

TEI ODD Menu Actions

In addition, the following default actions are available in the **TEI ODD** menu when editing in **Author** mode (some of them are also available in the contextual menu):

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu:

Insert Row Above

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

Insert Column Before

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

ID Options

Opens the **ID Options** dialog box that allows you to configure options for generating IDs in **Author** mode. The dialog box includes the following:

ID Pattern

The pattern for the ID values that will be generated. This text field can be customized using constant strings and most of the supported *Editor Variables* on page 594.

Element name or class value to generate ID for

The elements for which ID values will be generated, specified using class attribute values. To customize the list, use the **Add**, **Edit**, or **Remove** buttons.

Auto generate IDs for elements

If enabled, Oxygen XML Editor plugin will automatically generate unique IDs for the elements listed in this dialog box when they are created in **Author** mode.

Remove IDs when copying content in the same document

Allows you to control whether or not pasted elements that are listed in this dialog box should retain their existing IDs when copying content in the same document. To retain the element IDs, disable this option.

Generate IDs

Oxygen XML Editor plugin generates unique IDs for the current element (or elements), depending on how the action is invoked:

- When invoked on a single selection, an ID is generated for the selected element at the cursor position.
- When invoked on a block of selected content, IDs are generated for all top-level elements and elements from the list in the **ID Options** dialog box that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

Refresh References

You can use this action to manually trigger a refresh and update of all referenced resources.

Tags display mode Submenu

 **Full Tags with Attributes** - Displays full tag names with attributes for both block level and in-line level elements.

 **Full Tags** - Displays full tag names without attributes for both block level and in-line level elements.

 **Block Tags** - Displays full tag names for block level elements and simple tags without names for in-line level elements.

 **Inline Tags** - Displays full tag names 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 and is as close as possible to a word-processor view.

Profiling/Conditional Text Submenu

Edit Profiling Attributes - Allows you to configure the *profiling attributes* and their values.

Show Profiling Colors and Styles - Select this option to turn on conditional styling.

Show Profiling Attributes - Select 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 - When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

List of all profiling condition sets that match the current document type - You can click a listed condition set to activate it.

 **Profiling Settings** - Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

TEI ODD Drag/Drop Actions

Dragging a file from *the **Project** view* or ***DITA Maps Manager** view* and dropping it into a TEI ODD document that is edited in **Author** mode, creates 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 one of the following conditions are true:

- The local name of the root is `TEI.2`.
- The public id of the document is *-//TEI P4*.

The default schema that is used if one is not detected in the TEI P4 document is `tei2.dtd` and it is stored in `[OXYGEN_DIR]/frameworks/tei/xml/teip4/schema/dtd/`.

The default CSS files used for rendering TEI P4 content in **Author** mode is stored in `[OXYGEN_DIR]/frameworks/tei/xml/tei/css/`.

The default catalogs for the TEI P4 document type are as follows:

- `[OXYGEN_DIR]/frameworks/tei/xml/teip4/schema/dtd/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/teip4/custom/schema/dtd/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/teip4/stylesheet/catalog.xml`

To watch our video demonstration about TEI editing, go to *http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html*.

TEI P4 Author Mode Actions

A variety of actions are available in the TEI P4 framework that can be added to the **TEI P4** menu, the **Author Custom Actions** toolbar, the contextual menu, and the **Content Completion Assistant**.

TEI P4 Toolbar Actions

The following default actions are readily available on the **TEI P4 (Author Custom Actions)** toolbar when editing in **Author** mode (by default, they are also available in the **TEI P4** menu and some of them are in various submenus of the contextual menu):

B **Bold**

Changes the style of the selected text to `bold` by surrounding it with `hi` tag and setting the `rend` attribute to `bold`. You can use this action on multiple non-contiguous selections.

Italic

Changes the style of the selected text to `italic` by surrounding it with `hi` tag and setting the `rend` attribute to `italic`. You can use this action on multiple non-contiguous selections.

Underline

Changes the style of the selected text to `underline` by surrounding it with `hi` tag and setting the `rend` attribute to `ul`. You can use this action on multiple non-contiguous selections.

Insert Section

Inserts a new section or subsection, depending on the current context. For example, if the current context is `div1`, then a `div2` is inserted. By default, this action also inserts a paragraph element as a child node.

Insert Paragraph

Insert a new paragraph element at current cursor position.

Insert Image

Inserts an image reference at the cursor position. Depending on the current location, an image-type element is inserted.

Insert List Item

Inserts a list item in the current list type.

Insert Ordered List

Inserts an ordered list at the cursor position. A child list item is also automatically inserted by default.

Insert Itemized List

Inserts an itemized list at the cursor position. A child list item is also automatically inserted by default.

Sort

Sorts cells or list items in a table.

Insert Table

Opens a dialog box 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 Below

Inserts a new table row with empty cells below the current row. This action is available when the cursor is positioned inside a table.

Insert Column After

Inserts a new table column with empty cells after the current column. This action is available when the cursor is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the cursor is positioned between two cells, Oxygen XML Editor plugin a new cell at cursor position. If the cursor is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at cursor position.

Delete Row

Deletes the table row located at cursor position.

Table Join/Split Drop-Down Menu

The following join and split actions are available from this menu:

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 cursor is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current cursor 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 cursor 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 Editor plugin 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 cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

TEI P4 Menu Actions

In addition, the following default actions are available in the **TEI P4** menu when editing in **Author** mode (some of them are also available in the contextual menu):

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu:

Insert Row Above

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

Insert Column Before

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

ID Options

Opens the **ID Options** dialog box that allows you to configure options for generating IDs in **Author** mode. The dialog box includes the following:

ID Pattern

The pattern for the ID values that will be generated. This text field can be customized using constant strings and most of the supported *Editor Variables* on page 594.

Element name or class value to generate ID for

The elements for which ID values will be generated, specified using class attribute values. To customize the list, use the **Add**, **Edit**, or **Remove** buttons.

Auto generate IDs for elements

If enabled, Oxygen XML Editor plugin will automatically generate unique IDs for the elements listed in this dialog box when they are created in **Author** mode.

Remove IDs when copying content in the same document

Allows you to control whether or not pasted elements that are listed in this dialog box should retain their existing IDs when copying content in the same document. To retain the element IDs, disable this option.

Generate IDs

Oxygen XML Editor plugin generates unique IDs for the current element (or elements), depending on how the action is invoked:

- When invoked on a single selection, an ID is generated for the selected element at the cursor position.
- When invoked on a block of selected content, IDs are generated for all top-level elements and elements from the list in the **ID Options** dialog box that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

Refresh References

You can use this action to manually trigger a refresh and update of all referenced resources.

Tags display mode Submenu

Full Tags with Attributes - Displays full tag names with attributes for both block level and in-line level elements.

Full Tags - Displays full tag names without attributes for both block level and in-line level elements.

Block Tags - Displays full tag names for block level elements and simple tags without names for in-line level elements.

Inline Tags - Displays full tag names 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 and is as close as possible to a word-processor view.

Profiling/Conditional Text Submenu

Edit Profiling Attributes - Allows you to configure the *profiling attributes* and their values.

Show Profiling Colors and Styles - Select this option to turn on conditional styling.

Show Profiling Attributes - Select 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 - When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

List of all profiling condition sets that match the current document type - You can click a listed condition set to activate it.

Profiling Settings - Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

TEI P4 Drag/Drop Actions

Dragging a file from *the **Project** view* or *__DITA Maps Manager__ view* and dropping it into a TEI P4 document that is edited in **Author** mode, creates 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://github.com/TEIC/oxygen-tei*

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 Editor plugin.

   a) Set the Oxygen XML Editor plugin `frameworks` folder to the `oxygen/frameworks` subfolder of the folder of the SVN working copy.

   *Open the **Preferences** dialog box*, go to **Global**, and set the path of the SVN working copy in the **Use custom frameworks** option.

   b) *Open the **Preferences** dialog box*, 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 Editor plugin install folder.

The TEI P5 Document Type

The **Text Encoding Initiative (TEI) Guidelines** is an international and interdisciplinary standard that enables libraries, museums, publishers, and individual scholars to represent a variety of literary and linguistic texts for online research, teaching, and preservation.

A file is considered to be a TEI P5 document when one of the following conditions are true:

- The document namespace is *http://www.tei-c.org/ns/1.0*.
- The public id of the document is *-//TEI P5*.

The default schema that is used if one is not detected in the TEI P5 document is `tei_all.rng` and it is stored in `[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/`.

The CSS file used for rendering TEI P5 content in **Author** mode is located in `[OXYGEN_DIR]/frameworks/tei/xml/tei/css/tei_oxygen.css`.

The default catalogs for the TEI P5 document type are as follows:

- `[OXYGEN_DIR]/frameworks/tei/xml/tei/schema/dtd/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/tei/custom/schema/dtd/catalog.xml`
- `[OXYGEN_DIR]/frameworks/tei/xml/tei/stylesheet/catalog.xml`

To watch our video demonstration about TEI editing, go to *http://oxygenxml.com/demo/WYSIWYG_TEI_Editing.html*.

TEI P5 Author Mode Actions

A variety of actions are available in the TEI P5 framework that can be added to the **TEI P5** menu, the **Author Custom Actions** toolbar, the contextual menu, and the **Content Completion Assistant**.

TEI P5 Toolbar Actions

The following default actions are readily available on the **TEI P5 (Author Custom Actions)** toolbar when editing in **Author** mode (by default, they are also available in the **TEI P5** menu and some of them are in various submenus of the contextual menu):

Bold

Changes the style of the selected text to `bold` by surrounding it with `hi` tag and setting the `rend` attribute to `bold`. You can use this action on multiple non-contiguous selections.

Italic

Changes the style of the selected text to `italic` by surrounding it with `hi` tag and setting the `rend` attribute to `italic`. You can use this action on multiple non-contiguous selections.

Underline

Changes the style of the selected text to `underline` by surrounding it with `hi` tag and setting the `rend` attribute to `ul`. You can use this action on multiple non-contiguous selections.

Insert Section

Inserts a new section or subsection, depending on the current context. For example, if the current context is `div1`, then a `div2` is inserted. By default, this action also inserts a paragraph element as a child node.

Insert Paragraph

Insert a new paragraph element at current cursor position.

Insert Image

Inserts an image reference at the cursor position. Depending on the current location, an image-type element is inserted.

Insert List Item

Inserts a list item in the current list type.

Insert Ordered List

Inserts an ordered list at the cursor position. A child list item is also automatically inserted by default.

Insert Itemized List

Inserts an itemized list at the cursor position. A child list item is also automatically inserted by default.

Sort

Sorts cells or list items in a table.

Insert Table

Opens a dialog box 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 Below

Inserts a new table row with empty cells below the current row. This action is available when the cursor is positioned inside a table.

Insert Column After

Inserts a new table column with empty cells after the current column. This action is available when the cursor is positioned inside a table.

Insert Cell

Inserts a new empty cell depending on the current context. If the cursor is positioned between two cells, Oxygen XML Editor plugin a new cell at cursor position. If the cursor is inside a cell, the new cell is created after the current cell.

Delete Column

Deletes the table column located at cursor position.

Delete Row

Deletes the table row located at cursor position.

Table Join/Split Drop-Down Menu

The following join and split actions are available from this menu:

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 cursor is positioned between two cells.

Join Cell Above

Joins the content of the cell from the current cursor 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 cursor 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 Editor plugin 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 cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell To The Right

Splits the cell from the current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Above

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

Split Cell Below

Splits the cell from current cursor 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 Editor plugin decreases the column span of the source cell with one.

TEI P5 Menu Actions

In addition, the following default actions are available in the **TEI P5** menu when editing in **Author** mode (some of them are also available in the contextual menu):

Table submenu

In addition to the table actions available on the toolbar, the following actions are available in this submenu:

Insert Row Above

Inserts a new table row with empty cells above the current row. This action is available when the cursor is positioned inside a table.

Insert Rows

Opens a dialog box that allows you to insert any number of rows and specify the position where they will be inserted (**Above** or **Below** the current row).

Insert Column Before

Inserts a column before the current one.

Insert Columns

Opens a dialog box that allows you to insert any number of columns and specify the position where they will be inserted (**Above** or **Below** the current column).

ID Options

Opens the **ID Options** dialog box that allows you to configure options for generating IDs in **Author** mode. The dialog box includes the following:

ID Pattern

The pattern for the ID values that will be generated. This text field can be customized using constant strings and most of the supported *Editor Variables* on page 594.

Element name or class value to generate ID for

The elements for which ID values will be generated, specified using class attribute values. To customize the list, use the **Add**, **Edit**, or **Remove** buttons.

Auto generate IDs for elements

If enabled, Oxygen XML Editor plugin will automatically generate unique IDs for the elements listed in this dialog box when they are created in **Author** mode.

Remove IDs when copying content in the same document

Allows you to control whether or not pasted elements that are listed in this dialog box should retain their existing IDs when copying content in the same document. To retain the element IDs, disable this option.

Generate IDs

Oxygen XML Editor plugin generates unique IDs for the current element (or elements), depending on how the action is invoked:

- When invoked on a single selection, an ID is generated for the selected element at the cursor position.
- When invoked on a block of selected content, IDs are generated for all top-level elements and elements from the list in the **ID Options** dialog box that are found in the current selection.

Note: The **Generate IDs** action does not overwrite existing ID values. It only affects elements that do not already have an `id` attribute.

Refresh References

You can use this action to manually trigger a refresh and update of all referenced resources.

Tags display mode Submenu

Full Tags with Attributes - Displays full tag names with attributes for both block level and in-line level elements.

Full Tags - Displays full tag names without attributes for both block level and in-line level elements.

Block Tags - Displays full tag names for block level elements and simple tags without names for in-line level elements.

Inline Tags - Displays full tag names 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 and is as close as possible to a word-processor view.

Profiling/Conditional Text Submenu

Edit Profiling Attributes - Allows you to configure the *profiling attributes* and their values.

Show Profiling Colors and Styles - Select this option to turn on conditional styling.

Show Profiling Attributes - Select 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 - When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

List of all profiling condition sets that match the current document type - You can click a listed condition set to activate it.

Profiling Settings - Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

TEI P5 Drag/Drop Actions

Dragging a file from *the **Project** view* or ***DITA Maps Manager** view* and dropping it into a TEI P5 document that is edited in **Author** mode, creates a link to the dragged file (the `ptr` element with the `target` attribute) 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 TEI P5 document inserts a graphic element (the `graphic` element with the `url` attribute) at the drop location, similar to the **Insert Image** toolbar action.

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://github.com/TEIC/oxygen-tei*

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 Editor plugin.
   a) Set the Oxygen XML Editor plugin `frameworks` folder to the `oxygen/frameworks` subfolder of the folder of the SVN working copy.

      *Open the **Preferences** dialog box*, go to **Global**, and set the path of the SVN working copy in the **Use custom frameworks** option.

   b) *Open the **Preferences** dialog box*, 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 Editor plugin 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 Editor plugin.

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 Editor plugin `frameworks` folder.

The JATS Document Type

The JATS (NISO Journal Article Tag Suite) document type is a technical standard that defines an XML format for scientific literature.

A file is considered to be a JATS document when the PUBLIC ID of the document contains the string `-//NLM//DTD`.

Default schemas that are used if one is not detected in the JATS document are stored in `[OXYGEN_DIR]/frameworks/jats/O2-DTD/`.

The default CSS files used for rendering JATS content in **Author** mode are stored in `[OXYGEN_DIR]/frameworks/jats/css/`.

The default XML catalog, `JATS-catalog-O2.xml`, is stored in `[OXYGEN_DIR]/frameworks/O2-DTD/`.

JATS Author Mode Actions

A variety of actions are available in the JATS framework that can be added to the **JATS** menu, the **Author Custom Actions** toolbar, the contextual menu, and the **Content Completion Assistant**.

JATS Toolbar Actions

The following default actions are readily available on the **JATS (Author Custom Actions)** toolbar when editing in **Author** mode (by default, they are also available in the **JATS** menu and in various submenus of the contextual menu):

Bold

Surrounds the selected text with a `bold` tag. You can use this action on multiple non-contiguous selections.

Italic

Surrounds the selected text with an `italic` tag. You can use this action on multiple non-contiguous selections.

Underline

Surrounds the selected text with an `underline` tag. You can use this action on multiple non-contiguous selections.

Insert Paragraph

Insert a new paragraph element at current cursor position.

Insert Image

Inserts an image reference at the cursor position. Depending on the current location, an image-type element is inserted.

Insert List Item

Inserts a list item in the current list type.

Insert Unordered List

Inserts an unordered list at the cursor position. A child list item is also automatically inserted by default.

Insert Ordered List

Inserts an ordered list at the cursor position. A child list item is also automatically inserted by default.

JATS Menu Actions

In addition, the following default actions are available in the **JATS** menu when editing in **Author** mode:

Refresh References

You can use this action to manually trigger a refresh and update of all referenced resources.

Tags display mode Submenu

Full Tags with Attributes - Displays full tag names with attributes for both block level and in-line level elements.

Full Tags - Displays full tag names without attributes for both block level and in-line level elements.

Block Tags - Displays full tag names for block level elements and simple tags without names for in-line level elements.

Inline Tags - Displays full tag names 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 and is as close as possible to a word-processor view.

Profiling/Conditional Text Submenu

Edit Profiling Attributes - Allows you to configure the *profiling attributes* and their values.

Show Profiling Colors and Styles - Select this option to turn on conditional styling.

Show Profiling Attributes - Select 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 - When this option is enabled, the content filtered by the currently applied condition set is greyed-out. To show only the content that matches the currently applied condition set, disable this option.

List of all profiling condition sets that match the current document type - You can click a listed condition set to activate it.

Profiling Settings - Opens the *profiling options preferences page*, where you can manage profiling attributes and profiling conditions sets. You can also configure the profiling styles and colors options from the *colors/styles preferences page* and the *attributes rendering preferences page*.

JATS Drag/Drop Actions

Dragging a file from *the **Project** view* or *__DITA Maps Manager__ view* and dropping it into a JATS document that is edited in **Author** mode, creates a link to the dragged file (the `ext-link` element with the `xlink:href` attribute) 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 JATS document inserts an image element (the `inline-graphic` element with the `xlink:href` attribute) at the drop location, similar to the **Insert Image** toolbar action.

JATS Transformation Scenarios

The following default transformation scenario is available for JATS documents:

- **JATS Preview (simple HTML)** - Converts a JATS document to a simple HTML document.

JATS Templates

Default templates are available for JATS documents. They are stored in `[OXYGEN_DIR]/frameworks/jats/templates` folder and they can be used for easily creating basic JATS documents.

The default JATS templates that are available when creating *new documents from templates* are as follows:

- **Archiving** - JATS archiving tag set version 1.0.
- **Authoring** - JATS authoring tag set version 1.0.
- **Book** - JATS book tag set version 1.0.
- **Publishing** - JATS publishing tag set version 1.0.

The EPUB Document Type

Three distinct frameworks are supported for the EPUB document type:

- **NCX** - A declarative global navigation definition.
- **OCF** - The Open Container Format (OCF) defines a mechanism by which all components of an Open Publication Structure (OPS) can be combined into a single file system entity.
- **OPF** - The Open Packaging Format (OPF) defines the mechanism by which all components of a published work that conforms to the Open Publication Structure (OPS) standard (including metadata, reading order, and navigational information) are packaged in an OPS Publication.

Note: Oxygen XML Editor plugin supports both OPF 2.0 and OPF 3.0.

Document Templates

The default templates for the *NCX* and *OCF* document types are located in the `[OXYGEN_DIR]/frameworks/docbook/templates` folder.

The default template for the *OPF 2.0* document type is located in the `[OXYGEN_DIR]/frameworks/docbook/templates/2.0` folder.

The default template for the *OPF 3.0* document type is located in the `[OXYGEN_DIR]/frameworks/docbook/templates/3.0` folder.

The following EPUB templates are available when creating *new documents from templates*:

- **NCX - Toc** - New table of contents.
- **OCF - Container** - New container based OCF.
- **OCF - Encryption** - New encryption based OCF.
- **OCF - Signatures** - New signature based OCF.
- **OPF 2.0 - Content (2.0)** - New OPF 2.0 content.
- **OPF 3.0 - Content (3.0)** - New OPF 3.0 content.

The DocBook Targetset Document Type

DocBook *Targetset* documents are used to resolve cross references with the DocBook `olink`.

A file is considered to be a *Targetset* when the root name is `targetset`.

The default schema, `targetdatabase.dtd`, for this type of document is stored in `[OXYGEN_DIR]/frameworks/docbook/xsl/common/`.

Document Templates

The default template for DocBook *Targetset* documents is located in the `[OXYGEN_DIR]/frameworks/docbook/templates/Targetset` folder.

The following DocBook *Targetset* template is available when creating *new documents from templates*:

- **DocBook Targetset - Map** - New Targetset map.

Chapter 9

Author Mode Customization

Topics:

This section contains an *Author Mode Customization Guide* on page 552, *CSS Support in Author Mode* on page 636, a collection of *Frequently Asked Questions regarding the Oxygen XML Editor plugin API*, and other topics in regards to customizing the **Author** mode.

Author Mode Customization Guide

The **Author** mode editor of Oxygen XML Editor plugin was designed to provide a friendly user interface for 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.

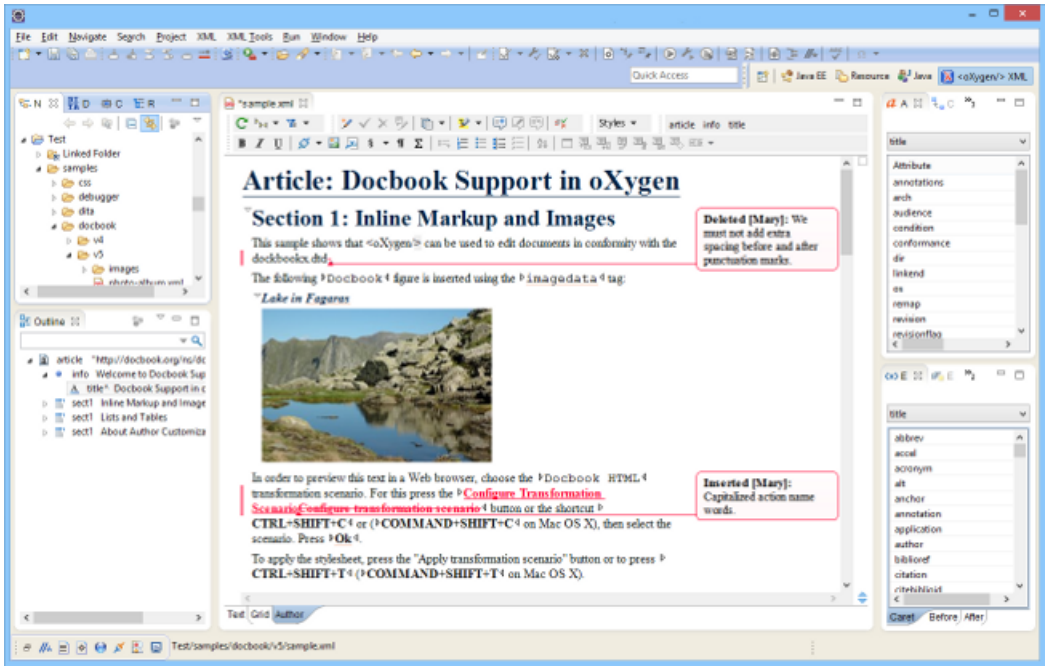


Figure 270: Oxygen XML Editor plugin Author Visual Editor

Although Oxygen XML Editor plugin comes with already configured frameworks for DocBook, DITA, TEI, and XHTML you might need to create a customization of the editor to handle other types of documents. A 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 working with XML files.

There are several ways to customize the editor:

1. Create a CSS file defining styles for the XML elements you will work with, and create XML files that reference the CSS through an `xml-stylesheet` processing instruction.
2. Fully configure a document type association. This involves putting together the CSS stylesheets, XML schemas, actions, menus, bundling them, and distributing an archive. The CSS and GUI elements are settings for the Oxygen XML Editor plugin **Author** mode. The other settings such as the templates, catalogs, and transformation scenarios are general settings and are enabled whenever the association is active, regardless of the editing mode (**Text**, **Grid**, or **Author**).

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

In order to provide as-you-type validation and to compute valid insertion proposals, Oxygen XML Editor plugin needs an XML grammar (XML Schema, DTD, or RelaxNG) associated to the XML. The grammar specifies how the internal structure of the XML is defined.

Consider a use-case in which several users are testing a system and must send report results to a content management system. The customization should provide a visual editor for these kind of documents. The following XML Schema,

`test_report.xsd` defines 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>
```

CSS Stylesheet

A set of rules must be defined for describing how the XML document is to be rendered in **Author** mode. This is done using Cascading Style Sheets (CSS). 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. However, blocks that contains in-line boxes arrange its children in a horizontal flow. That is why the paragraph lines are also blocks, while the traditional "bold" and "italic" sections are represented as in-line 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 a *table-row* style. Similarly, the `row` elements must contain elements with a *table-cell* style.

To make it easy to understand, the following section describes how each element from a schema is formatted using a CSS file. Note that this is just one of infinite possibilities for formatting the content.

`report`

This element is the root element of a 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 large font. The **block** display is used so that the subsequent elements will be placed below it, and its font is changed 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 has the **block** display. Also, the background color is changed to make it standout.

```
description {
    display:block;
    background-color:#EEEEFF;
    color:black;
}
```

`line`

A line of text in the description. A specific aspect is not defined and it just indicates that the display should be **block** style.

```
line {
    display:block;
}
```

`important`

The `important` element defines important text from the description. Since it can be mixed with text, its display property must be set to **inline**. Also, the text is emphasized with **bold**to make it easier to spot.

```
important {
    display:inline;
    font-weight:bold;
}
```

`results`

The `results` element displays the list of *test_names* and the results 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 the 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;
}
```

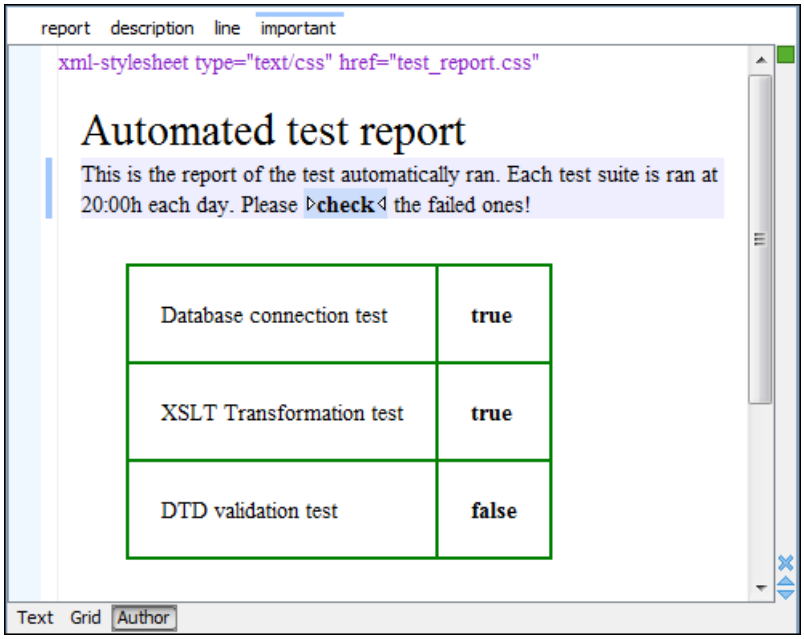


Figure 271: Report Rendered in Author Mode

Note: You can edit attributes in-place in the **Author** mode using *form-based controls*.

Associating a Stylesheet 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, such as namespaces and custom extensions, of the CSS specification are also supported. Oxygen XML Editor plugin also supports stylesheets coded with the LESS dynamic stylesheet language.

There are several methods for associating a stylesheet (CSS or LESS) with an XML document:

1. Insert the `xml-stylesheet` processing instruction with the `type` attribute at the beginning of the XML document. If you do not want to alter your XML documents, *you should create a new document type (framework)*.

   CSS example:

   ```
   <?xml-stylesheet type="text/css" href="test.css"?>
   ```

   LESS example:

   ```
   <?xml-stylesheet type="text/css" href="test.less"?>
   ```

   Note: XHTML documents need a `link` element, with the `href` and `type` attributes in the `head` child element, as specified in the *W3C CSS specification*. XHTML example:

   ```
   <link href="/style/screen.css" rel="stylesheet" type="text/css"/>
   ```

   Tip: You can also insert the `xml-stylesheet` processing instruction by using the **Associate XSLT/CSS Stylesheet** action that is available on the toolbar or in the **XML** menu.

2. Configure a *Document Type Association* by adding a new CSS or LESS file in the settings. To do so, *open the **Preferences** dialog box* and go to **Document Type Association**. Edit the appropriate framework, open the **Author** tab, then the **CSS** tab. Press the **New** button to add a new CSS or LESS file.

   Note: The Document Type Associations are read-only, so you need to extend an existing one.

You can read more about associating a CSS to a document in the section about *customizing the CSS of a document type*.

If a document has no CSS association or the referenced 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.

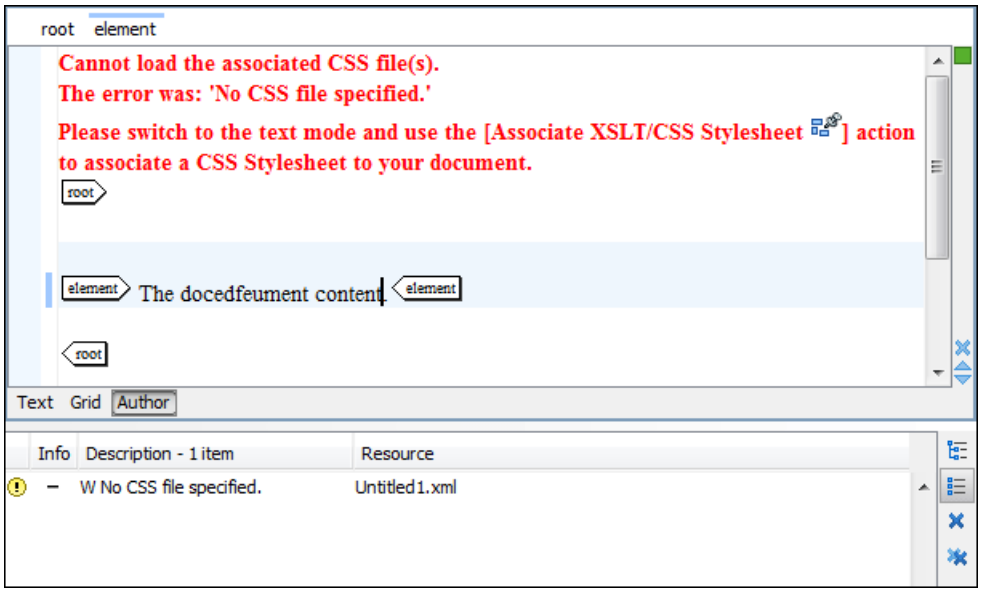


Figure 272: Document with no CSS association default rendering

The XML Instance Template

Based on the XML Schema and CSS file Oxygen XML Editor plugin can help the content author in loading, editing, and validating the test reports. An XML file template must be created, which is 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 reference 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 Editor plugin 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 Editor plugin website*. Also it is available in the *Oxygen SDK Maven Project*.

Adding or Editing a Document Type Association (Framework)

To add or edit a *Document Type Association*, *open the **Preferences** dialog box* and go to **Document Type Association**. From this ***Document Type Association** preferences page* you can use the **New**, **Edit**, **Duplicate**, or **Extend** buttons to open a ***Document Type** configuration dialog box* that allows you to customize a new or existing document type (framework).

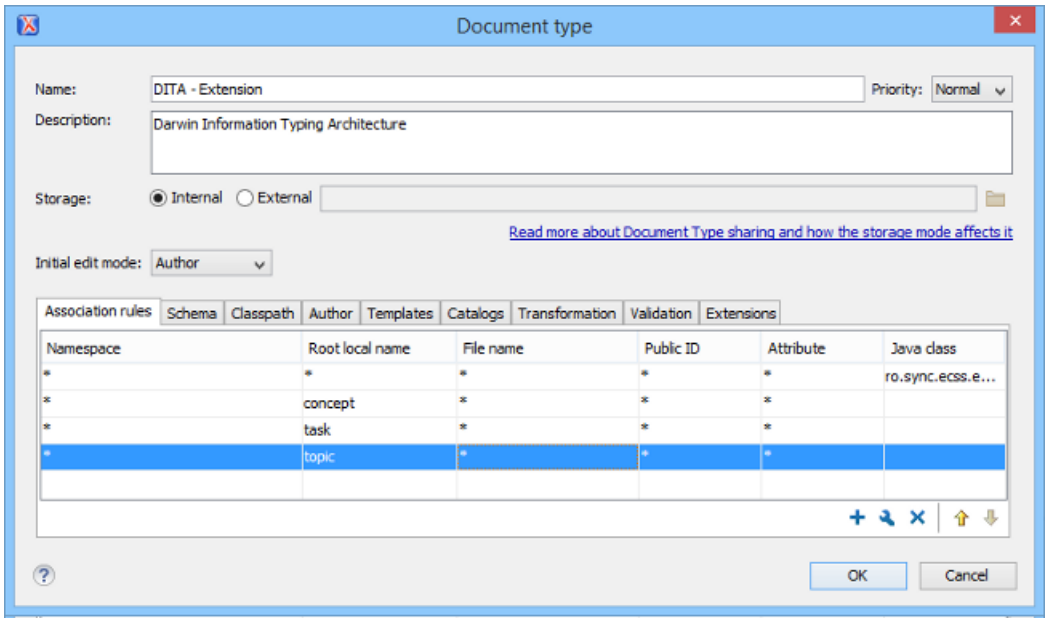


Figure 273: The Document Type Configuration Dialog Box

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 the following: 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: Frameworks that have the same priority are sorted alphabetically.

- **Description** - The document type description displayed as a tool tip in the ***Document Type Association*** *preferences page*.
- **Storage** - The location where the document type is saved. If you select the **External** storage option, 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 Editor plugin project file (for Project-level Document Type Association options) or in the Oxygen XML Editor plugin internal options (for Global-level Document Type Association Options). You can change the Document Type Association options level in the ***Document Type Association*** *preferences page*.
- **Initial edit mode** - Allows you to select the initial editing mode for this document type: **Editor specific**, **Text**, **Author, Grid** and **Design** (available only for the W3C XML Schema editor). If the **Editor specific** option is selected, the initial editing mode is determined based upon the editor type. You can find the mapping between editors and edit modes in the ***Edit modes*** *preferences page.* You can impose an initial mode for opening files that 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 box* 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 the Java class that is called to determine if the document type should be used for an XML document. Java class must either implement the `ro.sync.ecss.extensions.api.DocumentTypeCustomRuleMatcher` interface or extend the `ro.sync.ecss.extensions.api.DocumentTypeAdvancedCustomRuleMatcher` abstract class from the *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

Configure Actions, Menus, and Toolbars for a Framework

You can configure actions, menus, and toolbars that are specific to a document type in the **Author** mode to gain a productive editing experience, by using the ***Document Type*** *configuration dialog box*.

To add or configure actions, menus, or toolbars follow this procedure:

1. *Open the* ***Preferences*** *dialog box*, 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*** *configuration dialog box* go to the **Author** tab, then go to **Actions**.

3. Click the **New** button and use the ***Action*** *dialog box* to create an action.

Configure the Insert Section Action for a Framework

This section presents all the steps that you need to follow, to define the **Insert Section** action. It is assumed that the icon files, § (`Section16.gif`) for the menu item and § (`Section20.gif`) for the toolbar, are already available. Although you could use the same icon size for both the 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.

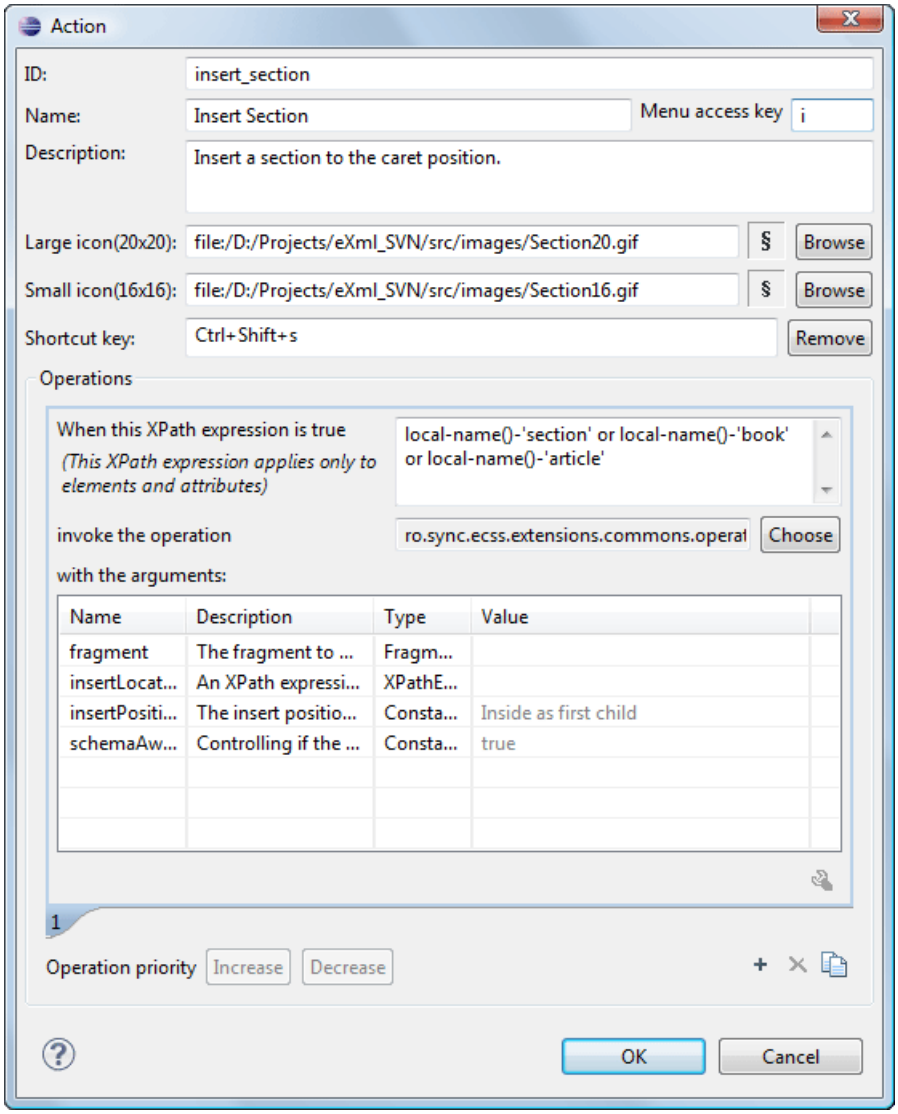


Figure 274: The Action Dialog Box

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 cursor position**.

5. Set the **Large icon (20x20)** field to `${frameworks}/sdf/Section20.gif`. 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 reference the images not by the file name, but by their relative path location in the class-path.

   If the image file `Section20.gif` is located in the **images** directory inside the jar archive, you can reference it by using **/images/Section20.gif**. The jar file must be added into the **Classpath** list.

6. Set the **Small icon (16x16)** field to `${frameworks}/sdf/Section16.gif`.
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. The first action mode enabled by the evaluation of its associated XPath expression will be executed when the action is triggered by the user. The Xpath expression needs to be version 2.0 and its scope must be only element and attribute nodes of the edited document. Otherwise, the expression will not return a match and will not trigger the action. If the expression is left empty, the action will be enabled anywhere in the scope of the root element. 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 cursor 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 cursor position.

   `insertPosition` - select "**Inside**".

Configure the Insert Table Action for a Framework

The procedure described below will create an action that inserts a table with three rows and three columns into a document. The first row is the table header. As with *the insert section action*, you will use the `InsertFragmentOperation` built-in operation.

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 (Meta 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 cursor 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**.

Configure the Main Menu for a Framework

Defined actions can be grouped into customized menus in the Oxygen XML Editor plugin menu bar.

1. Open the ***Document Type*** *configuration dialog box* for the ***SDF*** *framework* and go to the **Author** tab.
2. Go to the **Menu** subtab. 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 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.

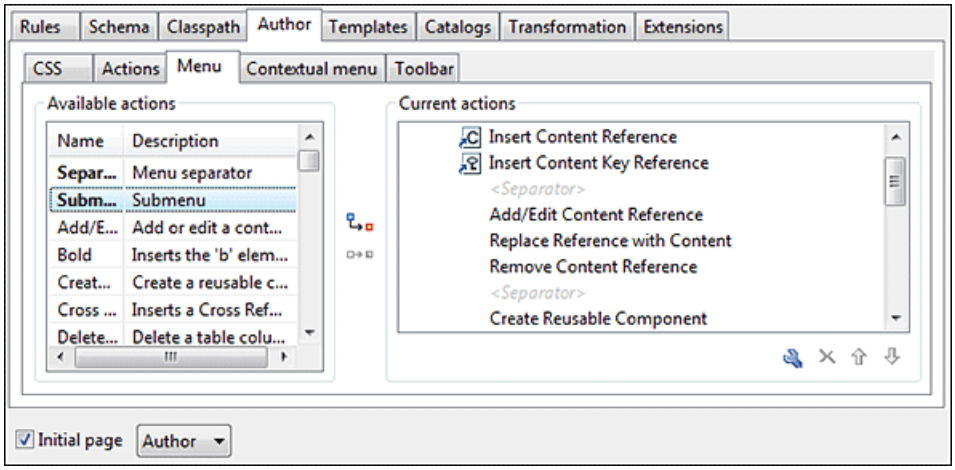


Figure 275: Configuring the Menu

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** mode actions (common to all document types) and the two actions created previously (with **Insert table** under the **Table** submenu).

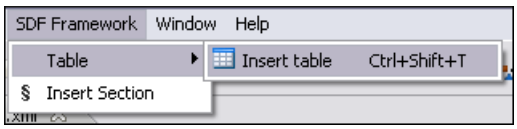


Figure 276: Author Mode Menu

Configure the Contextual Menu for a Framework

The contextual menu is displayed when you right-click in the **Author** editing area. You can only configure the bottom part of the menu, since the top part is reserved for a list of generic actions (such as Copy, Paste, Undo, etc.)

1. Open the ***Document Type*** *configuration dialog box* for the particular framework and go to the **Author** tab. Next, go to the **Contextual Menu** subtab.
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.

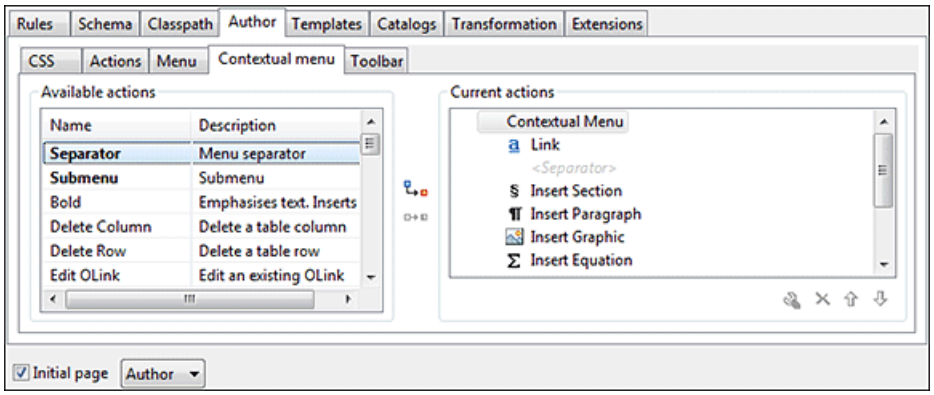


Figure 277: Configuring the Contextual Menu

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.

Configure the Toolbars for a Framework

The procedure below describes how to add defined actions to a toolbar. These steps use examples from the two previous help topics that described how to define the ***Insert Section*** and ***Insert Table*** actions. You can also configure additional toolbars to add other custom actions.

1. Open the ***Document Type*** *configuration dialog box* for the ***SDF*** *framework* and select the **Author** tab. Next, go to the **Toolbar** subtab.

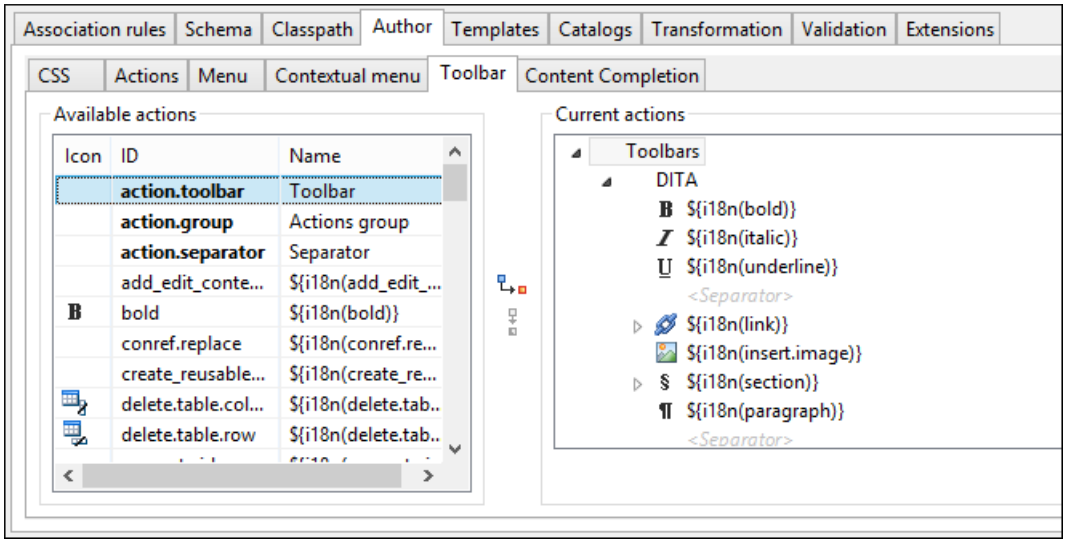


Figure 278: Configuring the Toolbar

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 279: 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. If your toolbar is not displayed when switching to the **Author** mode, right click the main toolbar, select **Configure Toolbars**, and make sure the **Author Custom Actions 1** toolbar is enabled.

Configure Content Completion for a Framework

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** configuration dialog box* for the ***SDF** framework* and select the **Author** tab. Next, go to the **Content Completion** tab.

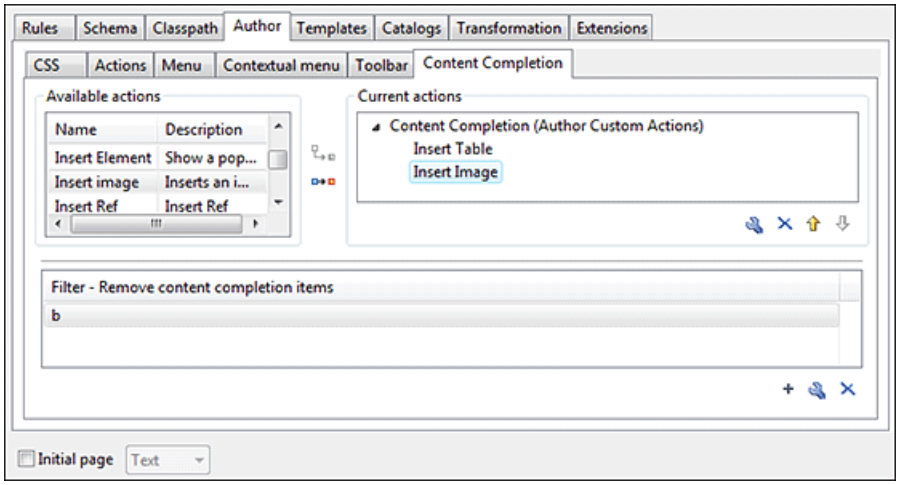


Figure 280: Customize Content Completion

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. An **Insert Action** dialog box appears, giving you the possibility to select where to provide the selected action.

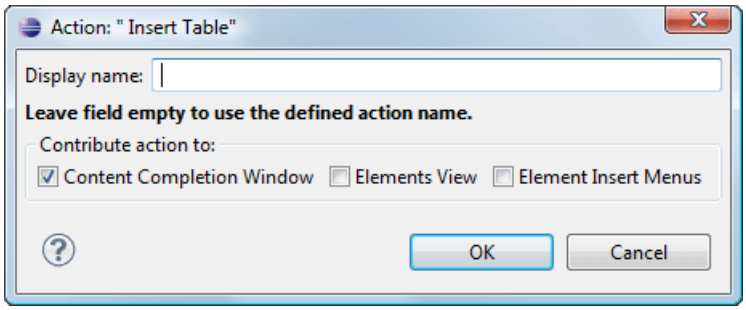


Figure 281: Insert Action Dialog Box

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 **Remove item** dialog box is displayed, allowing you to input the item name and to choose the controls that filter it. The **Item name** combo box accepts wildcards.

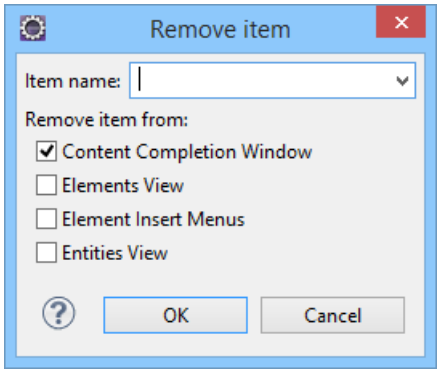


Figure 282: Remove Item Dialog Box

Author Mode Default Operations

The default operations for the **Author** mode, along with their arguments are as follows:

- **`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 namespace declarations of the inserted fragment will be adapted 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 575.

- **`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 cursor position. For more details about the list of parameters go to *The Arguments of SurroundWithFragmentOperation* on page 577.

- **`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 cursor 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 the cursor offset. The argument of this operation is:

  - **`fragment`** - The XML fragment containing the MathML content which should be inserted.

- **`OpenInSystemAppOperation`**

  Opens a resource in the system application that is associated with the resource in the operating system. The arguments of this operation is:

  - **`resourcePath`** - An XPath expression that, when executed, returns the path of the resource to be opened. Editor variables are expanded in the value of this parameter, before the expression is executed.
  - **`isUnparsedEntity`** - Possible values are `true` or `false`. If the value is `true`, the value of the **`resourcePath`** argument is treated as the name of an unparsed entity.

- **`ChangeAttributeOperation`**

  This operation allows you to add/modify/remove an attribute. You can use this operation in your own custom **Author** mode 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-place 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 the cursor position is unwrapped.

- **`ToggleSurroundWithElementOperation`**

  This operation allows wrapping and unwrapping content in a specific wrapper element which can have certain attributes specified on it. It is useful to implement toggle actions such as highlighting text as bold, italic, or underline. The operation supports processing multiple selection intervals, such as multiple cells within a table column selection. 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 whether or not the insertion is valid, based upon the schema. If the insertion is not valid, then wrapping action will be broken up into smaller intervals until the wrapping action is valid. For example, if you try to wrap a *paragraph* element with a *bold* element, it would not be valid, so the operation will wrap the text inside the paragraph instead, since it would be valid at that position.

- **`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 cursor 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.

  Notice: For the WebApp Component, these operations cannot be invoked using the JavaScript API.

  These operations have 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 cursor 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 cursor location.

- **`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 cursor position, Before, After, Inside as first child or Inside as last child.

- **`caretPosition`**

  The position of the cursor after the action is executed. It can be: `Preserve`, `Before`, `Start`, `First editable position`, `End`, or `After`. If this parameter is not set, you can still indicate the position of the cursor by using the `${caret}` editor variable in the inserted content.

- **`expandEditorVariables`**

  Parameter controlling the expansion of editor variables returned by the script processing. Expansion is enabled by default.

- **`XQueryUpdateOperation`**

  Allows you to execute an XQuery Update script directly over content in **Author** mode.

  Notice: This operation is not applicable to the Author Component or the WebApp Component.

  It has one argument:

  - **`script`**

The XQuery Update script to be executed. The value can either be an XQuery script or a URL that points to the XQuery Update script. You can use the `${framework}` or `${frameworkDir}` editor variables to refer the scripts from the framework directory.

The script will be executed in the context of the node at the cursor position. If the script declares the following variable, it will also receive the selected nodes (assuming that entire nodes are selected):

```
declare variable $oxyxq:selection external;
```

An example of an XQuery Update Script that converts paragraphs to list items:

```
declare namespace oxyxq = "http://www.oxygenxml.com/ns/xqu";
(: This variable will be linked to the selected nodes assuming that there are
actually fully selected nodes. For example this selection will returnnull:
<p>{SEL_START}text{SEL_END} in para</p>
but this will give two "p" elements:
{SEL_END}<p>text</p><p>text2</p>{SEL_END}

If a multiple selection exists it will also be processed and forwarded. Again, only fully selected nodes will
 be passed.
:)
declare variable $oxyxq:selection external;

(: We will process either the selection or the context node :)
let $toProcess := if (empty($oxyxq:selection)) then
    (.)
else
    ($oxyxq:selection)

returnif (not(empty($toProcess))) then
        (
        (: Create the list :)
        let $ul :=
        <ul>
            {
                for $sel in $toProcess
                return
                    <li>{$sel}</li>
            }
        </ul>

        return
            (
            (: Delete the processed nodes :)
            for $sel in $toProcess
            return
                delete node $sel,
            (: Inserts the constructed list :)
            insert node $ul
                before $toProcess[1]
            )
        )
    else
        ()
```

- **`JSOperation`**

  Allows you to call the Java API from custom JavaScript content.

  Notice: For the WebApp Component, this operation cannot be invoked using the JavaScript API.

  This operation accepts the following parameter:

  - **`script`**

    The JavaScript content to execute. It must have a function called `doOperation()`, which can use the predefined `authorAccess` variable. The `authorAccess` variable has access to the entire *`ro.sync.ecss.extensions.api.AuthorAccess`* Java API.

The following example is a script that retrieves the current value of the **type** attribute on the current element, allows the end user to edit its new value and sets the new value in the document:

```
function doOperation(){
  //The current node is either entirely selected...
 currentNode = authorAccess.getEditorAccess().getFullySelectedNode();
 if(currentNode == null){
  //or the cursor is placed in it
  caretOffset = authorAccess.getEditorAccess().getCaretOffset();
  currentNode = authorAccess.getDocumentController().getNodeAtOffset(caretOffset);
 }
 //Get current value of the @type attribute
  currentTypeValue = "";
  currentTypeValueAttr = currentNode.getAttribute("type");
  if(currentTypeValueAttr != null){
    currentTypeValue = currentTypeValueAttr.getValue();
  }
 //Ask user for new value for attribute.
 newTypeValue = javax.swing.JOptionPane.showInputDialog("Input @type value", currentTypeValue);

 if(newTypeValue != null){
   //Create and set the new attribute value for the @type attribute.
   attrValue = new Packages.ro.sync.ecss.extensions.api.node.AttrValue(newTypeValue);
   authorAccess.getDocumentController().setAttribute("type", attrValue, currentNode);
 }
}
```

Note: If you have a script called `commons.js` in the framework directory, you can call functions defined inside it from your custom script content so that you can use that external script file as a library of functions.

- **`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 Editor plugin.

  - **`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 cursor 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 cursor 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 cursor position will be used.
  - **`name`** - The pseudo-class local name.

- **`ShowElementDocumentationOperation`**

  Opens the associated specification HTML page for the current element. The operation accepts as parameter an URL pattern that points to the HTML page containing the documentation.

- **`XQueryUpdateOperation`**

  An implementation of an operation that applies an XQuery Update script. The changes are performed directly over the `AuthorNode` model. The operation accepts the following parameter:

  - **`script`** - can be either an XQuery script or an URL pointing to the XQuery update script (you can use `${framework}` and `${frameworkDir}` to refer scripts from the framework directory).

  The script will be executed in the context of the cursor node. If the XQuery script declares the `selection` variable, it will also receive the selected nodes (assuming that the selection consists entirely of nodes).

  The following example makes use of the `selection` variable to convert selected paragraphs into a list:

```
declare namespace oxyxq = "http://www.oxygenxml.com/ns/xqu";
(: This variable will be linked to the selected nodes assuming that there are
actually fully selected nodes. For example this selection will return null:
<p>{SEL_START}text{SEL_END} in para</p>
but this will give two "p" elements:
{SEL_END}<p>text</p><p>text2</p>{SEL_END}

If a multiple selection exists it will also be processed and forwarded.
Again, only fully selected nodes will be passed.
:)
declare variable $oxyxq:selection external;

(: We will process either the selection or the context node :)
let $toProcess := if (empty($oxyxq:selection)) then
    (.)
else
    ($oxyxq:selection)

return
    if (not(empty($toProcess))) then
        (
        (: Create the list :)
        let $ul :=
        <ul>
            {
                for $sel in $toProcess
                return
                    <li>{$sel}</li>
            }
        </ul>

        return
            (
            (: Delete the processed nodes :)
            for $sel in $toProcess
            return
                delete node $sel,
            (: Inserts the constructed list :)
            insert node $ul
                before $toProcess[1]
            )
        )
    else
        ()
```

- **`RemovePseudoClassOperation`**

An operation that removes a pseudo-class from an element. Accepts the following parameters:

- **`name`** - Name of the pseudo-class to be removed.
- **`elementLocation`** - The XPath location that identifies the element. Leave it empty for the current element.

Let's consider that there is a pseudo-class called `myClass` on the element `paragraph` and there are CSS styles matching the pseudo-class.

```
paragraph:myClass{
  font-size:2em;
  color:red;
}
paragraph{
  color:blue;
}
```

In the previous example, by removing the pseudo-class, the layout of the `paragraph` is rebuilt by matching the other rules (in this case, the foreground color of the `paragraph` element will become blue.

- **`TogglePseudoClassOperation`**

  An implementation of an operation to toggle on/off the pseudo-class of an element. Accepts the following parameters:

  - **`name`** - Name of the pseudo-class to be toggled on/off.
  - **`elementLocation`** - The XPath location that identifies the element. Leave it empty for the current element.

```
paragraph:myClass{
  color:red;
}
paragraph{
  color:blue;
}
```

By default, the paragraph content is rendered in blue. Suppose that we have a `TogglePseudoClassOperation` configured for the `myClass` pseudo-class. Invoking it the first time will set the `myClass` pseudo-class and the paragraph will be rendered in red. Invoking the operation again, will remove the pseudo-class and the visible result will be a blue rendered `paragraph` element.

- **`ExecuteMultipleWebappCompatibleActionsOperation`**

  An implementation of an operation that runs a sequence of Oxygen XML WebApp-compatible actions, defined as a list of IDs.

- **`DeleteElementsOperation`**

  Deletes the nodes indicated by the `elementLocations` parameter XPath expression. If missing, the operation will delete the node at the cursor location.

- **`DeleteElementOperation`**

  Deletes the node indicated by the `elementLocation` parameter XPath expression. If missing, the operation will delete the node at the cursor location.

- **`InsertXIncludeOperation`**

  Insert an **`XInclude`** element at the cursor offset. Opens a dialog box that allows you to browse and select content to be included in your document and automatically generates the corresponding XInclude instruction.

Author mode operations can include parameters that contain the following editor variables:

- *${caret}* - The position where the cursor is located. This variable can be used in a code template, in **Author** mode operations, or in a selection plugin.
- *${selection}* - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** mode operations, or in a selection plugin.
- *${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}* - To prompt for values at runtime, use the *ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value'')* editor variable. You can set the following parameters:
  - `'message'` - The displayed message. Note the quotes that enclose the message.
  - `type` - Optional parameter, with one of the following values:

Parameter	
`url`	**Format:** `${ask('message', url, 'default_value')}`
	Description: Input is considered a URL. Oxygen XML Editor plugin checks that the provided URL is valid.
	Example: • `${ask('Input URL', url)}` - The displayed dialog box has the name `Input URL`. The expected input type is URL. • `${ask('Input URL', url, 'http://www.example.com')}` - The displayed dialog box has the name `Input URL`. The expected input type is URL. The input field displays the default value `http://www.example.com`.
`password`	**Format**: *${ask('message', password, 'default')}*
	Description: The input is hidden with bullet characters.
	Example: • `${ask('Input password', password)}` - The displayed dialog box has the name `'Input password'` and the input is hidden with bullet symbols. • `${ask('Input password', password, 'abcd')}` - The displayed dialog box has the name `'Input password'` and the input hidden with bullet symbols. The input field already contains the default `abcd` value.
`generic`	**Format**: *${ask('message', generic, 'default')}*
	Description: The input is considered to be generic text that requires no special handling.
	Example: • `${ask('Hello world!')}` - The dialog box has a `Hello world!` message displayed. • `${ask('Hello world!', generic, 'Hello again!')}` - The dialog box has a `Hello world!` message displayed and the value displayed in the input box is `'Hello again!'`.
`relative_url`	**Format:** `${ask('message', relative_url, 'default')}`
	Description: Input is considered a URL. Oxygen XML Editor plugin tries to make the URL relative to that of the document you are editing.  **Note:** If the `$ask` editor variable is expanded in content that is not yet saved (such as an *untitled* file, whose path cannot be determined), then Oxygen XML Editor plugin will transform it into an absolute URL.
	Example:

Parameter	
	• *${ask('File location', relative_url, 'C:/example.txt')}* - The dialog box has the name `'File location'`. The URL inserted in the input box is made relative to the current edited document location.
`combobox`	**Format:** `${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a drop-down menu. The drop-down menu is populated with the given `rendered_value` values. Choosing such a value will return its associated value (`real_value`).  **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The drop-down menu displays the three given operating systems. The associated value will be returned based upon your selection.  **Note:** In this example, the default value is indicated by the `osx` key. However, the same result could be obtained if the default value is indicated by `Mac OS X`, like in the following example: `${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'Mac OS X')}` • `${ask('Mobile OS', combobox, ('win':'Windows Mobile';'ios':'iOS';'and':'Android'), 'Android')}`
`editable_combobox`	**Format:** `${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a drop-down menu with editable elements. The drop-down menu is populated with the given `rendered_value` values. Choosing such a value will return its associated real value (`real_value`) or the value inserted when you edit a list entry.  **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The drop-down menu displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.
`radio`	**Format:** `${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`

Parameter	
	Description: Displays a dialog box that offers a series of radio buttons. Each radio button displays a `'rendered_value` and will return an associated `real_value`. **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The radio button group allows you to choose between the three operating systems. **Note:** In this example `Mac OS X` is the default selected value and if selected it would return `osx` for the output.

  - `'default-value'` - optional parameter. Provides a default value.

- *${timeStamp}* - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- *${uuid}* - Universally unique identifier, a 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 Editor plugin installation folder as file path.
- *${homeDir}* - The path (as file path) of the user home folder.
- *${pn}* - Current project name.
- *${env(VAR_NAME)}* - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the *${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 Editor plugin as it was already in the document at the cursor 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 cursor is placed at that location. It handles any in-place editors used to edit attributes. It will be ignored if the fragment specifies a cursor position using the cursor editor variable. The possible values of this action are **true** and **false**.

`schemaAware`

This argument applies only on the surround with element operation and controls whether or not the insertion is valid, based upon the schema. If the insertion is not valid, then wrapping action will be broken up into smaller intervals until the wrapping action is valid. For example, if you try to wrap a *paragraph* element with a *bold* element, it would not be valid, so the operation will wrap the text inside the paragraph instead, since it would be valid at that position.

The Arguments of `SurroundWithFragmentOperation`

The **Author** mode operation `SurroundWithFragmentOperation` has only one argument:

- fragment

  The XML fragment that will surround the selection. For example, 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>
```

  Since 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.

Add a Custom Operation to an Existing Framework

This task explains how to add a custom **Author** mode operation to an existing document type.

1. Setup a sample project by following the *instructions for installing the SDK*. 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 box*, go to **Document Type Association**, and edit the document type (you need write access to the `[OXYGEN_DIR]`) to open the ***Document Type** configuration dialog box*.
   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.

Using Retina/HiDPI Images in Author Mode

Oxygen XML Editor plugin provides support for Retina and HiDPI images through simple naming conventions. The higher resolution images are stored in the same images folder as the normal resolution images and they are identified by a scaling factor that is included in the name of the image files. For instance, images with a Retina scaling factor of 2 will include *`@2x`* in the name (for example, `myImage@2x.png`).

You can reference an image to style an element in a CSS by using the `url` function in the `content` property, as in the following example:

```
listItem:before{
   content: url('../img/myImage.png');
}
```

This would place the image that is loaded from the `myImage.png` file just before the `listItem` element. However, if you are using a Retina display (on a Mac), the icon looks a bit blurry as it automatically gets scaled, or if you are using an HiDPI display (on a Windows-based PC), the icon remains at the original size, thus it will look very small. To solve this rendering problem you need to be able to reference both a *normal* DPI image and a *high* DPI image. However, referencing both of them from the CSS is not practical, as there is no standard way of doing this.

Starting with version 17, Oxygen XML Editor plugin interprets the argument of the `url` function as key rather than a fixed URL. Therefore, when running on a system with a Retina or HiDPI display, Oxygen XML Editor plugin will first try to find the image file that corresponds to the retina scaling factor. For instance, using the previous example, Oxygen XML Editor plugin would first try to find `myImage@2x.png`. If this file is not found, it defaults back to the *normal* resolution image file (`myImage.png`).

Oxygen XML Editor plugin also supports dark color themes. This means that the background of the editor area can be of a dark color and the foreground a lighter color. On a dark background, you may find it useful to invert the colors of images. Again, this can be done with simple naming conventions. If an image designed for a dark background is not found, the *normal* image is used.

Retina/HiDPI Naming Convention

Refer to the following table for examples of the Retina/HiDPI image naming convention that is used in Oxygen XML Editor plugin:

Color Theme	Referred Image File	Double Density Image File	Triple Density Image File
normal	../img/myImage.png	../img/myImage@2x.png	../img/myImage@3x.png
dark	../img/myImage_dark.png	../img/myImage_dark@2x.png	../img/myImage_dark@3x.png

Adding Retina/HiDPI Icons in a Framework

Higher resolution icons can also be included in customized frameworks for rendering them in a Retina or HiDPI display. The icons can be referenced directly from the Document Type customization (from the ***Action** dialog box*) or from an API (*`ro.sync.exml.workspace.api.node.customizer.XMLNodeRendererCustomizer`*).

As with any image, the higher resolution icons are stored in the same images folder as the normal resolution images and they are identified by a scaling factor that is included in the name of the image files. For instance, icons with a Retina scaling factor of 2 will include `@2x` in the name (for example, `myIcon@2x.png`).

Developers should not specify the path of the alternate icons (`@2x` or `@3x`) in the ***Action** dialog box* or the *`XMLNodeRendererCustomizer` API*. When using a Retina or HiDPI display, Oxygen XML Editor plugin automatically searches the folder of the *normal* icon for a corresponding image file with a Retina scaling factor in the name. If the higher resolution icon file does not exist, the *normal* icon is scaled and used instead.

Java API - Extending Author Functionality through Java

Oxygen XML Editor plugin **Author** mode 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. The following examples are just a few of the possible situations:

- You need to enter an element whose attributes will be edited by the user through a graphical user interface.
- The user must send selected element content (or the whole document) to a server for some kind of processing.
- Content authors need to extract pieces of information from a server and insert it directly into the edited XML document.
- You need to apply an XPath expression on the current document and process the nodes of the resulting node set.

To extend the Oxygen XML Editor plugin **Author** mode functionality through Java, you will need the *Oxygen SDK* available *on the Oxygen XML Editor plugin website*. It includes the source code of the **Author** mode operations in the predefined document types and the full documentation (in Javadoc format) of the public API available for **Author** mode custom actions.

The subsequent Java examples make use of AWT classes. If you are developing extensions for the Oxygen XML Editor plugin XML Editor plugin for Eclipse, you will have to use their SWT counterparts.

Attention: Make sure the Java classes of your custom **Author** mode operations are compiled with the same Java version used by Oxygen XML Editor plugin. Otherwise the classes may not be loaded by the Java virtual machine. For example if you run Oxygen XML Editor plugin with a Java 1.6 virtual machine but the Java classes of your custom **Author** mode 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- Simple Use of a Dialog Box from an Author Mode Operation

In this example, we start adding functionality for inserting images in the **Simple Documentation Framework**. 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 box with a text field, in which the user can enter a full URL, or he can browse for a local file.

1. Setup a 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 Editor plugin when the action is configured. It returns the list of arguments (name and type) that are accepted by the operation.
   - The `getDescription` method is used by Oxygen XML Editor plugin when the operation is configured. It returns a description of the operation.

Here is the implementation of these three methods:

```
/**
 * Performs the operation.
 */
public void doOperation(
                AuthorAccess authorAccess,
                ArgumentsMap arguments)
 throws IllegalArgumentException,
                   AuthorOperationException {

 JFrame oxygenFrame = (JFrame) authorAccess.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 cursor 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 Editor plugin website*.

Important:

Make sure you always specify the namespace of the inserted fragments.

```
<image xmlns='http://www.oxygenxml.com/sample/documentation'
  href='path/to/image.png'/>
```

3. Package the compiled class into a jar file. An example of an Ant script that packages the `classes` folder content into a jar archive named `sdf.jar` is listed below:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="project" default="dist">
    <target name="dist">
   <jar destfile="sdf.jar" basedir="classes">
    <fileset dir="classes">
     <include name="**/*"/>
     </fileset>
   </jar>
    </target>
</project>
```

4. Copy the `sdf.jar` file into the `frameworks/sdf` folder.
5. Add the `sdf.jar` to the class path. To do this, *open the **Preferences** dialog box*, go to **Document Type Association**, select **SDF**, and press the **Edit** button.
6. Select the **Classpath** tab in the lower part of the ***Document Type** configuration dialog box* and press the **Add** button . In the displayed dialog box, enter the location of the jar file, relative to the Oxygen XML Editor plugin `frameworks` folder.
7. We now create the action that will use the defined operation. Go to the **Actions** subtab. 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. Next, we 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`.

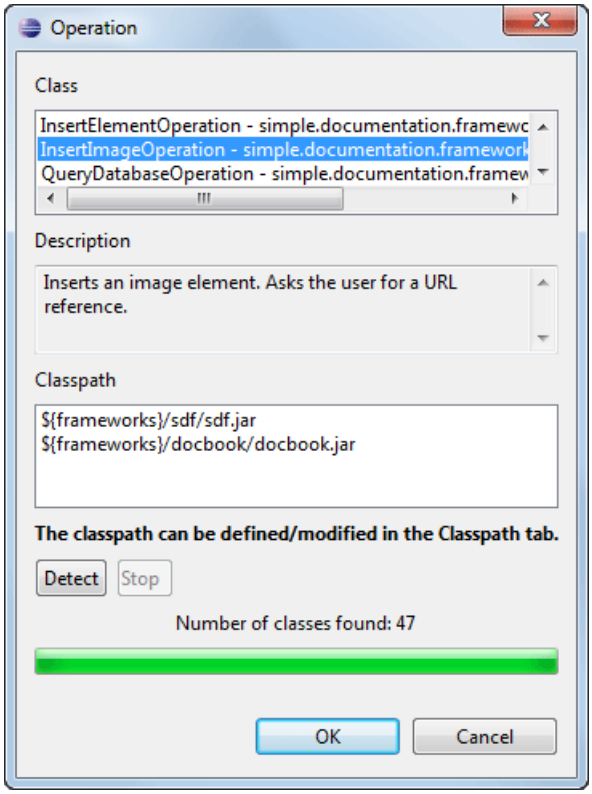


Figure 283: Selecting the Operation

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 cursor inside a `section` between two para elements (for instance). Press the button associated with the action from the toolbar. In the dialog box, 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 a 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 cursor 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 Editor plugin website*.

7. Package the compiled class into a jar file.
8. Copy the jar file and the JDBC driver files into the `frameworks/sdf` directory.
9. Add the jars to the class path. To do this, open the ***Document Type Association*** *preferences page*, select **SDF** and press the **Edit** button. Select the **Classpath** tab in the lower part of the ***Document Type*** *configuration dialog box* and press the **Add** button . In the displayed dialog box, enter the location of the jar file, relative to the Oxygen XML Editor plugin `frameworks` folder.
10. Go to the **Actions** subtab. 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 (Meta 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.

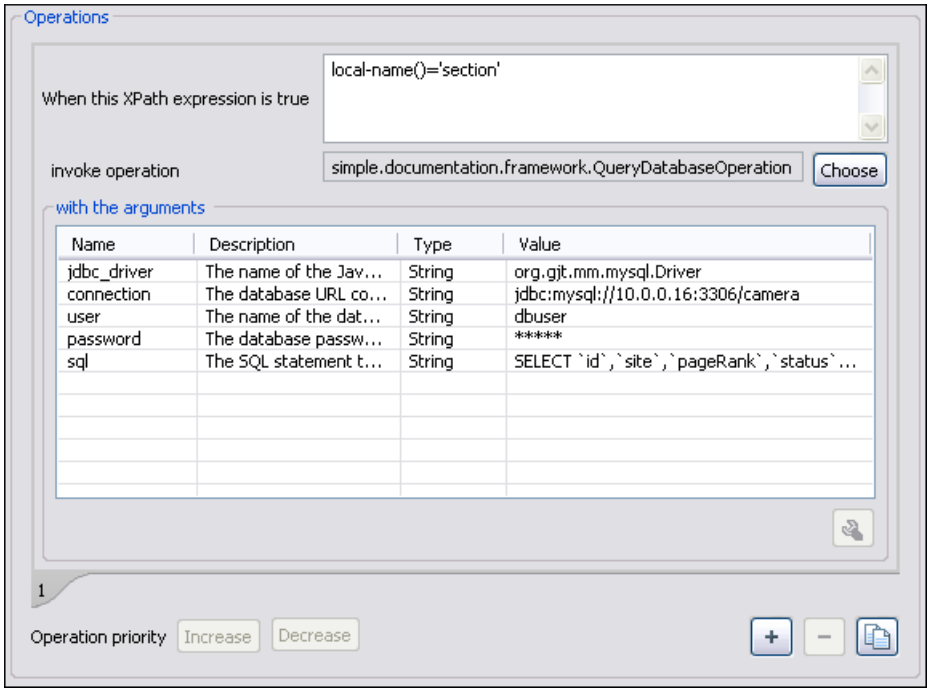


Figure 284: Java Operation Arguments Setup

To test the action, you can open the *sdf_sample.xml* sample and place the cursor 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.

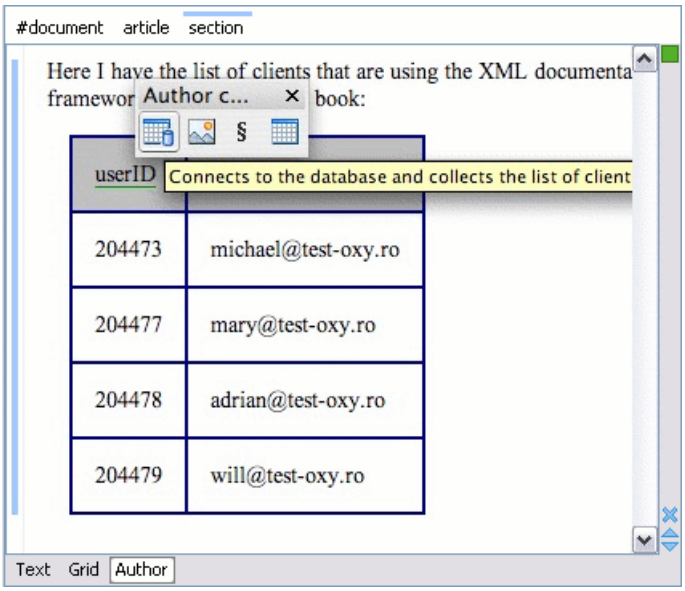


Figure 285: Table Content Extracted from the Database

Localizing Frameworks

Oxygen XML Editor plugin 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.

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 Editor plugin matches the GUI language with the language set in the `translation.xml` file. If 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 the following:

- Framework actions
- Menu entries
- Contextual menus
- Toolbars
- Static CSS content

For example, if you want to localize the bold action, *open the **Preferences** dialog box* and go to **Document Type Association**. Use the **New** or **Edit** button to open the ***Document type** configuration dialog box*, 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 the associated value for a certain key. This allows all the dialog boxes that you present from your custom operations to have labels translated in different languages.

You can also reference 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 box* and go to **Document Type Association** > **Author** > **Actions**. 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.

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`.

Next, we 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 bottom section of the ***Document Type** configuration dialog box*, there are a series of tabs. The first one refers to the schema that is used for validation of the documents that match the defined **Association Rules**.

Important: If the document refers a schema, using for instance a `DOCTYPE` declaration or a `xsi:schemaLocation` attribute, the schema from the document type association will not be used when validating.

Schema Type

Select from the combo box the value **XML Schema**.

Schema URI

Enter the value `${frameworks}/sdf/schema/sdf.xsd`. We should use the *${frameworks}* editor variable in the schema URI path instead of a full path in order to be valid for different Oxygen XML Editor plugin installations.

Important: The *${frameworks}* variable is expanded at the validation time into the absolute location of the directory containing the frameworks.

Second Step - The CSS

If you read the *Simple Customization Tutorial* then you already have some basic notions about creating simple styles. The example document contains elements from different namespaces, so you will use CSS Level 3 extensions supported by the **Author** mode 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 results in Oxygen XML Editor plugin changing 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;
}
```

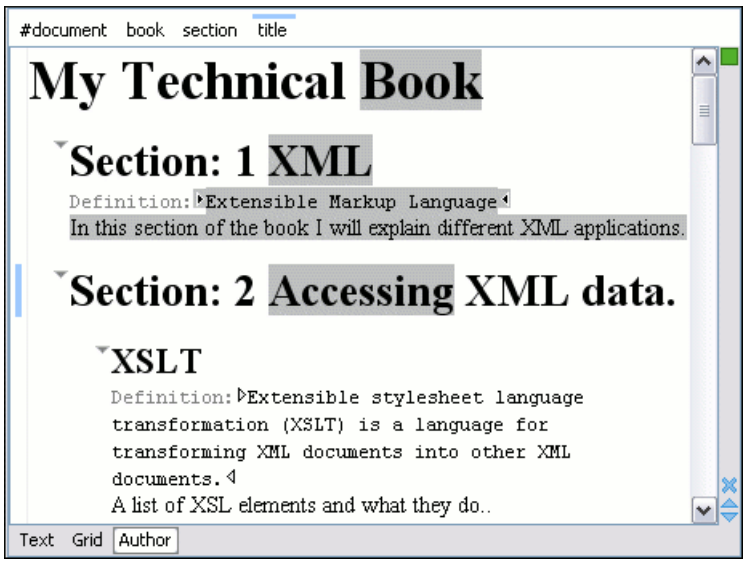


Figure 286: A sample of nested sections and their titles.

In the above screenshot you can see a sample XML document rendered by the CSS stylesheet. The selection "avoids" the text that is generated by the CSS "content" property. This happens because the CSS generated text is not present in the XML document and is just a visual aid.

Styling the Inline Elements

The "bold" style is obtained by using the `font-weight` CSS property with the value `bold`, while the "italic" style is specified by the `font-style` property:

```
b {
    font-weight:bold;
}

i {
    font-style:italic;
}
```

Styling Images

The CSS 2.1 does not specify how an element can be rendered as an image. To overpass this limitation, Oxygen XML Editor plugin supports a CSS Level 3 extension allowing to load image data from an URL. The URL of the image must be specified by one of the element attributes and it is resolved through the catalogs specified in Oxygen XML Editor plugin.

```
image{
    display:block;
    content: attr(href, url);
```

```
    margin-left:2em;
}
```

Our `image` element has the required attribute `href` of type `xs:anyURI`. The `href` attribute contains an image location so the rendered content is obtained by using the function:

```
attr(href, url)
```

The first argument is the name of the attribute pointing to the image file. The second argument of the `attr` function specifies the type of the content. If the type has the `url` value, then Oxygen XML Editor plugin identifies the content as being an image. If the type is missing, then the content will be the text representing the attribute value.

Oxygen XML Editor plugin 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);
}
```

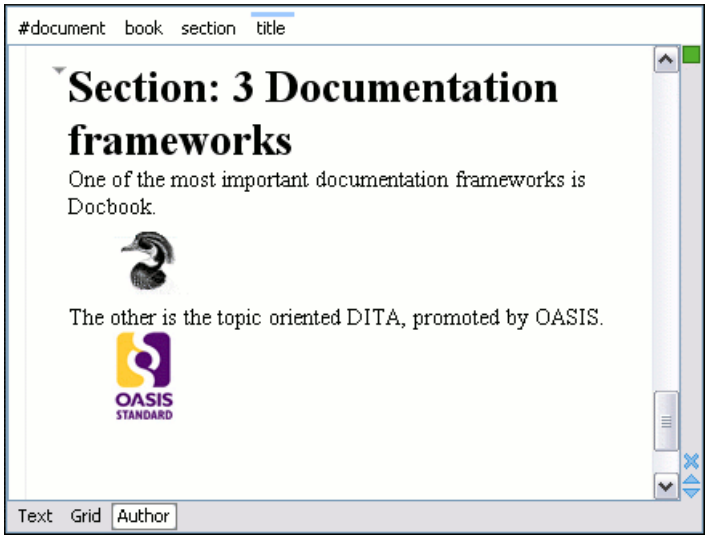


Figure 287: Samples of images in Author

Testing the Document Type Association

To test the new Document Type create an XML instance that is conforming with the *Simple Documentation Framework* association rules. You will not specify an XML Schema location directly in the document, using an `xsi:schemaLocation` attribute; Oxygen XML Editor plugin will detect instead its associated document type and use the specified schema.

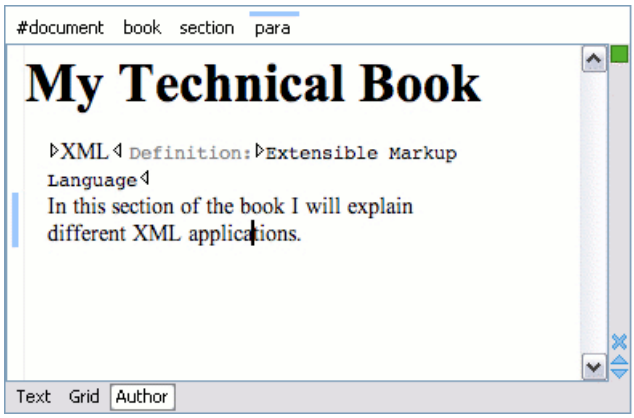
```
<book xmlns="http://www.oxygenxml.com/sample/documentation"
      xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">

    <title>My Technical Book</title>
    <section>
        <title>XML</title>
        <abs:def>Extensible Markup Language</abs:def>
        <para>In this section of the book I will
           explain different XML applications.</para>
    </section>
</book>
```

When trying to validate the document there should be no errors. Now modify the `title` to `title2`. Validate again. This time there should be one error:

```
cvc-complex-type.2.4.a: Invalid content was found starting with element
'title2'. One of '{"http://www.oxygenxml.com/sample/documentation":title}'
is expected.
```

Undo the tag name change. Press on the **Author** button at the bottom of the editing area. Oxygen XML Editor plugin should load the CSS from the document type association and create a layout similar to this:



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 Editor plugin 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 Editor plugin 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 XML Editor plugin 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 Editor plugin by running the provided start-up script files.

You should copy the created schema files `abs.xsd` and `sdf.xsd`, `sdf.xsd` being the master schema, to the `schema` directory and the CSS file `sdf.css` to the `css` directory.

Packaging and Deploying

Using a file explorer, go to the Oxygen XML Editor plugin `[OXYGEN_DIR]/frameworks` directory. Select the `sdf` directory and make an archive from it. Move it to another Oxygen XML Editor plugin installation (eventually on another computer). Extract it in the `[OXYGEN_DIR]/frameworks` directory. Start Oxygen XML Editor plugin and test the association as explained above.

If you create multiple document type associations and you have a complex directory structure it might be easy from the deployment point of view to use an Oxygen XML Editor plugin All Platforms distribution. Add your framework files to it, repackage it and send it to the content authors.

Attention: When deploying your customized `sdf` directory, make sure that your `sdf` directory contains the `sdf.framework` file (that is the file defined as External Storage in the ***Document Type Association*** *preferences page* shall always be stored inside the `sdf` directory). If your external storage points somewhere else Oxygen XML Editor plugin will not be able to update the Document Type Association options automatically on the deployed computers.

Configuring New File Templates

You will create a set of document templates that the content authors will use as starting points for creating *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}`,`${currentFileURL}`, and `${cfdu}` editor variables when you save your documents in a data base.

3. Open the ***Document Type*** *configuration dialog box* 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** document dialog box. 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, such as a file or folder path, a time-stamp, or a date. It is used in the definition of a command (for example, the input URL of a transformation, the output file path of a transformation, or the command line of an external tool) to make a command or a parameter generic and re-usable with other input files. When the same command is applied to different files, the notation is expanded at the execution of the command so that the same command has different effects depending on the actual file.

You can use the following editor variables in Oxygen XML Editor plugin commands of external engines or other external tools, in transformation scenarios, and in validation scenarios:

- *${oxygenHome}* - Oxygen XML Editor plugin installation folder as URL.
- *${oxygenInstallDir}* - Oxygen XML Editor plugin installation folder as file path.
- *${framework}* - The path (as URL) of the current framework, as part of the `[OXYGEN_DIR]/frameworks` directory.
- *${framework(fr_name)}* - The path (as URL) of the `fr_name` framework.
- *${frameworkDir(fr_name)}* - The path (as file path) of the `fr_name` framework.

  Note: Since 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 Editor plugin employs the following algorithm when searching for a given framework name:

  - All frameworks are sorted, from high to low, according to their ***Priority*** setting from the ***Document Type*** *configuration dialog box*. Only frameworks that have the **Enabled** checkbox set are taken into account.
  - Next, if the two or more frameworks have the same name and priority, a further sorting based on the **Storage** setting is made, in the exact following order:
    - Frameworks stored in the internal Oxygen XML Editor plugin options.
    - Additional frameworks added in the ***Locations*** *preferences page*.
    - Frameworks installed using the add-ons support.
    - Frameworks found in the *main frameworks location* (**Default** or **Custom**).

- *${frameworks}* - The path (as URL) of the `[OXYGEN_DIR]` directory.
- *${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.
- *${af}* - The local file path of the ZIP archive that includes the current edited document.
- *${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 cursor is located. This variable can be used in a code template, in **Author** mode operations, or in a selection plugin.
- *${selection}* - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** mode operations, or in a selection plugin.
- *${id}* - Application-level unique identifier; a short sequence of 10-12 letters and digits which is not guaranteed to be universally unique.
- *${uuid}* - Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java *UUID* class.
- *${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, with one of the following values:

Parameter	
`url`	**Format:** `${ask('message', url, 'default_value')}`
	Description: Input is considered a URL. Oxygen XML Editor plugin checks that the provided URL is valid.
	Example:

Parameter	
	• `${ask('Input URL', url)}` - The displayed dialog box has the name `Input URL`. The expected input type is URL. • `${ask('Input URL', url, 'http://www.example.com')}` - The displayed dialog box has the name `Input URL`. The expected input type is URL. The input field displays the default value `http://www.example.com`.
`password`	**Format**: *${ask('message', password, 'default')}*
	Description: The input is hidden with bullet characters.
	Example: • `${ask('Input password', password)}` - The displayed dialog box has the name `'Input password'` and the input is hidden with bullet symbols. • `${ask('Input password', password, 'abcd')}` - The displayed dialog box has the name `'Input password'` and the input hidden with bullet symbols. The input field already contains the default `abcd` value.
`generic`	**Format**: *${ask('message', generic, 'default')}*
	Description: The input is considered to be generic text that requires no special handling.
	Example: • `${ask('Hello world!')}` - The dialog box has a `Hello world!` message displayed. • `${ask('Hello world!', generic, 'Hello again!')}` - The dialog box has a `Hello world!` message displayed and the value displayed in the input box is `'Hello again!'`.
`relative_url`	**Format:** `${ask('message', relative_url, 'default')}`
	Description: Input is considered a URL. Oxygen XML Editor plugin tries to make the URL relative to that of the document you are editing. **Note:** If the `$ask` editor variable is expanded in content that is not yet saved (such as an *untitled* file, whose path cannot be determined), then Oxygen XML Editor plugin will transform it into an absolute URL.
	Example: • *${ask('File location', relative_url, 'C:/example.txt')}* - The dialog box has the name `'File location'`. The URL inserted in the input box is made relative to the current edited document location.
`combobox`	**Format:** `${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a drop-down menu. The drop-down menu is populated with the given `rendered_value` values. Choosing such a value will return its associated value (`real_value`). **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example:

Parameter	
	• `${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The drop-down menu displays the three given operating systems. The associated value will be returned based upon your selection. **Note:** In this example, the default value is indicated by the `osx` key. However, the same result could be obtained if the default value is indicated by `Mac OS X`, like in the following example: `${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'Mac OS X')}` • `${ask('Mobile OS', combobox, ('win':'Windows Mobile';'ios':'iOS';'and':'Android'), 'Android')}`
`editable_combobox`	**Format:** `${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a drop-down menu with editable elements. The drop-down menu is populated with the given `rendered_value` values. Choosing such a value will return its associated real value (`real_value`) or the value inserted when you edit a list entry. **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The drop-down menu displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.
`radio`	**Format:** `${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a series of radio buttons. Each radio button displays a `'rendered_value` and will return an associated `real_value`. **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The radio button group allows you to choose between the three operating systems. **Note:** In this example `Mac OS X` is the default selected value and if selected it would return `osx` for the output.

  - `'default-value'` - optional parameter. Provides a default value.

- *${date(pattern)}* - Current date. The allowed patterns are equivalent to the ones in the *Java SimpleDateFormat class*. **Example:** `yyyy-MM-dd`;

  Note: This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to *http://www.w3.org/TR/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 a parameter of an **Author** mode operation.

    Example:

    ```
    ${ask('Set new ID attribute', generic, '${xpath_eval(@id)}')}
    ```

- *${i18n(key)}* - Editor variable used only at framework level to allow translating names and descriptions of **Author** mode actions in multiple actions. For more details see the *Localizing Frameworks* on page 585 section.

Custom Editor Variables

An editor variable can be created 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, or a custom FO processor.

You can create or configure custom editor variables in the ***Custom Editor Variables** preferences page*. To create a custom editor variable, click the **New** button and 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** of that variable.

Create Your Own Stylesheet Templates

Oxygen XML Editor plugin 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 Editor plugin installation directory.
- Open Oxygen XML Editor plugin 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 reference 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** configuration dialog box*.

To test the catalog settings, restart Oxygen XML Editor plugin and try to validate a new sample **Simple Documentation Framework** document. There should be no errors.

> The `sdf.xsd` schema that validates the document refers the other file `abs.xsd` through an import element:
>
> ```
> <xs:import namespace=
>  "http://www.oxygenxml.com/sample/documentation/abstracts"
>  schemaLocation="http://www.oxygenxml.com/SDF/abs.xsd"/>
> ```
>
> The `schemaLocation` attribute references the `abs.xsd` file:
>
> ```
> xsi:schemaLocation="http://www.oxygenxml.com/sample/documentation/abstracts"
>     http://www.oxygenxml.com/SDF/abs.xsd"/>
> ```
>
> The catalog mapping is:
>
> ```
> http://www.oxygenxml.com/SDF/abs.xsd -> schema/abs.xsd
> ```
>
> This means that all the references to http://www.oxygenxml.com/SDF/abs.xsd must be resolved to the `abs.xsd` file located in the `schema` directory. The URI element is used by URI resolvers, for example for resolving a URI reference used in an XSLT stylesheet.

Configuring Transformation Scenarios

When distributing a framework to the users, it is a good idea to have the transformation scenarios already configured. This helps the content authors publish their work in various formats. Being contained in the **Document Type Association**, the scenarios can be distributed along with the actions, menus, toolbars, and catalogs.

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 box* and go to **Document Type Associations**. Open the **Document Type** dialog for the **SDF** framework then choose the **Transformation** tab. Click the **New** button and choose the appropriate type of transformation (for example, **XML transformation with XSLT**).

   In the **New scenario** dialog box, fill in 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`.

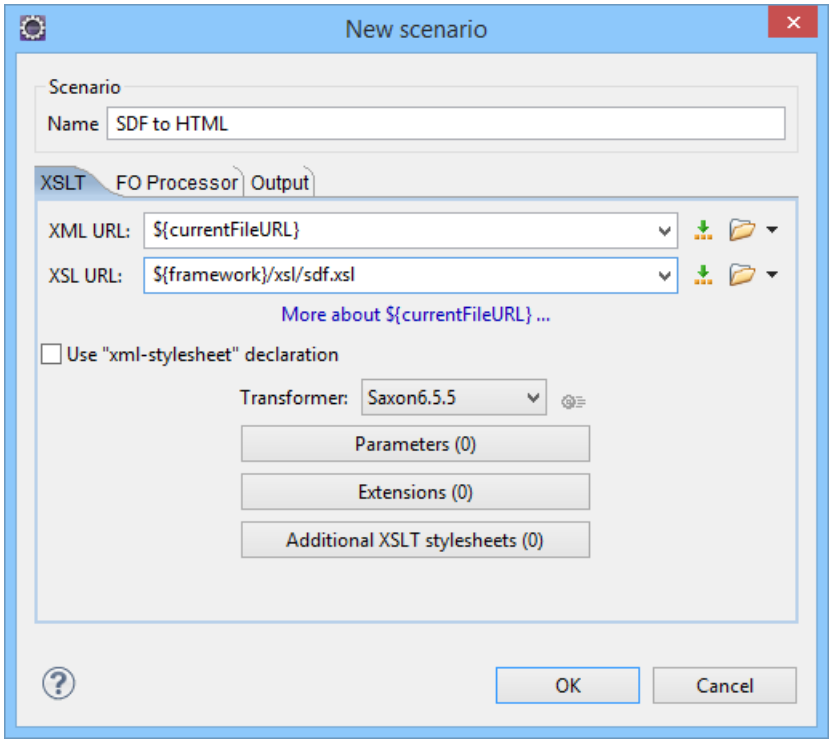


Figure 288: Configuring a New Transformation Scenario

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: To set the browser or system application that will be used, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**. This will take precedence over the default system application settings.

   - Enable the **Saved file** option.

5. Click the **OK** button to save the new scenario.

Now the scenario is listed in the **Transformation** tab:

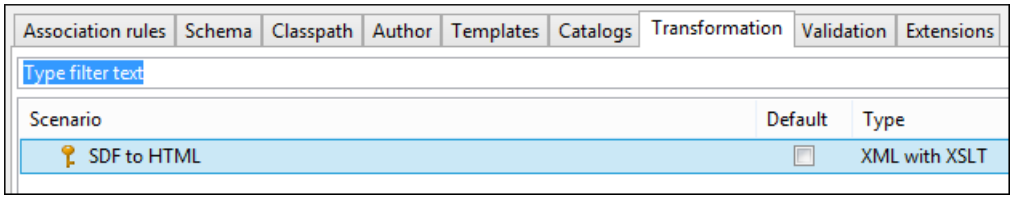


Figure 289: The transformation tab

To test the transformation scenario that you just created, open the **SDF** XML sample from the *Example Files Listings*. Click the **Apply Transformation Scenario(s)** button to display the **Transform with** dialog box. The scenario list contains the scenario you defined earlier . Select the *SDF to HTML* scenario that you just defined and click the **Apply associated** button. The HTML file is saved in the same folder as the XML file and displayed in the browser.

Configuring Validation Scenarios

You can distribute a framework with a series of already configured validation scenarios. Also, this provides enhanced validation support that allows 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 box* and go to **Document Type Association**.
2. **Edit** the specific framework to open the ***Document Type** configuration dialog box*, then choose the **Validation** tab. This tab displays a list of document types in which you can define validation scenarios. To set one of the validation scenarios as the default for a specific document type, check the **Default** box for that specific document type.
3. To add a new scenario, press the **New** button.
   The **New scenarios** dialog box is displayed. It lists all the validation units for the scenario.

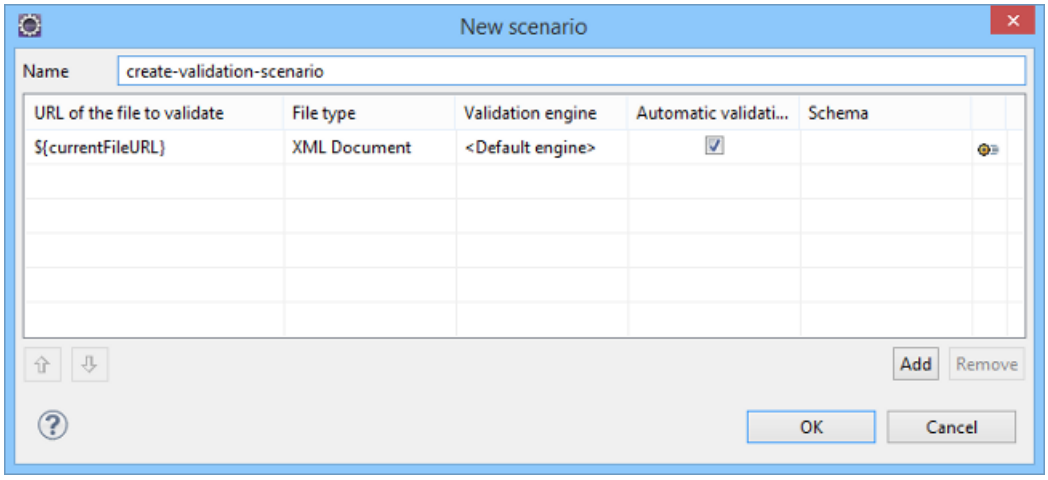


Figure 290: Add / Edit a Validation Unit

The **New scenario** dialog box includes the following information:

- **Name** - The name of the current validation scenario.
- **URL of the file to validate** - The URL of the main module that includes the current module. It is also the entry module of the validation process when the current one is validated.
- **File type** - The type of the document that is validated in the current validation unit. Oxygen XML Editor plugin automatically selects the file type depending on the value of the **URL of the file to validate** field.
- **Validation engine** - One of the engines available in Oxygen XML Editor plugin for validation of the type of document to which the current module belongs. **Default engine** means that the default engine is used to run the validation and is set in the **Preferences** pages for the current document type (XML, XML Schema, XSLT, XQuery, etc.).

- **Automatic validation** - If this option is checked, the validation operation defined by this row is also applied by *the automatic validation feature*. If the **Automatic validation** feature is *disabled in Preferences*, then this option is ignored, as the Preference setting has a higher priority.
- **Schema** - This option becomes active when you set the **File type** to **XML Document**.
- **Settings** - Opens the **Specify Schema** dialog box that allows you to set a schema for validating XML documents, or a list of extensions for validating XSL or XQuery documents. You can also set a default phase for validation with a Schematron schema.

4. If you want to add a new validation unit, press the **Add** button.
5. To edit the URL of the main validation module, click its cell in the **URL of the file to validate** column.

   Specify the URL of the main module by doing one of the following:

   - Enter the URL in the text field or select it from the drop-down list.
   - Use the **Browse** drop-down button to browse for a local, remote, or archived file.
   - Use the **Insert Editor Variable** button to insert an *editor variable* or a *custom editor variable*.

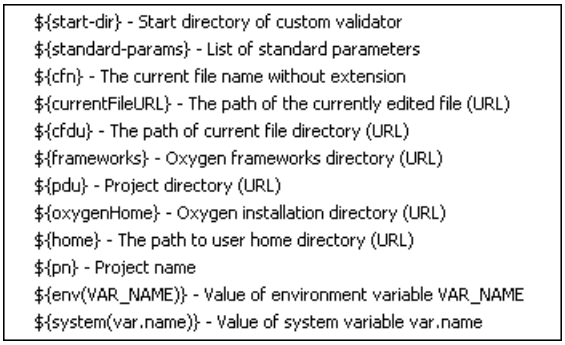


Figure 291: Insert an Editor Variable

6. Select the **File type** of the validated document.

   Note that this determines the list of possible validation engines.
7. Select the **Validation engine** by clicking its cell and selecting it from the drop-down list.
8. Select the **Automatic validation** option if you want the current unit to be used by *the automatic validation feature*.
9. Choose the schema to be used during validation (the schema detected after parsing the document or a custom one).
10. Press **Ok**.

    The newly created validation scenario is now included in the list of scenarios in the **Validation** tab.

Configuring Extensions

You can add extensions to your Document Type Association using the **Extensions** tab from the ***Document Type** configuration dialog box*.

Configuring an Extensions Bundle

Starting with Oxygen XML Editor plugin 10.3 version a single bundle was introduced acting as a provider for all other extensions. The individual extensions can still be set and if present they take precedence over the single provider, but this practice is being discouraged and the single provider should be used instead. To set individual extensions, *open the **Preferences** dialog box*, 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 Editor plugin 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** editing 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 Editor plugin, 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 Mode Schema Aware Editing Handler* on page 607 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** mode 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 referenced 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** editing mode, when the drop action changes, and when the drop occurs. Each method receives the `DropTargetEvent` containing information about the drag and drop operation. The drag and drop extensions are available on **Author** mode for both Oxygen XML Editor plugin Eclipse plugin and standalone application. The Text mode corresponding listener is available only for Oxygen XML Editor plugin Eclipse plugin. The methods corresponding to each implementation are: `createAuthorAWTDndListener`, `createTextSWTDndListener` and `createAuthorSWTDndListener`.

```
public AuthorDnDListener createAuthorAWTDndListener() {
     return new SDFAuthorDndListener();
}
```

   For more details about the **Author** mode 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 referenced resource is **location**. To be able to obtain the content of the referenced 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** editing 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** editing 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. If 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 Editor plugin 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 class path. To do this, *open the **Preferences** dialog box*, go to **Document Type Association**, select **SDF**, press the **Edit** button, select the **Classpath** tab and press the ✚ **Add** button . In the displayed dialog box, enter the location of the jar file, relative to the Oxygen XML Editor plugin `frameworks` folder.
15. Register the Java class by going to the **Extensions** tab. Press the **Choose** button and select 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 Editor plugin website*.

Customize Profiling Conditions

For each document type, you can configure the phrase-type elements that wrap the profiled content by setting a custom *`ro.sync.ecss.extensions.api.ProfilingConditionalTextProvider`*. This configuration is set by default for DITA and DocBook frameworks.

Customizing Smart Paste Support

The *Smart Paste* feature preserves certain style and structure information when copying content from some of the most common applications and pasting into *document types that support Smart Paste* in Oxygen XML Editor plugin. For other document types, the default behavior of the paste operation is to keep only the text content without the styling.

The style of the pasted content can be customized by editing an XSLT stylesheet for the particular document type. The XSLT stylesheet must accept an XHTML flavor of the copied content as input, and transform it to the equivalent XML markup that is appropriate for the target document type of the paste operation.

A default XSLT stylesheet that can be edited to customize the mapping between the markup of the copied content and the markup of the pasted content is included the following directory for each supported document type: `[OXYGEN INSTALLATION DIRECTORY]/frameworks/[Document Type]/resources`. The name of this default XSLT file begins with `xhtml2`, followed by the document type (for example, `xhtml2dita.xsl` for the DITA document type).

> **The Default Stylesheet for DITA Document Type**
>
> To customize the mapping for a DITA topic, edit the `[OXYGEN INSTALLATION DIRECTORY]/frameworks/dita/resources/xhtml2dita.xsl` stylesheet. To test the modifications done in the stylesheet, you can start pasting content without having to restart Oxygen XML Editor plugin.

Note: Oxygen XML Editor plugin implements the Smart Paste feature by *setting up* a stylesheet which is obtained by Oxygen XML Editor plugin from the `getImporterStylesheetFileName` method. This method is available in an instance of *the `AuthorExternalObjectInsertionHandler` class*, which is returned by the `createExternalObjectInsertionHandler` method of *the `ExtensionsBundle` instance* of the target document type.

Implementing an *Author Extension* State Listener

The *`ro.sync.ecss.extensions.api.AuthorExtensionStateListener`* implementation is notified when the *Author Extension* where the listener is defined is activated or deactivated in the Document Type detection process.

Note: The Javadoc documentation of the *Author API* used in the example files is *available on the Oxygen XML Editor plugin website*. Also it is available in the *oXygen 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** editing mode, should be used to perform custom initializations and to register listeners, such as *`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 OptionListener.
   authorAccess.getOptionsStorage().addOptionListener(sdfOptionListener);

   // Add author DocumentListeners.
   sdfAuthorDocumentListener = new SDFAuthorListener();
   authorAccess.getDocumentController().addAuthorListener(
               sdfAuthorDocumentListener);

   // Add MouseListener.
   sdfMouseListener = new SDFAuthorMouseListener();
   authorAccess.getEditorAccess().addAuthorMouseListener(sdfMouseListener);

   // Add CaretListener.
   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 specific **Author** mode actions and informations related to components such as 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 authorAccess. 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** mode 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 authorAccess. The document controller can also be used to perform operations involving document modifications.

To provide access to the **Author** mode component-related functionality and information, the authorAccess 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 cursor 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 OptionListener.
  authorAccess.getOptionsStorage().removeOptionListener(sdfOptionListener);

  // Remove DocumentListeners.
  authorAccess.getDocumentController().removeAuthorListener(
              sdfAuthorDocumentListener);

  // Remove MouseListener.
  authorAccess.getEditorAccess().removeAuthorMouseListener(sdfMouseListener);

  // Remove CaretListener.
  authorAccess.getEditorAccess().removeAuthorCaretListener(sdfCaretListener);

  // Other actions...

}
```

Implementing an Author Mode Schema Aware Editing Handler

To implement your own handler for actions such as 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 Editor plugin 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 if ***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` makes it possible to handle other events, such as 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 Editor plugin website*.

Configuring a Content Completion Handler

You can filter or contribute to items offered for content completion by implementing the *`ro.sync.contentcompletion.xml.SchemaManagerFilter`* interface.

Note: The Javadoc documentation of the *Author API* used in the example files is *available on the Oxygen XML Editor plugin 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 Editor plugin or by contributing to the list of proposals. The filter can be applied on elements, attributes or on their values. Attributes filtering can be implemented using the `filterAttributes` method and changing the default content completion list of *`ro.sync.contentcompletion.xml.CIAttribute`* for the element provided by the current *`ro.sync.contentcompletion.xml.WhatAttributesCanGoHereContext`* context. For example, the `SDFSchemaManagerFilter` checks if the element from the current context is the `table` element and adds the `frame` attribute to the `table` list of attributes.

```
/**
 * Filter attributes of the "table" element.
 */
public List<CIAttribute> filterAttributes(List<CIAttribute> attributes,
    WhatAttributesCanGoHereContext context) {
  // If the element from the current context is the 'table' element add the
  // attribute named 'frame' to the list of default content completion proposals
  if (context != null) {
    ContextElement contextElement = context.getParentElement();
```

```
    if ("table".equals(contextElement.getQName())) {
      CIAttribute frameAttribute = new CIAttribute();
      frameAttribute.setName("frame");
      frameAttribute.setRequired(false);
      frameAttribute.setFixed(false);
      frameAttribute.setDefaultValue("void");
      if (attributes == null) {
        attributes = new ArrayList<CIAttribute>();
      }
      attributes.add(frameAttribute);
    }
  }
  return attributes;
}
```

The elements that can be inserted in a specific context can be filtered using the `filterElements` method. The `SDFSchemaManagerFilter` uses this method to replace the `td` child element with the `th` element when `header` is the current context element.

```
public List<CIElement> filterElements(List<CIElement> elements,
    WhatElementsCanGoHereContext context) {
  // If the element from the current context is the 'header' element remove the
  // 'td' element from the list of content completion proposals and add the
  // 'th' element.
  if (context != null) {
    Stack<ContextElement> elementStack = context.getElementStack();
    if (elementStack != null) {
      ContextElement contextElement = context.getElementStack().peek();
      if ("header".equals(contextElement.getQName())) {
        if (elements != null) {
          for (Iterator<CIElement> iterator = elements.iterator(); iterator.hasNext();) {
            CIElement element = iterator.next();
            // Remove the 'td' element
            if ("td".equals(element.getQName())) {
              elements.remove(element);
              break;
            }
          }
        } else {
          elements = new ArrayList<CIElement>();
        }
        // Insert the 'th' element in the list of content completion proposals
        CIElement thElement = new SDFElement();
        thElement.setName("th");
        elements.add(thElement);
      }
    }
  } else {
    // If the given context is null then the given list of content completion elements contains
    // global elements.
  }
  return elements;
}
```

The elements or attributes values can be filtered using the `filterElementValues` or `filterAttributeValues` methods.

Note: The complete source code can be found in the Simple Documentation Framework project, included in the **oxygen-sample-framework** module of the *Oxygen SDK* , available as a Maven archetype *on the Oxygen XML Editor plugin website*.

Configuring a Link target element finder

The link target reference finder represents the support for finding references from links which indicate specific elements inside an XML document. This support will only be available if a schema is associated with the document type.

If you do not define a custom link target reference finder, the `DefaultElementLocatorProvider` implementation will be used by default. The interface which should be implemented for a custom link target reference finder is *`ro.sync.ecss.extensions.api.link.ElementLocatorProvider`*. As an alternative, the *`ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider`* implementation can also be extended.

The used `ElementLocatorProvider` will be queried for an *`ElementLocator`* when a link location must be determined (when a link is clicked). Then, to find the corresponding (linked) element, the obtained `ElementLocator` will be queried for each element from the document.

Note: The Javadoc documentation of the *Author API* used in the example files is *available on the Oxygen XML Editor plugin 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 Editor plugin website*. Also it is available in the *Oxygen SDK Maven Project*.

Table 8: 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 reference. In our case the element which has references is **ref** and the attribute indicating the referenced resource is **location**. You will have to implement a Java extension class for obtaining the referenced resources.

Note: The Javadoc documentation of the *Author API* used in the example files is *available on the Oxygen XML Editor plugin 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 referenced content. It takes as argument an `AuthorNode` that represents the node for which the display name is needed. The referenced 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** mode 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 referenced content. It takes as arguments an `AuthorNode` that represents the node with the reference and the `AuthorAccess` with access methods to the **Author** mode 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 Editor plugin website*.

In the listing below, the XML document contains the **ref** element:

```
<ref location="referenced.xml">Reference</ref>
```

When no reference resolver is specified, the reference has the following layout:

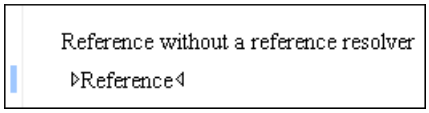


Figure 292: Reference with no specified reference resolver

When the above implementation is configured, the reference has the expected layout:

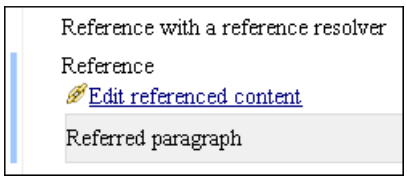


Figure 293: Reference with reference resolver

Configuring CSS Styles Filter

You can modify the CSS styles for each `ro.sync.ecss.extensions.api.node.AuthorNode` rendered in the **Author** mode using an implementation of `ro.sync.ecss.extensions.api.StylesFilter`. You can implement the various callbacks of the interface either by returning the default value given by Oxygen XML Editor plugin or by contributing to the value. The received styles `ro.sync.ecss.css.Styles` can be processed and values can be overwritten with your own. For example you can override the `KEY_BACKGROUND_COLOR` style to return your own implementation of `ro.sync.exml.view.graphics.Color` or override the `KEY_FONT` style to return your own implementation of `ro.sync.exml.view.graphics.Font`.

Note: The Javadoc documentation of the *Author API* used in the example files is *available on the Oxygen XML Editor plugin 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 Editor plugin 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;
}
```

Since in the *schema*, the `td` tag has the attributes **row_span** and **column_span** that are not automatically recognized by Oxygen XML Editor plugin, 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 Editor plugin. 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 622.

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** mode we need to provide the means to determine them. As explained in the *Configuring tables* on page 616, if you use the table element attribute **width** Oxygen XML Editor plugin can determine the table width automatically. In this example the table has `col` elements with **width** attributes that are not recognized by default. You will need to implement a Java extension class to determine the column widths.

Note: The Javadoc documentation of the *Author API* used in the example files is *available on the Oxygen XML Editor plugin 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 Editor plugin website*.

In the listing below, the XML document contains the table element:

```
<table width="300">
    <customcol width="50.0px"/>
    <customcol width="1*"/>
    <customcol width="2*"/>
    <customcol width="20%"/>
    <header>
        <td>C1</td>
        <td>C2</td>
        <td>C3</td>
        <td>C4</td>
    </header>
    <tr>
        <td>cs=1, rs=1</td>
        <td>cs=1, rs=1</td>
        <td row_span="2">cs=1, rs=2</td>
        <td row_span="3">cs=1, rs=3</td>
    </tr>
    <tr>
        <td>cs=1, rs=1</td>
        <td>cs=1, rs=1</td>
    </tr>
    <tr>
        <td column_span="3">cs=3, rs=1</td>
    </tr>
</table>
```

When no table column width provider is specified, the table has the following layout:

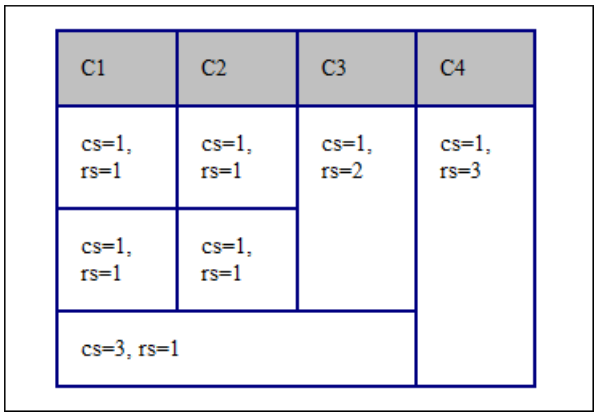


Figure 294: Table layout when no column width provider is specified

When the above implementation is configured, the table has the correct layout:

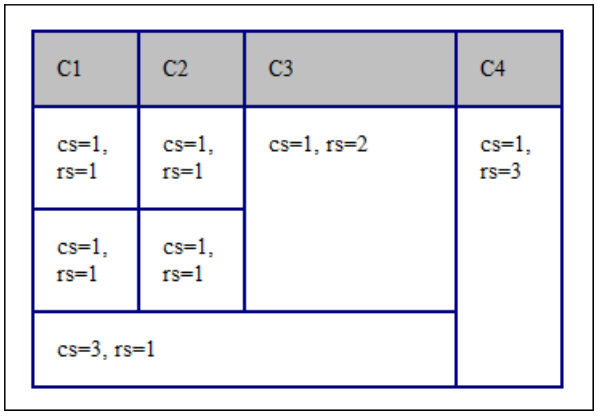


Figure 295: Columns with custom widths

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 616, you need to indicate Oxygen XML Editor plugin a method to determine the cell spanning. If you use the cell element attributes `rowspan` and `colspan` or `rows` and `cols`, Oxygen XML Editor plugin can determine the cell spanning automatically. In our example the `td` element uses the attributes `row_span` and `column_span` that are not recognized by default. You will need to implement a Java extension class for defining the cell spanning.

Note: The Javadoc documentation of the *Author API* used in the example files is *available on the Oxygen XML Editor plugin 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 Editor plugin website*.

6. In the listing below, the XML document contains the table element:

```
<table>
    <header>
        <td>C1</td>
        <td>C2</td>
        <td>C3</td>
        <td>C4</td>
    </header>
    <tr>
        <td>cs=1, rs=1</td>
        <td column_span="2" row_span="2">cs=2, rs=2</td>
        <td row_span="3">cs=1, rs=3</td>
    </tr>
```

```
    <tr>
        <td>cs=1, rs=1</td>
    </tr>
    <tr>
        <td column_span="3">cs=3, rs=1</td>
    </tr>
</table>
```

When no table cell span provider is specified, the table has the following layout:

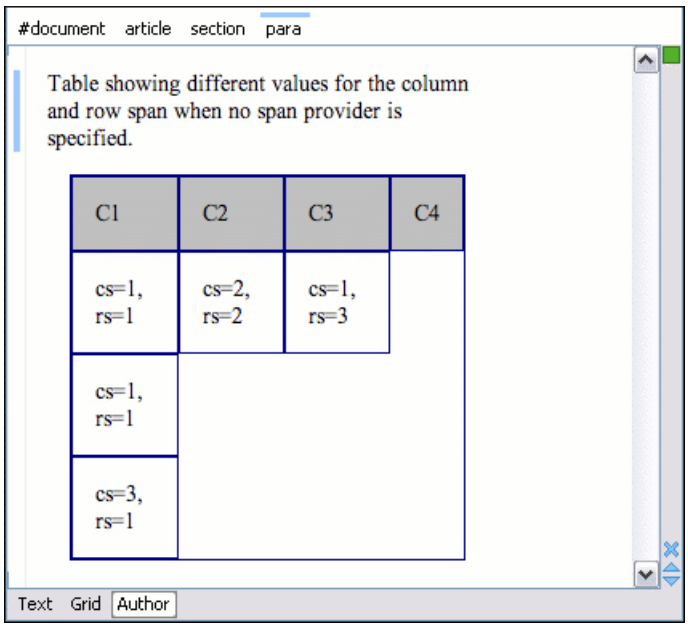


Figure 296: Table layout when no cell span provider is specified

When the above implementation is configured, the table has the correct layout:

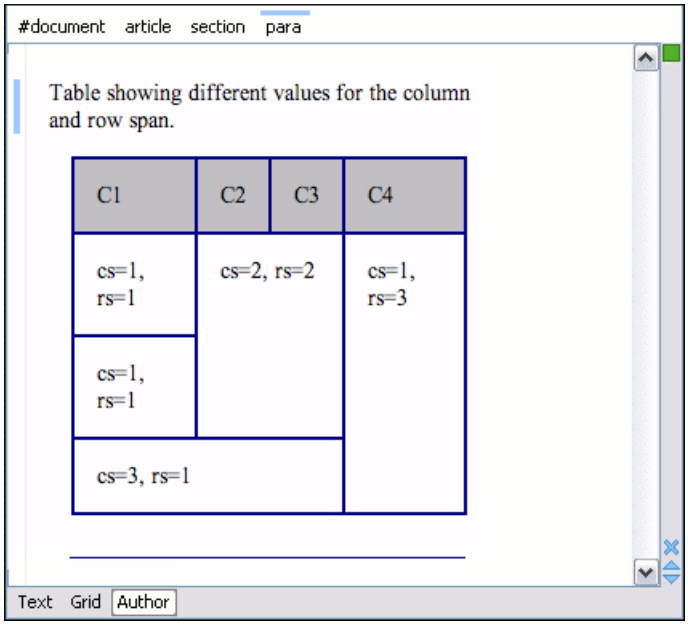


Figure 297: Cells spanning multiple rows and columns.

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 616 section which describes the CSS properties needed for defining a table, you need to indicate Oxygen

XML Editor plugin 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 Editor plugin 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 Editor plugin 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 Editor plugin 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 298: 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 Editor plugin 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 Editor plugin uses `XMLNodeRendererCustomizer` implementations for the following frameworks: DITA, DITA Map, 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 Editor plugin 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** configuration dialog box* (**Extensions bundle** field in the **Extensions** tab).
- As an individual extension, associated with your document type in the ***Document type** configuration dialog box* (**XML node renderer customizer** field in the **Individual extensions** section of the **Extensions** tab).

Support for Retina/HiDPI Displays

To support Retina or HiDPI displays, the icons provided by the `XMLNodeRendererCustomizer` should be backed up by a copy of larger size using the proper *Retina/HiDPI naming convention*.

For example, for the `title` element, if the `XMLNodeRendererCustomizer` returns the path `${framework}/images/myImg.png`, then in order to support Retina images with a scaling factor of 2, an extra file (`myImg@2x.png`)should be added to the same images directory (`${framework}/images/myImg@2x.png`). If the higher resolution icon (the *@2x* file) does not exist, the *normal* icon is scaled and used instead.

For more information about using Retina/HiDPI images, refer to the *Using Retina/HiDPI Images in Author Mode* section.

Customizing the Main CSS of a Document Type

The easiest way to customize the *main* CSS stylesheet of a document type is to create a new CSS stylesheet, save it as an *alternate* CSS file that will be applied as an additional layer to the *main* CSS, and then select it from the **Styles** drop-down menu in **Author** mode.

For example, suppose that you want to customize the *main* CSS for DITA documents. To do this, follow these steps:

1. First, create a new CSS stylesheet and save it in the `[OXYGEN_DIR]/frameworks/dita/css/edit` folder (where the default *main* stylesheet named `style-basic.css` is located).
2. Edit the DITA framework and go to its **CSS** subtab:
   a) *Open the **Preferences** dialog box* and go to **Document Type Association**.
   b) Select the DITA document type and press the **Edit** button.
   c) Go to the **CSS** subtab of the **Author** tab.

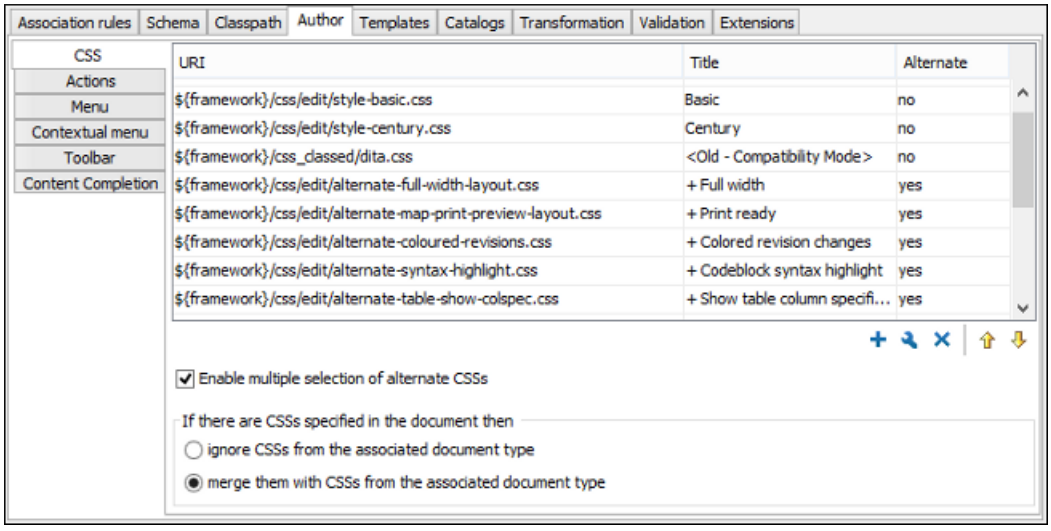
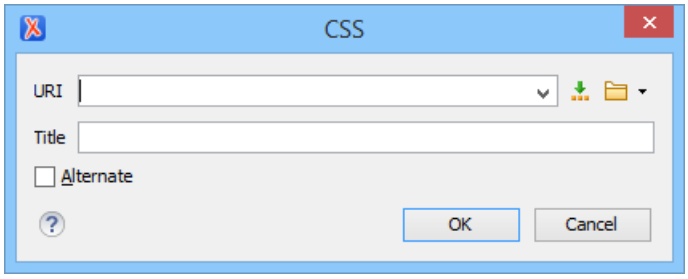


Figure 299: CSS Subtab of the Document Type Association Author Tab

3. Add the new stylesheet as an *alternate* CSS stylesheet:

a) Click the + **Add** button to open a dialog box that allows you to specify the URI and Title for your newly created stylesheet.
b) Check the **Alternate** option to define it as an *alternate* stylesheet that will applied as an additional layer to the *main* CSS.



4. Press **OK** in all the dialog boxes to validate the changes.
5. Select your newly created CSS stylesheet from the **Styles** drop-down menu on the toolbar in **Author** mode.
You can now edit DITA documents based on the new CSS stylesheet. You can also edit the new CSS stylesheet itself and see its effects on rendering DITA documents in the **Author** mode by using the **Refresh** action that is available on the **Author** toolbar and in the **DITA** menu.

Sharing a Document Type (Framework)

Oxygen XML Editor plugin allows you to share customizations of a specific XML type by creating your own *Document Type (framework)* in the ***Document Type Association*** *preferences page*.

A document type (framework) can be shared with other authors by using the following method:

Save it in a Custom Folder

To share your customized framework with other members of your team, you can save it to a separate folder inside the `[OXYGEN_DIR]/frameworks` directory by following these steps:

Important: For this approach to work, the application must be installed in a folder with full write access.

1. Go to `[OXYGEN_DIR]/frameworks` and create a directory for your new framework (for example, `custom_framework`). This directory will contain the resources for your customized framework. See the **DocBook** framework structure from `[OXYGEN_DIR]/frameworks/docbook` as an example.
2. Create your custom document type (framework) and specify the `custom_framework` directory for the **External** storage option.
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. See the *Author Mode Customization Guide* on page 552 section for more details on creating and configuring a new document type (framework).
4. Add any additional resources (CSS files, new file templates, schemas used for validation, catalogs, etc.) to the directory you created in step 1.
5. After completing your customizations in the ***Document Type Association*** *preferences page*, you should have a new configuration file saved in: `[OXYGEN_DIR]/frameworks/custom_framework/custom.framework`.
6. To share the new framework directory with other users, have them copy it to their `[OXYGEN_DIR]/frameworks` directory. The new framework will be available in the list of Document Types when Oxygen XML Editor plugin starts.

Note: If 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.

Sharing an Extended Document Type (Framework)

You can extend a predefined, built-in document type (such as **DITA** or **DocBook**) using the ***Document Type Association*** *preferences page*, make modifications to it, and then share the extension with your team.

Extending a Framework to be Shared

For the purpose of providing specific instructions for sharing an extended framework, suppose that you want to share an extension of the **DITA** framework in which you have removed certain elements from the content completion list. The follow steps describe how you can create an extended framework that can be shared with others:

1. In a location where you have full write access, create a folder structure similar to this: `custom_frameworks/dita-extension`.
2. *Open the* ***Preferences*** *dialog box* and go to ***Document Type Association*** > ***Locations***. In this preferences page, add the path to your `custom_frameworks` folder in the **Additional frameworks directories** list.
3. Go to the ***Document Type Association*** *preferences page* and select the **DITA** document type configuration and use the **Extend** button to create an extension for it.
4. Give the extension an appropriate name (for example, `DITA - Custom`), select **External** for the **Storage** option, and specify an appropriate path (for example, `path/to/.../custom_frameworks/dita-extension/dita-extension.framework`).
5. Make your changes to the extension. For example, you could go to the **Content Completion** subtab of the **Author** tab and in the **Filter - Remove content completion items** list, add elements that you do not want to be presented to the end users.
6. Click **OK** to close the dialog box and then **OK** or **Apply** to save the changes to the ***Document Type Association*** *preferences page*.

   Results

   After you perform these steps you will have a fully functional framework in the `dita-extension` folder and it can be shared with others.

Sharing the Extended Framework

There are several ways that you can share the extended framework with others:

- Copy it to their `[OXYGEN_DIR]/frameworks` directory.
- Create a `custom_frameworks` folder (anywhere on disk) and copy the extended framework into it. Then add the path to your `custom_frameworks` folder in the **Additional frameworks directories** list in the ***Document Type Association*** > ***Locations*** *preferences page*.

After your team members install the framework they can check the list of Document Types in the ***Document Type Association*** *preferences page* to see if the framework is present and if it appears before the bundled **DITA** framework (meaning that it has higher priority).

Adding Custom Persistent Highlights

The *`Author API`* includes a class that allows you to create or remove custom persistent highlights, set new properties for the highlights, and customize their appearance. An example of a possible use case would be if you want to implement your own way of editing review comments. The custom persistent highlights get serialized in the XML document as processing instructions, with the following format:

```
<?oxy_custom_start prop1="val1"....?> xml content <?oxy_custom_end?>
```

This functionality is available through the *`AuthorPersistentHighlighter`* class that is accessible through the *`AuthorEditorAccess#getPersistentHighlighter()`* method.

For more information, see the JavaDoc details for this class at *http://www.oxygenxml.com/InstData/Editor/SDK/javadoc/ro/sync/ecss/extensions/api/highlights/AuthorPersistentHighlighter.html*.

Providing Additional Documentation for XML Elements and Attributes

Oxygen XML Editor plugin gathers documentation from the associated schemas (DTD, XML Schema, RelaxNG) and presents it for each element or attribute. For example, if you open the **Content Completion Assistant** for a recognized XML vocabulary, documentation is displayed for each element provided by the associated schema. Similar information is displayed when you hover over tag names presented in the **Elements** view. If you hover over attributes in the **Attributes** view you also see information about each attribute, gathered from the same schema.

If you have a document type configuration set up for your XML vocabulary, there is a special XML configuration file that can be added to provide additional documentation information or links to specification web pages for certain elements and attributes. To provide this additional information, follow these steps:

1. Create a new folder in the configuration directory for the document type.
   `OXYGEN_INSTALL_DIR/frameworks/dita/styleguide`
2. Use the **New** document wizard to create a file using the `Oxygen content completion styleguide` file template.
3. Save the file in the folder created in step 1, using the fixed name: `contentCompletionElementsMap.xml`.
4. *Open the **Preferences** dialog box*, go to **Document Type Association**, and edit the document type configuration for your XML vocabulary. Now you need to indicate where Oxygen XML Editor plugin will locate your mapping file by doing one of the following:
   - In the **Classpath** tab add a link to the newly created folder.
   - In the **Catalogs** tab *add a new catalog file*. The selected file needs to contain the following:

     ```
     <?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">
         <uri name="http://www.oxygenxml.com/{processed_dt_name}/styleguide/contentCompletionElementsMap.xml"
     uri="contentCompletionElementsMap.xml"/>
     </catalog>
     ```

     where *{processed_dt_name}* is the name of the document type in lower case and with spaces replaced by underscores.

     Note: If Oxygen XML Editor plugin finds a mapping file in both locations, the one in the **Catalogs** tab takes precedence.

5. Make the appropriate changes to your custom mapping file.
   You can look at how the DITA mapping file is configured:`OXYGEN_INSTALL_DIR/frameworks/dita/styleguide/contentCompletionElementsMap.xml`
   The associated XML Schema contains additional details about how each element and attribute is used in the mapping file.
6. Re-open the application and open an XML document.

In the **Content Completion Assistant** you should see the additional annotations for each element.

Configuring the Proposals in the Content Completion Assistant

Oxygen XML Editor plugin gathers information from the associated schemas (DTDs, XML Schema, RelaxNG) to determine the proposals that appear in the **Content Completion Assistant**. Oxygen XML Editor plugin also includes support that allows you to configure the possible attribute or element values for the proposals. To do so, a configuration file can be used, along with the associated schema, to add or replace possible values for attributes or elements that are proposed in the **Content Completion Assistant**. An example of a specific use-case is if you want the **Content Completion Assistant** to propose several possible values for the language code whenever you use an `xml:lang` attribute.

To configure content completion proposals, follow these steps:

1. Create a new `resources` folder (if it does not already exist) in the frameworks directory for the document type. For instance: `OXYGEN_INSTALL_DIR/frameworks/dita/resources`.

2. *Open the **Preferences** dialog box* and go to **Document Type Association**. Edit the document type configuration for your XML vocabulary, and in the **Classpath** tab add a link to that `resources` folder.
3. Use the **New** document wizard to create a configuration file using the **Content Completion Configuration** file template.
4. Make the appropriate changes to your custom configuration file. The file template includes details about how each element and attribute is used in the configuration file.
5. Save the file in the `resources` folder, using the fixed name: `cc_value_config.xml`.
6. Re-open the application and open an XML document. In the **Content Completion Assistant** you should see your customizations.

The Configuration File

The configuration file is composed of a series of `match` instructions that will match either an element or an attribute name. A new value is specified inside one or more `item` elements, which are grouped inside an `items` element. The behavior of the `items` element is specified with the help of the `action` attribute, which can have any of the following values:

- `append` - Adds new values to appear in the proposals list (default value).
- `addIfEmpty` - Adds new values to the proposals list, only if no other values are contributed by the schema.
- `replace` - Replaces the values contributed by the schema with new values to appear in the proposals list.

The values in the configuration file can be specified either directly or by calling an external XSLT file that will extract data from any external source.

Example - Specifying Values Directly

```
<!-- Replaces the values for an element with the local name "lg", from the given namespace -->
<match elementName="lg" elementNS="http://www.oxygenxml.com/ns/samples">
    <items action="replace">
        <item value="stanza"/>
        <item value="refrain"/>
    </items>
</match>

<!-- Adds two values for an attribute with the local name "type", from any namespace -->
<match attributeName="type">
    <items>
        <item value="stanza"/>
        <item value="refrain"/>
    </items>
</match>
```

Example - Calling an External XSLT Script

```
<xslt href="../xsl/get_values_from_db.xsl" useCache="false" action="replace"/>
```

In this example, the `get_values_from_db.xsl` is executed in order to extract values from a database.

Note: A comprehensive XSLT sample is included in the **Content Completion Configuration** file template.

Configuring Proposals in the Context for which the Content Completion was Invoked

A more complex scenario for configuring the content completion proposals would be if you want to choose the possible values to provide, depending on the context of the element in which the content completion was invoked.

Suppose that you want to propose certain possible values for one property (for example, *color*) and other values for another property (for example, *shape*). If the property represents a color, then the values should represent applicable colors, while if the property represents a shape, then the values should represent applicable shapes. See the following code snippets:

Your main document:

```
<sampleArticle>
  <!-- The possible values for @value should be "red" and "blue" -->
  <property name="color" value=""/>
  <!-- The possible values for @value should be "square" and "rectangle" -->
  <property name="shape" value=""/>
</sampleArticle>
```

The content completion configuration file:

```
<config xmlns="http://www.oxygenxml.com/ns/ccfilter/config">
    <match elementName="property" attributeName="value">
      <xslt href="get_values.xsl" useCache="false" action="replace"/>
    </match>
</config>
```

The stylesheet that defines the possible values based on the context of the property on which the content completion was invoked:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:saxon="http://saxon.sf.net/"
  exclude-result-prefixes="xs"
  version="2.0">

  <xsl:param name="documentSystemID" as="xs:string"></xsl:param>
  <xsl:param name="contextElementXPathExpression" as="xs:string"></xsl:param>

  <xsl:template name="start">
    <xsl:apply-templates select="doc($documentSystemID)"/>
  </xsl:template>

  <xsl:template match="/">
    <xsl:variable name="propertyElement"
        select="saxon:eval(saxon:expression($contextElementXPathExpression, ./*))"/>

    <items>
      <xsl:if test="$propertyElement/@name = 'color'">
        <item value='red'/>
        <item value='blue'/>
      </xsl:if>
      <xsl:if test="$propertyElement/@name = 'shape'">
        <item value='rectangle'/>
        <item value='square'/>
      </xsl:if>
    </items>
  </xsl:template>
</xsl:stylesheet>
```

The `contextElementXPathExpression` parameter will be bound to an XPath expression that identifies the element in the context for which the content completion was invoked.

Listing of the Example Files - 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

XML Schema file listings.

`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

CSS file listing.

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

XML file listing.

`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

XSL file listing.

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>
```

CSS Support in Author Mode

Author editing mode supports most CSS 2.1 selectors, numerous CSS 2.1 properties, and some CSS 3 selectors. Oxygen XML Editor plugin also supports stylesheets coded with the LESS dynamic stylesheet language. Also, some custom functions and properties that extend the W3C CSS specification, and are useful for URL and string manipulation, are available to developers who create **Author** editing frameworks.

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 your own XML catalog file that maps the location to a custom CSS in the *XML / XML Catalog* preferences page:

```
<uri name="http://host/path/to/location/custom.css" uri="path/to/custom.css"/>
```

Add a Custom Default CSS for Every XML Document

To add a custom CSS that is applied to every XML document, add the following mapping in your XML Catalog file:

```
<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. This allows your custom CSS to be used without the need to modify all other CSS stylesheets contributed in the document type configuration.

Selecting and Combining Multiple CSS Styles

Oxygen XML Editor plugin provides a **Styles** drop-down menu on the **Author Styles** toolbar that allows you to select one *main* (*non-alternate*) CSS style and multiple *alternate* CSS styles. An option in the preferences can be enabled to allow the *alternate* styles to behave like layers and be combined with the *main* CSS style. This makes it easy to change the look of the document.

An example of a common use case is when content authors want to use custom styling within a document. You can select a *main* CSS stylesheet that styles the whole document and then apply *alternate* styles, as layers, to specific parts of the document.

The *main* and *alternate* styles that are listed in the **Styles** drop-down menu can be controlled in the ***Document Type*** *configuration dialog box*. To access it, follow these steps:

1. *Open the* ***Preferences*** *dialog box* .
2. Go to **Document Type Association**.
3. Select the appropriate document type and press the **Edit** button.

The CSS styles (CSS files) associated with the particular document type are listed in the **CSS** subtab of the **Author** tab.

You can **Add**, **Edit**, or **Delete** styles from this dialog box to control the *main* and *alternate* styles associated to the particular document type. You can also change the order of the styles by using the **Move Up** and **Move**

Down buttons. This will also change the order that they appear in the **Styles** drop-down menu. The *alternate* styles are combined with the *main* CSS sequentially, in the order that they appear in this list. Therefore, if the same style rules are included in multiple CSS files, the rules that are defined in the last *alternate* style in this list will take precedence, since it is the last one to be combined (applied as a layer).

The CSS styles (and their order) shown in the following figure will match the styles listed in the **Styles** drop-down menu.

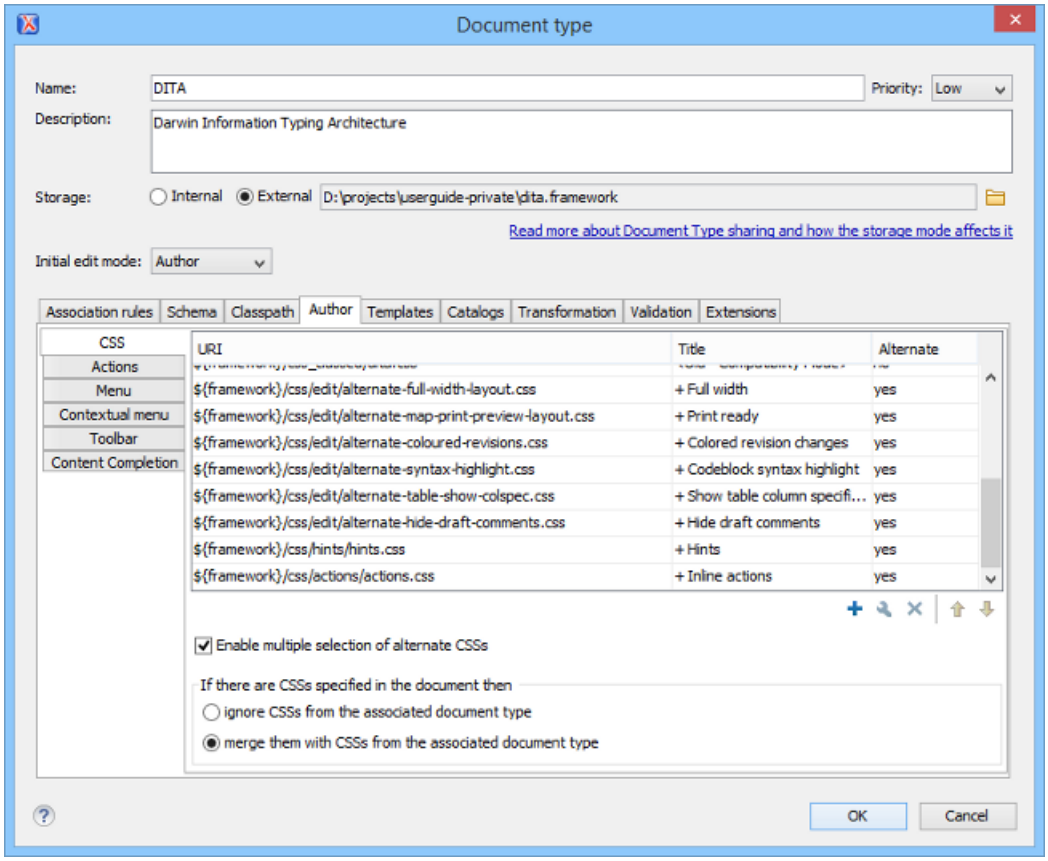


Figure 300: Main and Alternate CSS Styles in the Document Type Association Dialog Box

The **URI** column shows the path of each CSS file. The names listed in the **Styles** drop-down menu match the values in the **Title** column. The value in the **Alternate** column determines whether it is a *main* or *alternate* CSS. If the value is *no* it is a *main* CSS. If the value is *yes* it is an *alternate* CSS and the style can be combined with a *main* CSS or other *alternate* styles when using the **Styles** drop-down menu.

Note: To group alternate styles into categories (submenus), use a vertical bar character (|) in the **Title** column. You can use multiple vertical bars for multiple submenus. The text before each vertical bar will be rendered as the name of a submenu entry in the **Styles** drop-down menu, while the text after the final vertical bar will be rendered as the name of the style inside the submenu.

Example: Suppose that you want to add two alternate stylesheets in separate submenus, with the **Title** column set to `My Styles|User Assistance|Hints` and `My Styles|User Actions|Inline Actions`, respectively. Oxygen XML Editor plugin will add a `My Styles` submenu with two submenus (`User Assistance` that contains the `Hints` style, and `User Actions` that contains the `Inline Actions` style) in the **Styles** drop-down menu.

URI	Title	Alternate
${framework}/css/hints/hints.css	My Styles|User Assistance| Hints	yes
${framework}/css/actions/actions.css	My Styles|User Actions|Inline actions	yes

The **Enable multiple selection of alternate CSSs** box at the bottom of the pane must be checked in order for the *alternate* styles to be combined. They are applied like layers and you can activate any number of them. If this option is disabled, the *alternate* styles are treated like *main* CSS styles and you can only select one at a time. By default, this option is enabled for DITA documents. There are also a few options that allow you to specify how to handle the CSS if there are CSS styles specified in the document. You can choose to *ignore* or *merge* them.

The following rules apply for merging CSS styles:

- CSS files with the same title will be merged.
- CSS files without a title will contribute to all others.
- They are merged sequentially, in the order that they appear in the list.

The selections from the **Styles** drop-down menu are persistent, meaning that Oxygen XML Editor plugin will remember the selections when subsequent documents are opened.

Note: The application also supports working directly with LESS stylesheets, instead of CSS.

CSS Styles in DITA

Oxygen XML Editor plugin comes with a set of predefined CSS layer stylesheets for DITA documents (including maps). In the subsequent figure, a DITA document has the **Century** style selected for the *main* CSS and the *alternate* styles **Full width**, **Show table column specification**, **Hints**, and **Inline actions** are combined for additive styling to specific parts of the document.

Tip: The **Hints** style displays tooltips throughout DITA documents that offer additional information to help you with the DITA structure. The **Inline actions** style displays possible elements that are allowed to be inserted at various locations throughout DITA documents.

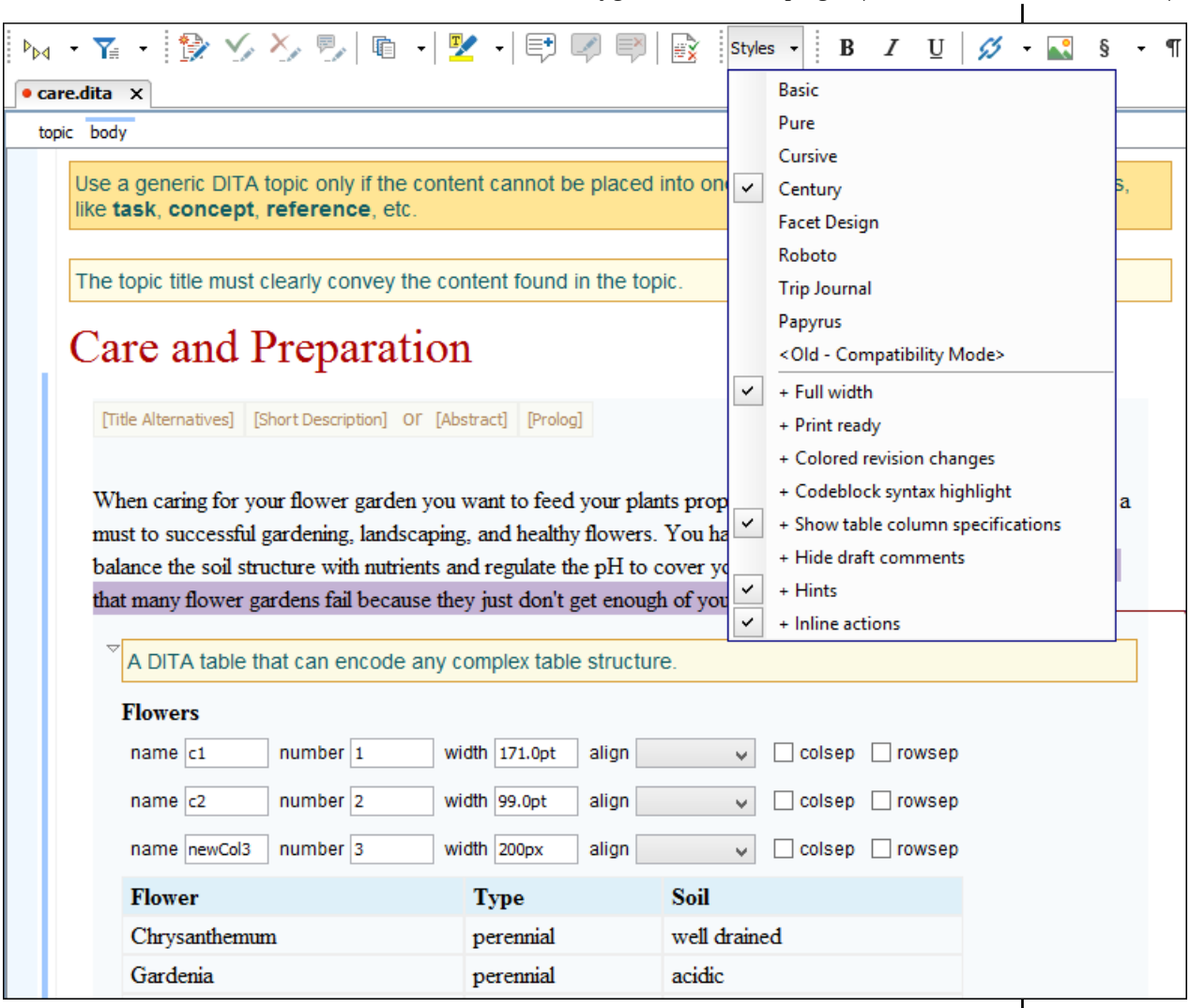


Figure 301: Styles Drop-down Menu in a DITA Document

The oxygen Media Type

The CSS stylesheets can specify how a document is presented on different types of media (on the screen, paper, etc.) You can specify that some of the selectors from your CSS should be taken into account only in the Oxygen XML Editor plugin **Author** mode and ignored in other media types. This can be accomplished by using the oxygen media type.

```
b{
 font-weight:bold;
 display:inline;
}

@media oxygen{
 b{
  text-decoration:underline;
 }
}
```

This example results in the text being bold if the document is opened in a web browser that does not recognize `@media oxygen`, while the text is bold and underlined when opened in Oxygen XML Editor plugin **Author** mode.

You can also use the oxygen media type to specify CSS selectors to be applied in certain operating systems or platforms by using the `os` and `platform` properties. For example, you can specify a set of style rules for displaying Oxygen XML Editor plugin in Windows, and a different set of style rules for Mac OS. The supported properties are as follows:

- **os** - The possible values are: `win`, `linux`, or `mac`.
- **platform** - The possible values are: `standalone` and `eclipse`

```
@media oxygen AND (os:"win") AND (platform:"standalone") {
    p{
        content:"PPP";
    }
}
```

Standard W3C CSS Supported Features

Oxygen XML Editor plugin 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.
`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

Expression	Name	CSS Level	Description / Example
			generated content before or after an element's content.
`E:before(n) and E:after(n)`	Pseudo elements	CSS Level 3	Multiple `':before(n)'` and `':after(n)'` pseudo-elements can be used to insert content before or after the content of an element (or other pseudo-element). For more information, see *the W3C CSS3 pseudo elements site*.
`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:has`	Relational pseudo-class	CSS Level 4	The `:has()` relational pseudo-class is a functional pseudo-class that takes a relative selector as an argument. For more information, see *The :has Relational Pseudo-Class* on page 644.
`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:focus-within`	The generalized input focus pseudo-class	CSS Level 4	The `:focus-within` pseudo-class applies to elements for which the `:focus` pseudo-class applies. Additionally, the ancestors of an element that matches `:focus-within` also match.
`E#myid`	The ID selector	CSS Level 2	Matches any E element with ID equal to "myid". **Important:** Limitation: In Oxygen XML Editor plugin the match is performed taking into account 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`.

Expression	Name	CSS Level	Description / Example
`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 642.
`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 644.

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 Editor plugin 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 Editor plugin supports the subject selector described in CSS Level 4 (currently a working draft at W3C *http://www.w3.org/TR/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
```

The `:has` Relational Pseudo-Class

Oxygen XML Editor plugin supports the CSS Level 4 subject selector (currently a working draft at W3C *http://www.w3.org/TR/selectors4/*), as described in the *Subject Selector* on page 644 topic. Oxygen XML Editor plugin also supports the `:has` relational pseudo-class that has similar functionality and it can match an element by taking its child elements into account. For more information, see *https://drafts.csswg.org/selectors-4/#relational*.

You can create conditions that take into account the structure of the matching element.

For example:

```
table:has( tbody > thead){
 border: 1px solid red;
}
```

This will result in a border being drawn for the table elements that contain at least a `thead` element in the `tbody` element.

Important: The current implementation has a known limitation. The general descendant selectors are taken into account as direct child selectors. For example, the following two CSS selectors are considered equivalent:

```
:has(b c)
```

and

```
:has(> b > c)
```

Supported CSS Properties

Oxygen XML Editor plugin validates all CSS 2.1 properties, but does not render *aural* and *paged* categories properties in **Author** mode, as well as some of the values of the *visual* category that are listed below under the **Ignored Values** column. For the Oxygen XML Editor plugin-specific (extension) CSS properties, go to *Oxygen XML Editor plugin CSS Extensions* on page 653.

Name	Rendered Values	Ignored Values
'background-attachment'	NONE	
'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'	NONE	
'border-collapse'	NONE	
'border-color'	<color> | inherit	transparent
'border-spacing'	NONE	
'border-style'	<border-style> | inherit	
'border-top' 'border-right' 'border-bottom' 'border-left'	[<border-width> || <border-style> || <border-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	

Name	Rendered Values	Ignored Values
'border'	[<border-width> || <border-style> || <border-color>] | inherit	
'bottom'	<length> | <percentage> | inherit	auto
'caption-side'	NONE	
'clear'	NONE	
'clip'	NONE	
'color'	<color> | inherit	
'content'	normal | none | [<string> | <URI> | <counter> | attr(<identifier>) | open-quote | close-quote]+ | inherit	no-open-quote | no-close-quote
'counter-increment'	[<identifier> <integer> ?]+ | none | inherit	
'counter-reset'	[<identifier> <integer> ?]+ | none | inherit	
'cursor'	NONE	
'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'	NONE	
'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'	NONE	
'font-weight'	normal | bold | bolder | lighter | 100 | 200 | 300	

Name	Rendered Values	Ignored Values
	`| 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'`	NONE	
`'left'`	`<length> | <percentage> | inherit`	`auto`
`'letter-spacing'`	NONE	
`'line-height'`	`normal | <number> | <length> | <percentage> | inherit`	
`'list-style-image'`	NONE	
`'list-style-position'`	NONE	
`'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'`	NONE	
`'max-width'`	`<length> | <percentage> | none | inherit` - supported for inline, block-level, and replaced elements (i.e. images, tables, table cells).	
`'min-height'`	Absolute values, such as `230px`, `1in`, `7pt`, `12em`.	Values proportional to the parent element height, such as `30%`.
`'min-width'`	`<length> | <percentage> | inherit` - supported for inline,	

Name	**Rendered Values**	**Ignored Values**
	block-level, and replaced elements, e. g. images, tables, table cells.	
'outline-color'	[<color> | invert | inherit	
'outline-style'	[<border-style> | inherit	
'outline-width'	[<border-width> | inherit	
'outline'	[<outline-width> || <outline-style> || <outline-color>] | inherit	
'overflow'	NONE	
'padding-top' 'padding-right' 'padding-bottom' 'padding-left'	<padding-width> | inherit	
'padding'	<padding-width> | inherit	
'position'	absolute | fixed - supported for block display elements, relative - supported for block and inline display elements	absolute | fixed not supported for inline display elements
'quotes'	NONE	
'right'	<length> | <percentage> | inherit	auto
'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 | inherit	
'text-indent'	<length> | <percentage> | inherit	
'text-transform'	none | capitalize | uppercase | lowercase | inherit	
'top'	<length> | <percentage> | inherit	auto
'unicode-bidi'	bidi-override| normal| embed| inherit	
'vertical-align'	baseline | sub | super | top | text-top | middle | bottom | text-bottom | inherit	<percentage> | <length>

Name	Rendered Values	Ignored Values
`'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 (i.e. images, tables, table cells).	
`'word-spacing'`	NONE	
`'z-index'`	NONE	

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 Editor plugin, the use of `attr()` function is available not only for the `content` property, but also for any other property. This is similar to the CSS Level 3 working draft: *http://www.w3.org/TR/2006/WD-css3-values-20060919/#functional*. The arguments of the function are:

`attr ( attribute_name , attribute_type , default_value )`

`attribute_name`

The attribute name. This argument is required.

`attribute_type`

The attribute type. This argument is optional. If it is missing, argument's type is considered `string`. This argument indicates what is the meaning of the attribute value and helps to perform conversions of this value. Oxygen XML Editor plugin accepts one of the following types:

`color`

The value represents a color. The attribute may specify a color in various formats. Oxygen XML Editor plugin supports colors specified either by name (`red`, `blue`, `green`, etc.) or as an RGB hexadecimal value `#FFEEFF`.

`url`

The value is an URL pointing to a media object. Oxygen XML Editor plugin supports only images. The attribute value can be a complete URL, or a relative one to the XML document. 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);
```

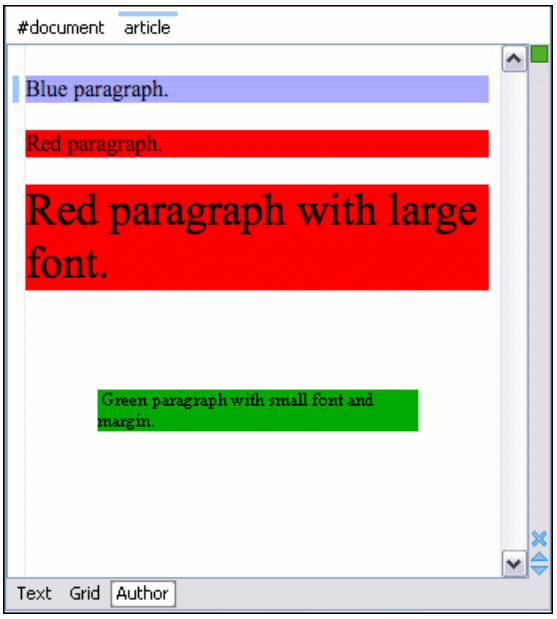
The attribute `font_size` represents the font size in *em* units. You can use this value to change the style of the element:

```
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 Editor plugin supports some of the at-rules specified by CSS Level 2.1 and 3.

The @font-face At-Rule

Oxygen XML Editor plugin 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 specified `font-family` must match the name of the font declared in the `.ttf` file.

The @media Rule

The @media rule allows you to set different style rules for various types of media in the same stylesheet. For example, you can set the font size to be different on the screen than on paper. Oxygen XML Editor plugin supports several media types, allowing you to set the style rules for presenting a document on various media (on screen, paper, etc.)

Supported Media Types

- `screen` - The styles marked with this media type are used only for rendering a document on screen.
- `print` - The styles marked with this media type are used only for printing a document.
- `all` - The styles marked with this media type are used for rendering a document in all supported types of media.
- `oxygen` - The styles marked with this media type are used only for rendering a document in the Oxygen XML Editor plugin **Author** mode. For more information, see *The oxygen Media Type* on page 639 section.
- `oxygen-dark-theme` - The styles marked with this media type are used only for rendering a document in the Oxygen XML Editor plugin **Author** mode when a dark theme is used (for example, *Graphite*).
- `oxygen-high-contrast-black` - The styles marked with this media type are used only for rendering a document in the Oxygen XML Editor plugin **Author** mode on a Windows High Contrast Theme with a black background.
- `oxygen-high-contrast-white` - The styles marked with this media type are used only for rendering a document in the Oxygen XML Editor plugin **Author** mode on a Windows High Contrast Theme with a white background.

```
@media oxygen{
 b{
  text-decoration:underline;
 }
}
@media oxygen-high-contrast-white{
 b{
  font-weight:bold;
 }
}
```

Supported Properties

Oxygen XML Editor plugin also supports a few properties to set specific style rules that depend upon the size of the visible area in **Author** mode. These supported properties are as follows:

- `min-width` - The styles selected in this property are applied if the visible area in **Author** mode is equal to or greater than the specified value.
- `max-width` - The styles selected in this property are applied if the visible area in **Author** mode is less than or equal to the specified value.

```
@media (min-width:500px){
 p{
    content:'XXX';
 }
}
```

```
@media (max-width:700px){
 p:after{
    content:'yyy';
 }
}
```

Oxygen XML Editor plugin CSS Extensions

CSS stylesheets provide support for displaying documents. When editing non-standard documents, Oxygen XML Editor plugin CSS extensions are useful.

Examples of how they can be used:

- Property for marking foldable elements in large files.
- Enforcing a display mode for the XML tags, regardless of the current mode selected by the user.
- Constructing a URL from a relative path location.
- String processing functions.

Additional CSS Selectors

Oxygen XML Editor plugin 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 box*, 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** mode 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;
}
```

To match particular entities, use the `oxy|entity` selector in expressions such as:

```
oxy|entity[name='amp'],
oxy|entity[name='lt'],
oxy|entity[name='gt'],
oxy|entity[name='quot'],
oxy|entity[name='apos'],
oxy|entity[name^='#']{
    -oxy-display-tags: none;
}
```

- 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 text contains *comments* and changes (if *Track Changes* was active when the content was modified).

If this content is referenced, 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:

Additional CSS Properties

Oxygen XML Editor plugin 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 Editor plugin allows you to declare some elements to be *foldable* (collapsible). This is especially useful when working with large documents organized in logical blocks, editing a large DocBook article or book for instance. Oxygen XML Editor plugin marks the foldable content with a small blue triangle. When you hover with your mouse pointer over this marker, a dotted line borders the collapsible content. The following actions are available in the **Folding** submenu of the contextual menu:

Toggle Fold

Toggles the state of the current fold.

Collapse Other Folds (Ctrl NumPad / (Meta NumPad / on OS X))

Folds all the elements except the current element.

Collapse Child Folds (Ctrl + NumPad- (Meta + NumPad- on OS X))

Folds the elements indented with one level inside the current element.

Expand Child Folds (Ctrl NumPad+ (Meta NumPad+ on OS X))

Unfolds all child elements of the currently selected element.

Expand All (Ctrl NumPad * (Meta NumPad * on OS X))

Unfolds all elements in the current document.

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 Editor plugin 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.

The `-oxy-placeholder-content` CSS Property

To control the displayed pseudo-content for empty elements, you can use the `-oxy-placeholder-content` CSS property.

The following example would change the `keyword` element to be displayed as `key`:

```
keyword{
    -oxy-placeholder-content:"key";
}
```

The `-oxy-show-placeholder` CSS Property

The `-oxy-show-placeholder` property allows you to decide whether the placeholder will 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 Editor plugin 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.

For example, suppose we have a **wrapper** XML element that allows 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 Editor plugin 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 Editor plugin 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 cursor 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 Editor plugin 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

Oxygen XML Editor plugin allows you to declare some elements to be *links*. This is especially useful when working with many documents that reference each other. The links allow for an easy way to get from one document to another. Clicking the link marker will open the referenced resource in an editor.

To define the element which should be considered a link, you must use the `link` property 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 Editor plugin 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 uses a `:before` and `:after` element and its tag markers will not 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 Editor plugin does not display element tags. You can use the **Partial Tags** button 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 Editor plugin uses all these parameters to construct an absolute location. Note that any of the parameters that are passed to the function can be either relative or absolute locations. These locations can be expressed as String objects, functions, or editor variables (built-in or custom).

`oxy_url( base_location , loc_1 , loc_2 )`

base_location

String representing the base location. If not absolute, will be solved relative to the CSS file URL.

loc_1 ... loc_n (optional)

Strings representing relative location path components.

The following function receives String objects as input parameters:

```
oxy_url('http://www.oxygenxml.com/css/test.css', '../dir1/', 'dir2/dir3/',
'../../dir4/dir5/test.xml')
```

and returns:

```
'http://www.oxygenxml.com/dir1/dir4/dir5/test.xml'
```

The following function receives the result of the evaluation of two other functions as parameters:

```
image[href]{
 content:oxy_url(oxy_base-uri(), oxy_replace(attr(href), '.jpeg', 'Thumbnail.jpeg'));
}
```

You can use the above example when you have image references and you want to see thumbnail images stored in the same folder.

The following function uses an editor variable as the first parameter to point to the Oxygen XML Editor plugin installation location:

```
image[href] {
    content: oxy_url('${oxygenHome}', 'logo.png');
}
```

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

The `oxy_capitalize` 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 is used to replace a string of text.

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 referenced 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="&quot;"/>` the CSS selector

```
element{
  content:oxy_attributes();
}
```

will display`att1="x" xmlns:a="2" x="""`.

The `oxy_substring()` Function

The `oxy_substring()` function is used to return a string of text.

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 is used to define searches.

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 is used to define last occurrence searches.

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 is used to evaluate XPath expressions.

The `oxy_xpath()` function has the following signature:

- `oxy_xpath ( XPathExpression [, processChangeMarkers , value ] [, evaluate , value ])`

  It evaluates the given XPath 2.0 expression using Saxon 9 and returns the result. XPath expressions that depend on the cursor location can be successfully evaluated only when the cursor is located in the actual XML content. Evaluation fails when the current editing context is inside a referenced **xi:include** section or inside artificially referenced content (for example, DITA `conref` or `topicref` references).

  The parameters of the function are as follows:

  - A required expression parameter, which is the XPath expression to be evaluated.

- An optional `processChangeMarkers` parameter, followed by its value, which can be either `true` or `false` (default value). When you set the parameter to `true`, the function returns the resulting text with all the change markers accepted (*delete* changes are removed and *insert* changes are preserved).
- An optional `evaluate` parameter, followed by its value, which can be one of the following:
  - `dynamic` - Evaluates the XPath each time there are changes in the document.
  - `dynamic-once` - Separately evaluates the XPath for each node that matches the CSS selector. It will not re-evaluate the expression when changes are made to other nodes in the document. This will lead to improved performance, but the displayed content may not be updated to reflect the actual document content.
  - `static` - If the same XPath is evaluated on several nodes, the result for the first evaluation will be used for all other matches. Use this only if the XPath does not contain a relationship with the node on which the CSS property is evaluated. This will lead to improved performance, but the static displayed content may not be updated to reflect the actual document content.

Note: When XPath expressions are evaluated, the entities and `xi:include` elements are replaced with the actual content that is referenced. For example, consider the following code snippet:

```
<article>
  <xi:include href="section1.xml" xmlns:xi="http://www.w3.org/2001/XInclude"/>
</article>
```

where `section1.xml` contains the following content:

```
<section>
  <p>Referenced content</p>
</section>
```

The latter will be the actual content in which the XPath expression is executed.

An Example of the `oxy_xpath()` Function

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(), '')), ' '))",
                processChangeMarkers,
                true),
           "| ");
}
```

Form Controls

Oxygen XML Editor plugin provides a variety of built-in form controls that allow users to interact with documents with familiar user interface objects.

Oxygen XML Editor plugin provides the following built-in form controls:

- *Text Field* - A graphical user interface box that allows you to enter a single line of text.
- *Combo Box* - A graphical user interface object that can be a drop-down menu or a combination of a drop-down menu and a single-line text field.
- *Check Box* - A graphical user interface box that you can click to select or deselect a value.
- *Pop-up* - A contextual menu that provides quick access to various actions.
- *Button* - A graphical user interface object that performs a specific action.
- *Button Group* - A graphical user interface group of buttons (such as radio buttons) that perform specific actions.
- *Text Area* - A box that allows you to enter multiple lines of text.
- *URL Chooser* - A dialog box that allows you to select the location of local or remote resources.
- *Date Picker* - A form control object that allows you to select a date in a specified format.
- *HTML Content* - A graphical user interface box that is used for rendering HTML content.

For customization purposes, Oxygen XML Editor plugin also supports *custom form controls in Java*.

To watch our video demonstration in regards to form controls, go to *http://oxygenxml.com/demo/Form_Controls.html*.

The Text Field Form Control

The `oxy_textfield` built-in form control is used for entering a single line of text in a graphical user interface box. A text field may include optional content completion capabilities, used to present and edit the value of an attribute or an element.

The `oxy_textfield` form control supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
  - **@attribute_name** - The name of the attribute whose 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.
  - **#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 (followed by the `%` character) length units. 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` (default value). To make the form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- `visible` - Specifies whether or not the form control is visible. The possible values of this property are `true` (default value) and `false`.
- `values` - Specifies the values that populate the content completion list of proposals. If these values are not specified in the CSS, they are collected from the associated XML Schema.
- `tooltips` - Associates tooltips to each value in the `values` property. The value of this property is a list of tooltips separated by commas. If you want the tooltip to display a comma, use the `${comma}` variable.
- `tooltip` - Specifies a tooltip to be displayed when you hover over the form control.
- `color` - Specifies the foreground color of the form control. If the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_textfield(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

Text Field Form Control

```
element {
    content: "Label: "
        oxy_textfield(
            edit, "@my_attr",
            values, "value1, value2",
            color, "red",
            columns, 40);
}
```

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_textfield` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_textfield` code template.

The Combo Box Form Control

The `oxy_combobox` built-in form control is used for providing a graphical user interface object that is a drop-down menu of proposed values. This form control can also be used for a combination of a drop-down menu and an editable single-line text field.

The `oxy_combobox` form control supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
  - **@attribute_name** - The name of the attribute whose 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.
  - **#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 (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).
- `visible` - Specifies whether or not the form control is visible. The possible values of this property are `true` (default value) and `false`.
- `editable` - This property accepts the **true** and **false** values. In addition to a drop-down menu, the **true** value also generates an editable text field box that allows you to insert other values than the proposed ones. The **false** value generates a drop-down menu that only accepts the proposed values.
- `tooltips` - Associates tooltips to each value in the `values` property. The value of this property is a list of tooltips separated by commas. If you want the tooltip to display a comma, use the `${comma}` variable.
- `values` - Specifies the values that populate the content completion list of proposals. If 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` (default value). To make the 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 only available for read-only combo boxes (the `editable` property is set to `false`).

- `color` - Specifies the foreground color of the form control. If the value of the `color` property is `inherit`, the form control has the same color as the element in which it is inserted.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_combobox(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

Combo Box Form Control

```
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");
}
```

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_combobox` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_combobox` code template.

The Check Box Form Control

The `oxy_checkbox` built-in form control is used for a graphical user interface box that you can click to enable or disable an option. A single check-box or multiple check-boxes can be used to present and edit the value on an attribute or element.

The `oxy_checkbox` form control supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
  - **@attribute_name** - The name of the attribute whose 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.
  - **#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` - If multiple check-boxes are used, the separator is used to compose the final result. If not specified, the *space* character is used.
- `tooltips` - Associates tooltips to each value in the `values` property. The value of this property is a list of tooltips separated by commas. If you want the tooltip to display a comma, use the `${comma}` variable.
- `visible` - Specifies whether or not the form control is visible. The possible values of this property are `true` (default value) and `false`.
- `values` - Specifies the values that are committed when the check-boxes are selected. If these values are not specified in the CSS, they are collected from the associated XML Schema.

  Note: Typically, when you use a comma in the values of a form control, the content that follows a comma is considered a new value. If you want to include a comma in the values, precede the comma with two backslashes. For example, `oxy_combobox(values, '1\\, 2\\, 3, 4, edit, false)` will display a combo box having the first value `1, 2, 3` and the second value `4`.

- `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` (default value). To make the form control inherit its font from its parent element, set the `fontInherit` property to `true`..
- `uncheckedValues` - Specifies the values that are committed when 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.. If this property is not specified, the `values` property is used as the label.
- `columns` - Controls the width of the form control. The unit size is the width of the **w** character.

- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_checkbox(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

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_checkbox(
        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_checkbox(
        edit, "@attribute",
        values, "true, yes, on",
        uncheckedValues, "false, no, off",
        resultSeparator, ","
        labels, "Present, Working, Started");
}
```

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_checkbox` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_checkbox` code template.

The Pop-up Form Control

The `oxy_popup` built-in form control is used to offer a contextual menu that provides quick access to various actions. A pop-up form control can display single or multiple selections.

The `oxy_popup` form control supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
  - **@attribute_name** - The name of the attribute whose 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.
  - **#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: If the value of the **rows** property is not specified, the default value of *12* is used.

- `color` - Specifies the foreground color of the form control. If 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 or not the form control is visible. The possible values of this property are `true` (default value) and `false`.
- `tooltips` - Associates tooltips to each value in the `values` property. The value of this property is a list of tooltips separated by commas. If 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. If these values are not specified in the CSS, they are collected from the associated XML Schema.

  Note: Typically, when you use a comma in the values of a form control, the content that follows a comma is considered a new value. If you want to include a comma in the values, precede the comma with two backslashes. For example, `oxy_combobox(values, '1\\, 2\\, 3, 4, edit, false)` will display a combo box having the first value `1, 2, 3` and the second value `4`.

- `resultSeparator` - If multiple check-boxes are used, the separator is used to compose the final result. If not specified, the *space* character is used.

  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 multiple values. The predefined values of this property are `single` (default value) and `multiple`.
- `labels` - Specifies the label associated with each entry used for presentation. If this property is not specified, the **values** property is used instead.
- `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 (followed by the `%` character) length units. 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 form control label. The possible values of this property are `ascending` and `descending`.
- `editorSort` - Allows you to sort the values rendered on the form control. The possible values of this property are `ascending` and `descending`.
- `rendererSeparator` - Defines a separator used when multiple values are rendered. If not specified, the value of the `resultSeparator` property is used.
- `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` (default value). To make the form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_popup(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

Pop-up Form Control

```
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",
            color, "blue",
            fontInherit, true);
    font-size:30px;
}
```

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_popup` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_popup` code template.

The Button Form Control

The `oxy_button` built-in form control is used for graphical user interface objects that invokes a custom **Author** mode action (defined in the associated Document Type) referencing it by its ID, or directly in the CSS.

The `oxy_button` form control supports the following properties:

- `actionContext` - Specifies the context in which the action associated with the form control is executed. Its possible values are `element` (default value) 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 cursor location. If the cursor 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` (default value). To make the form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- `color` - Specifies the foreground color of the form control. If 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 the associated *document type framework*, that is invoked when you click the button.

  Note: The element that contains the form control represents the context where the action is invoked.

- `action` - Defines an action directly, rather than using the `actionID` parameter to reference an action from the associated *document type framework*. This property is defined using the *`oxy_action` function*.

```
oxy_button(action, oxy_action(
          name, 'Insert',
          description, 'Insert an element after the current one',
          icon, url('insert.png'),
          operation, 'ro.sync.ecss.extensions.commons.operations.InsertFragmentOperation',
          arg-fragment, '<element>${caret}</element>',
          arg-insertLocation, '.',
          arg-insertPosition, 'After'
)
```

Tip: A code template is available to make it easy to add the `oxy_action` function.

- `visible` - Specifies whether or not the form control is visible. The possible values of this property are `true` (default value) and `false`.
- `transparent` - Flattens the aspect of the button form control, removing its border and background. The values of this property can be `true` or `false` (default value).
- `showText` - Specifies if the action text should be displayed on the button form control. If this property is missing then the button displays the icon only 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 the icon only 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` - To enable *button form controls* or *groups of buttons form controls* this property needs to be set to `true`. This property can be used to specify areas as *read-only* (by setting the `-oxy-editable` property to `false`). This is useful when you want to use an action that does not modify the context.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_button(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

Button Form Control

```
button:before {
    content: "Label:"
        oxy_button(
            /* This action is declared in the document type associated with the XML document.
 */
            actionID, "insert.popupWithMultipleSelection");
}
```

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_button` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_button` code template. Also, an `oxy_button_in_place_action` code template is available that inserts an `oxy_button` function that includes an `action` parameter.

The Button Group Form Control

The `oxy_buttonGroup` built-in form control is used for a graphical user interface group of buttons that invokes one of several custom **Author** mode actions (defined in the associated Document Type) referencing it by its ID, or directly in the CSS.

The `oxy_buttonGroup` form control supports the following properties:

- `actionIDs` - The IDs of the actions that will be presented in the group of buttons.
- `actionID` - The ID of the action, specified in the associated *document type framework*, that is invoked when you click the button.

  Note: The element that contains the form control represents the context where the action is invoked.

- `action_list` - Defines a list of actions directly, rather than using the `actionID` parameter to reference actions from the associated *document type framework*. This property is defined using the *`oxy_action_list` function*.

```
oxy_buttonGroup(
    label, 'A group of actions',
    icon, url('http://www.oxygenxml.com/img/icn_oxy20.png'),
    actions,
      oxy_action_list(
          oxy_action(
                name, 'Insert',
```

```
                description, 'Insert an element after the current one',
                operation, 'ro.sync.ecss.extensions.commons.operations.InsertFragmentOperation',
                arg-fragment, '<element></element>',
                arg-insertLocation, '.',
                arg-insertPosition, 'After'
            ),
            oxy_action(
                name, 'Delete',
                description, 'Deletes the current element',
                operation, 'ro.sync.ecss.extensions.commons.operations.DeleteElementOperation'
            )
        )
    )
```

Tip: A code template is available to make it easy to add the `oxy_action_list` function.

- `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` (default value) 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 cursor location. If the cursor is not inside the element that holds the form control, the `element` value is selected automatically.
- `visible` - Specifies whether or not the form control is visible. The possible values of this property are `true` (default value) and `false`.
- `actionStyle` - Specifies what to display for an action in the form control. The values of this property can be `text` (default value), `icon`, or both.
- `tooltip` - Specifies a tooltip to be displayed when you hover over the form control.
- `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` (default value). To make the form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- `enableInReadOnlyContext` - To enable *button form controls* or *groups of buttons form controls* this property needs to be set to `true`. This property can be used to specify areas as *read-only* (by setting the `-oxy-editable` property to `false`). This is useful when you want to use an action that does not modify the context.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_buttonGroup(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

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");
}
```

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_buttonGroup` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_buttonGroup` code template. Also, an `oxy_buttonGroup_in_place_action` code template is available that inserts an `oxy_buttonGroup` function that includes an `oxy_action_list` function.

The Text Area Form Control

The `oxy_textArea` built-in form control is used for entering multiple lines of text in a graphical user interface box. A text area may include optional syntax highlight capabilities to present the form control.

The `oxy_textArea` form control supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
  - **@attribute_name** - The name of the attribute whose 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.
  - **#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.

  - **#content** - This parameter is useful when an element has mixed or element-only content and you want to edit its content inside a text area form control.

    For example, if you have the following XML content:

    ```
    <codeblock outputclass="language-xml">START_TEXT<ph>phase</ph><apiname><text>API</text></apiname></codeblock>
    ```

    and your CSS includes the following snippet:

    ```
    codeblock:before{
    content:
        oxy_textArea(
          edit, '#content',
          contentType, 'text/xml');
    }
    ```

    then the text area form control will edit the following fragment:

    ```
    START_TEXT<ph>phase</ph><apiname><text>API</text></apiname>
    ```

    Note: When the value of the `edit` property is `#content`, the text area form control will also offer content completion proposals.

- **#content** - This parameter is useful when an element has mixed or element-only content and you want to edit its content inside a text area form control.

  For example, if you have the following XML content:

  ```
  <codeblock outputclass="language-xml">START_TEXT<ph>phase</ph><apiname><text>API</text></apiname></codeblock>
  ```

  and your CSS includes the following snippet:

  ```
  codeblock:before{
  content:
      oxy_textArea(
        edit, '#content',
        contentType, 'text/xml');
  }
  ```

  then the text area form control will edit the following fragment:

  ```
  START_TEXT<ph>phase</ph><apiname><text>API</text></apiname>
  ```

Note: When the value of the `edit` property is `#content`, the text area form control will also offer content completion proposals.

- `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 (followed by the `%` character) length units. 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` (default value). To make the form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- `visible` - Specifies whether or not the form control is visible. The possible values of this property are `true` (default value) and `false`.
- `rows` - This property specifies the number of rows that the form control presents. If 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 highlighting. 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 behavior of the **Tab** key. If the value of this property is set to `true` (default value), 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 whitespaces and new lines of the value are preserved and edited. If the `rows` and `columns` properties are not specifies, the 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 must be specified. If these are not specified, the form control considers the value to be `pre`.

  - `normal` - The white spaces and new lines are normalized.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_textArea(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

The following example presents a text area with CSS syntax highlighting that calculates its own dimension, and a second one with XML syntax highlighting 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);
}
```

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_textArea` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_textArea` code template.

The URL Chooser Form Control

The `oxy_urlChooser` built-in form control is used for a dialog box that allows you to select the location of local or remote resources. The inserted reference is made relative to the URL of the currently opened editor.

The `oxy_urlChooser` editor supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
  - **@attribute_name** - The name of the attribute whose 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.
  - **#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 (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).
- `color` - Specifies the foreground color of the form control. If 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 or not the form control is visible. The possible values of this property are `true` (default value) and `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` (default value). To make the form control inherit its font from its parent element, set the `fontInherit` property to `true`.
- `fileFilter` - string value that holds comma-separated file extensions. The URL chooser uses these extensions to filter the displayed files. A value such as `"jpg,png,gif"` is mapped to three filters that will display all `jpg`, `png`, and `gif` files respectively.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_urlChooser(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

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_urlChooser(
        edit, "@file",
        columns 25);
}
```

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_urlChooser` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_urlChooser` code template.

The Date Picker Form Control

The `oxy_datePicker` built-in form control is used for offering a text field with a calendar browser that allows to choose a certain date in a specified format.

The `oxy_datePicker` form control supports the following properties:

- `edit` - Lets you edit the value of an attribute, the text content of an element, or Processing Instructions (PI). This property can have the following values:
  - **@attribute_name** - The name of the attribute whose 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.
  - **#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 (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).
- `color` - Specifies the foreground color of the form control. If 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 or not the form control is visible. The possible values of this property are `true` (default value) and `false`.
- `validateInput` - Specifies if the form control is validated. If you introduce a date that does not respect the format, the `datePicker` form control is rendered with a red foreground. By default, the input is validated. To disable the validation, set this property to `false`.
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_datePicker(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

Date Picker Form Control

```
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");
}
```

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_datePicker` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_datePicker` code template.

The HTML Content Form Control

The `oxy_htmlContent` built-in form control is used for rendering HTML content. This HTML content is displayed as a graphical element shaped as a box. The shape of the box is determined by a given width and the height is computed based upon the length of the text.

The `oxy_htmlContent` form control supports the following properties:

- `href` - The absolute or relative location of a resource. The resource needs to be a well-formed HTML file.
- `id` - The unique identifier of an item. This is a `div` element that has a unique `id` and is a child of the `body` element. The `div` element is the container of the HTML content to be rendered by the form control.
- `content` - An alternative to the `href` and `id` pair of elements. It provides the HTML content that will be displayed in the form control.
- `width` - Specifies the width of the content area using relative (`em`, `ex`), absolute (`in`, `cm`, `mm`, `pt`, `pc`, `px`), and percentage (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).
- `hoverPseudoclassName` - Allows you to change the way an element is rendered when you hover over a form control. The value is the name of a CSS pseudo-class. When you hover over the form control, the specified pseudo-class will be set on the element that contains the form control.

```
p:before {
  content: oxy_htmlContent(hoverPseudoclassName, 'showBorder')
}
p:showBorder {
  border: 1px solid red;
}
```

You can customize the style of the content using CSS that is either referenced by the file identified by the `href` property or is defined in-line. If you change the HTML content or CSS and you want your changes to be reflected in the XML that renders the form control, then you need to refresh the XML file. If the HTML does not have an associated style, then a default text and background color will be applied.

In the following example, the form control collects the content from the *p_description* `div` element found in the *descriptions.html* file. The box is 400 pixels wide and is displayed before a paragraph identified by the `intro_id` attribute value.

```
p#intro_id:before {
    content:
        oxy_htmlContent(
            href, "descriptions.html",
            id, "p_description",
            width, 400px);
}
```

An alternative example, using the `content` property:

```
p#intro_id:before {
    content:
        oxy_htmlContent(
            content, "<div style='font-weight:bold;'>My content</div>",
```

```
            width, 400px);
}
```

Note: Anchor HTML elements are displayed but the links are inactive.

Note: You can use the **Content Completion Assistant** in the CSS or LESS editor to easily insert a sample of the form control by selecting the corresponding code template. The form control code templates are displayed with a symbol in the content complete list.

Tip: To insert a sample of the `oxy_htmlContent` form control, invoke the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_htmlContent` code template.

Implementing Custom Form Controls

If the built-in form controls are not sufficient for your needs, you can implement custom form controls in Java.

Custom Form Controls Implementation

You can specify custom form controls 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`*.

  Note: If the custom form control is intended to work in the Oxygen XML Editor plugin 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.

- **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** - If 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 XML Editor plugin will perform the commit logic by itself. You can use the **custom** value to perform the commit logic yourself.

The following is a sample Java code for implementing a custom combo box form control that 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);
  }
```

The custom form controls can use any of the predefined properties of the *built-in form controls*, as well as specified custom properties.

This following is an example of how to specify a custom form control in the CSS:

```
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"
    )
}
```

How to Implement Custom Form Controls

To implement a custom form control, follow these steps:

1. Download the Oxygen XML Editor plugin SDK at *http://oxygenxml.com/oxygen_sdk_maven.html*.
2. Implement the custom form control by extending `ro.sync.ecss.extensions.api.editor.InplaceEditorRendererAdapter`. You could also use `ro.sync.ecss.extensions.api.editor.AbstractInplaceEditor`, which offers some default implementations and listeners management.
3. Pack the previous implementation in a Java JAR library.
4. Copy the JAR library to the `[OXYGEN_DIR]/frameworks/[FRAMEWORK_DIR]` directory.
5. In Oxygen XML Editor plugin, *open the **Preferences** dialog box*, go to **Document Type Association**, edit the appropriate framework, and add the JAR library in the **Classpath** tab.
6. Specify the custom form control in your CSS, as described above.

Editing Processing Instructions Using Form Controls

Oxygen XML Editor plugin allows you to edit *processing instructions*, *comments*, and *CDATA* by using the built-in editors.

Oxygen XML Editor plugin allows you to edit *processing instructions*, *comments*, and *CDATA* by 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;
}
```

The `oxy_action()` Function

The `oxy_action()` function allows you to define actions directly in the CSS, rather than referencing them from the associated framework.

The `oxy_action()` function is used from *the `oxy_button()` function*.

The arguments received by the `oxy_action()` function are a list of properties that define an action. The following properties are supported:

- `name` - The name of the action. It will be displayed as the label for the button or menu item.

- `description` (optional) - A short description with details about the result of the action.
- `icon` (optional) - A path relative to the CSS pointing to an image (the icon for the action). The path can point to resources that are packed in Oxygen XML Editor plugin (`oxygen.jar`) by starting its value with `/` (for example, `/images/Remove16.png`). It can also be expressed as *editor variables*.
- `operation` - The name of the Java class implementing the *`ro.sync.ecss.extensions.api.AuthorOperation`* interface. There is also a variety of *predefined operations* that can be used.

  Note: If the name of the operation specified in the CSS is not qualified (has no Java package name), then it is considered to be one of the built-in Oxygen XML Editor plugin operations from `ro.sync.ecss.extensions.commons.operations` package. If the class is not found in this package, then it will be loaded using the specified name.

- `arg-<string>` - All arguments with the `arg-` prefix are passed to the operation (the string that follows the `arg-` prefix is passed).
- `ID` - (optional) - The ID of the action from the framework. If this is specified, all others parameters are disregarded.

```
oxy_button(
   action, oxy_action(
            name, 'Insert',
            description, 'Insert an element after the current one',
            icon, url('insert.png'),
            operation,
             'ro.sync.ecss.extensions.commons.operations.InsertFragmentOperation',
            arg-fragment, '<element>${caret}</element>',
            arg-insertLocation, '.',
            arg-insertPosition, 'After'),
   showIcon, true)
```

Tip: A code template is available to make it easy to add the `oxy_action` function with the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_action` code template..

The `oxy_action_list()` Function

The `oxy_action_list()` function allows you to define a list of actions directly in the CSS, rather than referencing them from the associated framework.

The `oxy_action_list()` function is used from *the `oxy_buttonGroup()` function*.

The arguments received by the `oxy_action_list()` function are a list of actions that are defined with *the `oxy_action()` function*. The following properties are supported in the `oxy_action_list()` function:

- `name` - The name of the action. It will be displayed as the label for the button or menu item.
- `description` (optional) - A short description with details about the result of the action.
- `icon` (optional) - A path relative to the CSS pointing to an image (the icon for the action). The path can point to resources that are packed in Oxygen XML Editor plugin (`oxygen.jar`) by starting its value with `/` (for example, `/images/Remove16.png`). It can also be expressed as *editor variables*.
- `operation` - The name of the Java class implementing the *`ro.sync.ecss.extensions.api.AuthorOperation`* interface. There is also a variety of *predefined operations* that can be used.

  Note: If the name of the operation specified in the CSS is not qualified (has no Java package name), then it is considered to be one of the built-in Oxygen XML Editor plugin operations from `ro.sync.ecss.extensions.commons.operations` package. If the class is not found in this package, then it will be loaded using the specified name.

- `arg-<string>` - All arguments with the `arg-` prefix are passed to the operation (the string that follows the `arg-` prefix is passed).

- `ID` - (optional) - The ID of the action from the framework. If this is specified, all others parameters are disregarded.

```
oxy_action_list(
          oxy_action(
                name, 'Insert',
                description, 'Insert an element after the current one',
              operation, 'ro.sync.ecss.extensions.commons.operations.InsertFragmentOperation',

                arg-fragment, '<element></element>',
                arg-insertLocation, '.',
                arg-insertPosition, 'After'
          ),
          oxy_action(
                name, 'Delete',
                description, 'Deletes the current element',
               operation, 'ro.sync.ecss.extensions.commons.operations.DeleteElementOperation'

          )
      )
```

Tip: A code template is available to make it easy to add the `oxy_action_list` function with the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_action_list` code template.

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 width of the content area using relative (`em`, `ex`), absolute (`in`, `cm`, `mm`, `pt`, `pc`, `px`), and percentage (followed by the `%` character) length units. The `width` property takes precedence over the `columns` property (if the two are used together).
- `color` - Specifies the foreground color of the form control. If 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. If 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`
  - `link` - For more information about this property see *the `link` property section*.

```
element{
    content: oxy_label(text, "Label Text", styles,
      "font-size:2em;color:red;link:attr(href);");
}
```

Instead of using the values of the `styles` property individually, you can define them in a CSS file as in the following example:

```
* {
    width: 40%;
    text-align:center;
}
```

Then refer that file with an *import* directive, as follows:

```
elem {
  content: oxy_label(text, 'my_label', styles, "@import 'labels.css';")
}
```

Caution: Extensive use of the `styles` property may lead to performance issues.

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_label()` function together with a *built-in form control* function to create a form control based layouts.

An example of a use case is if you have multiple attributes on a single element and you want use form controls on separate lines and style them differently. Consider the following CSS rule:

```
person:before {
  content: "Name:*" oxy_textfield(edit, '@name', columns, 20) "\A Address:" oxy_textfield(edit,
 '@address', columns, 20)
}
```

Suppose you only want the **Name** label to be set to **bold**, while you want both labels aligned to look like a table (the first column with labels and the second with a text field). To achieve this, you can use the `oxy_label()` to style each label differently.

```
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)
}
```

Tip: A code template is available to make it easy to add the `oxy_label` function with the **Content Completion Assistant** by pressing **Ctrl Space (Command Space on OS X)** and select the `oxy_label` code template..

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();
}
```

If the text from the target cannot extracted (for instance, if the `href` is not valid), you can use an optional argument to display fallback text.

```
*[class~="map/topicref"]:before{
    content: oxy_link-text("Cannot find the topic reference");
    link:attr(href);
}
```

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

Arithmetic Functions are supported.

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 Editor plugin 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** mode 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 cursor was placed

```
*:caret-visited {
  color:red;
}
```

You could create an *AuthorCaretListener* that sets the `caret-visited` pseudo-class to the element at the cursor location. The effect will be that all the elements traversed by the cursor become red.

The API that you can use from the `CaretListener`:

```
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 ***Author*** *mode operations* can be used directly in your framework to work with custom pseudo-classes:

1. *TogglePseudoClassOperation*
2. *SetPseudoClassOperation*
3. *RemovePseudoClassOperation*

Built in CSS Stylesheet

When Oxygen XML Editor plugin 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 if the CSS developer wants to modify a default behavior.

List of CSS Selector Contributed by Oxygen XML Editor plugin

```
@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:-oxy-morph !important;
    color: rgb(139, 38, 201) !important;
    white-space:pre-wrap !important;
    border-width:0px !important;
    margin:0px !important;
    padding: 0px !important;
}

oxy|processing-instruction[Pub],
oxy|processing-instruction[PubTbl],
oxy|processing-instruction[xm-replace_text],
oxy|processing-instruction[xm-deletion_mark],
oxy|processing-instruction[xm-insertion_mark_start],
oxy|processing-instruction[xm-insertion_mark_end],
oxy|processing-instruction[xml-model],
oxy|processing-instruction[xml-stylesheet]
{
    display:none !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;
  margin-right:0px !important;
  padding-right:2px !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[name='amp'],
oxy|entity[name='lt'],
oxy|entity[name='gt'],
oxy|entity[name='quot'],
oxy|entity[name='apos'],
oxy|entity[name^='#']{
    /*EXM-32236 Do not present tags for character entity references.*/
    -oxy-display-tags: none;
}

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.png) attr(href)  !important;
    link: attr(href) !important;
}
xi|include[parse="text"]:before {
    content:url(../images/link.png) !important;
}
xi|include[xpointer]:before {
    content:url(../images/link.png) 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;
}

@media oxygen-high-contrast-black, oxygen-dark-theme{
    oxy|doctype {
        color:#D0E2F4 !important;
    }
}

oxy|error {
    display:-oxy-morph !important;
    -oxy-editable:false !important;
    white-space:pre !important;
    font-weight:bold !important;
    color: rgb(178, 0, 0) !important;
    -oxy-display-tags: none;
}

oxy|error:before {
    content:url(../images/ReferenceError.png) "[" !important;
    color: rgb(178, 0, 0) !important;
}
oxy|error[level='warn']:before {
    content:url(../images/ReferenceWarn.png) "[" !important;
    color: rgb(200, 185, 0) !important;
}

oxy|error[level='warn'] {
    color: rgb(200, 185, 0) !important;
}

oxy|error:after {
    content:"]" !important;
}


*[xlink|href]:before {
    content:url(../images/link.png);
    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 !important;
}

svg|svg * {
```

```
    display:none !important;
    white-space:normal !important;
}

mml|math{
 display:inline !important;
    white-space: -oxy-trim-when-ws-only !important;
}
mml|math mml|*{
    display:none !important;
    white-space: normal !important;
}

/*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; }

@media oxygen-high-contrast-black, oxygen-dark-theme{
    xi|include:before,
    xi|include:after{
        color:#808080 !important;
    }
}
```

To show all entities in the **Author** mode as transparent, without a gray background, first define in your CSS after all imports the namespace:

```
@namespace oxy "http://www.oxygenxml.com/extensions/author";
```

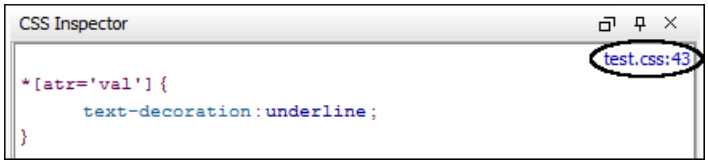
and then add the following selector:

```
oxy|entity {
    background-color: inherit !important;
}
```

Debugging CSS Stylesheets

To assist you with debugging and customizing CSS stylesheets the **Author** mode includes a ***CSS Inspector** view* to examine the CSS rules that match the currently selected element.

This tool is similar to the Inspect Element development tool that is found in most browsers. The **CSS Inspector** view allows you to see how the CSS rules are applied and the properties defined. Each rule that is displayed in this view includes a link to the line in the CSS file that defines the styles for the element that matches the rule. You can use the link to open the appropriate CSS file and edit the style rules. Once you have found the rule you want to edit, you can click the link in the top-right corner of that rule to open the CSS file in the editor.



There are two ways to open the CSS Inspector view:

1. Select **CSS Inspector** from the **Window** > **Show View** menu.
2. Select the **Inspect Styles** action from the contextual menu in **Author** mode.

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 Editor plugin standalone installation 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`.

To develop `JUnit` tests for your customizations using the **Eclipse** workbench, follow these steps:

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());
    }
}
```

API Frequently Asked Questions (API FAQ)

This section contains answers to common questions regarding the Oxygen XML Editor plugin customizations 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, if 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 on 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 customizing the application are possible:

1. Implement a plugin.

   A plugin serves a general purpose and influences any type of XML file that you open in Oxygen XML Editor plugin.

   For the Oxygen XML Editor pluginPlugins API, Javadoc, samples, and documentation, go to *http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins*

2. *Create or modify the document type* that is associated to your specific XML vocabulary.

   This document type can be used, for instance, 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 end users are editing DITA documents, all the toolbar actions that are specific for DITA are provided by the DITA Document Type. If you look in the ***Document Type Association** preferences page* there is a *DITA* document type. If you edit that document type you will see that it has an **Author** tab in the ***Document Type** configuration dialog box*. The subtabs in this tab can be used to define custom DITA actions and add them to the toolbars, main menus, or contextual menus.

   For information about developing your own document types (frameworks), see the *Author Mode Customization Guide* on page 552 chapter.

   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 ***Document Type Association** preferences page*. Then you can share that updated file with all users.

   The same folder contains some JAR libraries. These libraries contain custom Java operations that are called when the user presses certain toolbar actions.

   We have an *oXygen SDK* that contains the Java sources from all the DITA Java customizations:

   http://www.oxygenxml.com/oxygen_sdk.html#XML_Editor_Authoring_SDK

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 document in **Author** mode. 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 behavior, edit your document type and in the ***Document Type** configuration dialog box* go to the **Author** tab, then **Actions** subtab, and 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 cursor 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 Editor plugin 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 *customize 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 document content in **Author** mode?

Answer

There are two types of highlights you can add:

1. **Non-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* that allows you to add a highlight between certain offsets with a specified painter.

   For example, you can use this support to implement your own spell checker with a custom highlight for the unrecognized words.

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* class that allows you to add a persistent highlight between certain offsets, set new properties for a specific highlight, and render it with a specified 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 **Text** modes. The API is available both in the standalone application and in the Eclipse plugin.

Here is an elegant way to add actions to the **Author** page from your Eclipse plugin extension:

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>
```

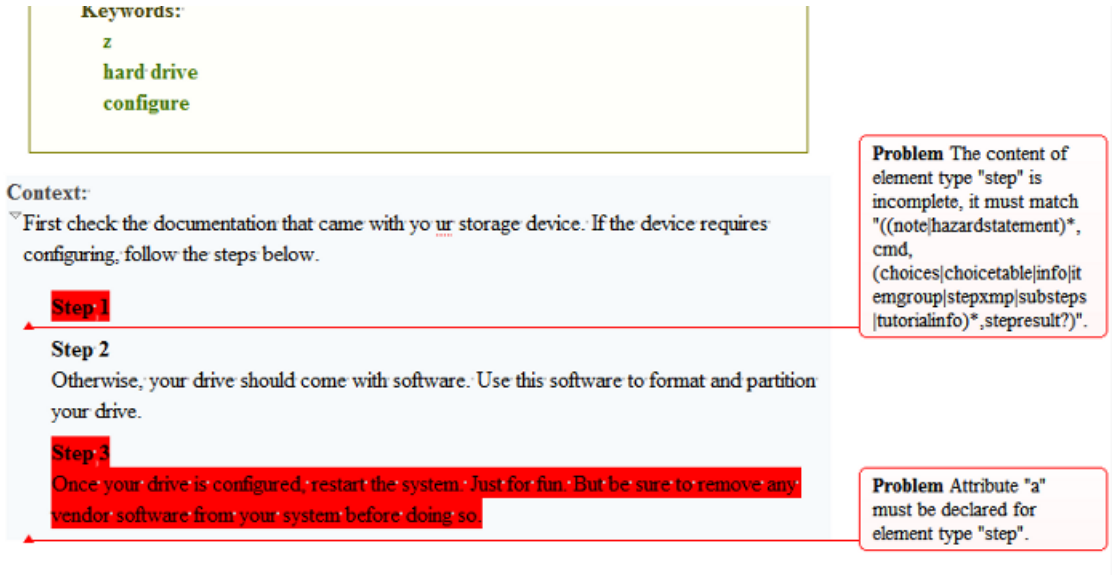
Adding Custom Callouts

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 XML Editor plugin 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** mode 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** mode.

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** mode actions depending on the cursor context?

Answer

By default, Oxygen XML Editor plugin does not toggle the enabled/disabled states for actions based on whether the activation XPath expressions for that certain **Author** mode action are fulfilled. This is done because the actions can be many and evaluating XPath expression on each cursor move can lead to performance problems. However, 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 cursor 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 `CaretListener` in order to avoid adding multiple listeners that perform the same functionality.

Dynamic Open File in Oxygen XML Editor plugin Distributed via JavaWebStart

Question

How can we dynamically open a file in an Oxygen XML Editor plugin distributed via JWS?

Answer

The JWS packager Ant build file which comes with Oxygen XML Editor plugin 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 is required by newer Java versions and means that it is impossible to automatically generate a JNLP file containing some dynamic arguments. The solution is to use the signed JNLP template feature of Java 7, bundle inside the JAR library a signed `APPLICATION_TEMPLATE.JNLP` instead of an `APPLICATION.JNLP` with a wildcard command line argument:

```
<application-desc main-class="ro.sync.jws.JwsDeployer">
   <argument>*</argument>
</application-desc>
```

Then you can replace the wildcard in the external placed JNLP to the actual, dynamic command line arguments value.

A different approach (more complicated though) would be to have the JNLP file signed and always referenced 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 Editor plugin 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. *Customize 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 Document in Author Mode

Question

How can we add multiple buttons, each showing different visualisation mode of the same document in **Author** mode (by associating additional/different CSS style sheet)?

Answer

In the toolbar of the **Author** mode there is a **Styles** drop-down menu that contains alternative CSS styles for the same document. To add an alternative CSS stylesheet, *open the **Preferences** dialog box*, go to **Document Type Association**, select the document type associated with your documents and press **Edit**. In the ***Document Type** configuration dialog box* that appears, go to the **Author** tab, and in the **CSS** subtab add references to alternate CSS stylesheets.

For example, one of the alternate CSS stylesheets that we offer for the `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 documentation that explains this 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 document content to the printer or to show the Print Preview dialog box, 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 Editor plugin 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** mode 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 am inserting a DITA *image* XML element using the API that 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) that 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 cursor 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 cursor 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 Editor plugin.

   Note: The extracted folder name depends on which product variant you have downloaded. For the purpose of this procedure the folder will be referred to as `[OXYGEN_DIR]`.

2. Set up the *oXygen SDK* following *this set of instructions*.
3. Create an Eclipse Java Project (for example, `MyPluginProject`) from one of the sample plugins (the Workspace Access plugin for example).
4. In the `MyPluginProject` folder, create a folder called `myPlugin`. In this new folder copy the `plugin.xml` from the sample plugin. Modify the added `plugin.xml` to add a library reference to the directory where Eclipse copies the compiled output. To find out where this directory is located, invoke the contextual menu of the project (in the **Project** view), and go to **Build Path** > **Configure Build Path**. Then inspect the value of the **Default output folder** text box.

   Example: If the compiled output folder is `classes`, then the you need to add in the `plugin.xml` the following library reference:

   ```
   <library name="../classes"/>
   ```

5. Copy the `plugin.dtd` from the `[OXYGEN_DIR]/plugins` folder in the root `MyPluginProject` folder.
6. In the `MyPluginProject`'s build path add external JAR references to all the JAR libraries in the `[OXYGEN_DIR]/lib` folder. Now your `MyPluginProject` should compile successfully.
7. In the Eclipse IDE, create a new *Java Application* configuration for debugging. Set the **Main class** box to `ro.sync.exml.Oxygen`. Click the **Arguments** tab and add the following code snippet in the **VM arguments** input box, making sure that the path to the plugins directory is the correct one:

   ```
   -Dcom.oxygenxml.app.descriptor=ro.sync.exml.EditorFrameDescriptor -Xmx1024m
   -XX:MaxPermSize=384m -Dcom.oxygenxml.editor.plugins.dir=D:\projects\MyPluginProject
   ```

   Note: If you need to configure the plugin for , set the `com.oxygenxml.app.descriptor` to `ro.sync.exml.AuthorFrameDescriptor` or `ro.sync.exml.DeveloperFrameDescriptor`, respectively.

8. Add a break point in the source of one of your 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 Editor plugin to a folder on your hard drive.

   Note: Name the folder `[OXYGEN_DIR]`.

2. Create an Eclipse Java Project (for example, `MySDKProject`) with the corresponding Java sources (for example, a custom implementation of the `ro.sync.ecss.extensions.api.StylesFilter` interface).
3. In the Project 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 XML Editor plugin from the `[OXYGEN_DIR]` and in the ***Document Type Association*** *preferences page*, edit the document type (for example **DITA**) to open the ***Document Type*** *configuration dialog box*. 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 a DocBook 4 `xref` displays in **Author** mode based on what element is at the `linkend`?

Follow these steps:

1. Create a Maven Java project and add a dependency on the Oxygen XML Editor plugin classes:

```
<dependency>
    <groupId>com.oxygenxml</groupId>
    <artifactId>oxygen-sdk</artifactId>
    <version>${oxygen.version}</version>
</dependency>
```

   where `${oxygen.version}` is the version of Oxygen XML Editor plugin.

   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 the `[OXYGEN_DIR]\frameworks\docbook\docbook.jar` to the class path of the project.
3. Create a class that 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 *oXygen SDK*).
5. Pack your extension classes in a JAR file. Copy the JAR to: `[OXYGEN_DIR]\frameworks\docbook\custom.jar`.
6. Start Oxygen XML Editor plugin.
7. *Open the* ***Preferences*** *dialog box* 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 the `docbook` framework folder to something else (such as `custom_docbook`) and share it with others.

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 Editor plugin, 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?

Suppose that 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 cursor 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 CaretListener when we select from the CaretListener.
   */
  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 one of the *built-in form controls*. However, in some cases you do not 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, if you want a combo box form control and the values to populate the combo are specified inside a file (or 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 *oXygen 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

The 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. Add a listener that 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. An API to open XML documents in the application:

```
ro.sync.exml.workspace.api.Workspace.open(URL)
```

So you can create up a plugin that 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 XML Editor plugin to update the revised date on a DITA document when it's saved?

Answer

The 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 to get Oxygen XML Editor plugin to automatically generate a file name comprising a UUID plus file extension using the SDK?

Answer

This could be done implementing a plugin for Oxygen XML Editor plugin using the Plugins SDK:

http://www.oxygenxml.com/oxygen_sdk.html#Developer_Plugins

There is a type of plugin called *Workspace Access* that 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 display an alternative chooser dialog box, 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 Editor plugin using the 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 Editor plugin Distribution

Question

Is it possible to create a custom view in Eclipse that can insert certain XML fragments in the documents opened with the Oxygen XML Editor plugin?

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 Editor plugin 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 cursor position
//        authPage.getDocumentController().insertXMLFragment("<testTag/>", authPage.getCaretOffset());
//      } catch (AuthorOperationException e) {
//        // TODO Auto-generated catch block
//        e.printStackTrace();
//      }
    }
```

Chapter 10

Transforming Documents

Topics:

XML documents can be transformed into a variety of user-friendly output formats that can be viewed by other users. This process is known as a *transformation*.

Transformation Scenarios

A transformation scenario is a set of complex operations and settings that gives you the possibility to obtain outputs of multiple types (XML, HTML, PDF, EPUB, etc.) from the same source of XML files and stylesheets.

Executing a transformation scenario implies multiple actions, such as:

- Validating the input file.
- Obtaining intermediate output files (for example, formatting objects for the XML to PDF transformation).
- Using transformation engines to produce the output.

Before transforming an XML document in Oxygen XML Editor plugin, you need to define a transformation scenario to apply to that document. A scenario is a set of values for various parameters that define a transformation. It is not related to a particular document, but rather to a document type. Types of transformation scenarios include:

- **Scenarios that Apply to XML Files** - This type of scenario contains the location of an XSLT stylesheet that is applied on the edited XML document, as well as other transformation parameters.
- **Scenarios that Apply to XSLT Files** - This type of scenario contains the location of an XML document, on which the edited XSLT stylesheet is applied, as well as other transform parameters.
- **Scenarios that Apply to XQuery Files** - This type of scenario contains the location of an XML source, on which the edited XQuery file is applied, as well as other transform parameters. When the XML source is a native XML database, the XML source field of the scenario is empty because the XML data is read with XQuery-specific functions, such as `document()`. When the XML source is a local XML file, the URL of the file is specified in the XML input field of the scenario.
- **Scenarios that Apply to SQL Files** - This type of scenario specifies a database connection for the database server that runs the SQL file that is associated with the scenario. The data processed by the SQL script is located in the database.
- **Scenarios that Apply to XProc Files** - This type of scenario contains the location of an XProc script, as well as other transform parameters.
- **DITA-OT Scenarios** - This type of scenario provides the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Editor plugin comes with a built-in version of Ant and a built-in version of DITA-OT, although you can also set other versions in the scenario.
- **ANT Scenarios** - This type of scenario contains the location of an Ant build script, as well as other transform parameters.

Note:

Status messages generated during the transformation process are displayed in the ***Console** view*.

Defining a New Transformation Scenario

Defining a transformation scenario is the first step in the process of transforming a document. The following types of scenarios are available:

- ***XML Transformation with XSLT*** - Specifies the transformation parameters and location of an XSLT stylesheet that is applied to the edited XML document. This scenario is useful when you develop an XML document and the XSLT document is in its final form.
- ***XML Transformation with XQuery*** - Specifies the transform parameters and location of an XQuery file that is applied to the edited XML document.
- ***DITA-OT Transformation*** - Specifies the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Editor plugin comes with a built-in version of Ant and a built-in version of DITA-OT but different versions can be set in the scenario.
- ***ANT Transformation*** - Allows you to configure the options and parameters of an Ant build script.
- ***XSLT Transformation*** - Specifies the transformation parameters and location of an XML document to which the edited XSLT stylesheet is applied. This scenario is useful when you develop an XSLT document and the XML document is in its final form.
- ***XProc Transformation*** - Specified the transformation parameters and location of an XProc script.

- ***XQuery Transformation*** -Specifies the transformation parameters and location of an XML source to which the edited XQuery file is applied. When the XML source is a native XML database, the XML source field of the scenario is empty because the XML data is read with XQuery-specific functions, such as `document()`. When the XML source is a local XML file, the URL of the file is specified in the XML input field of the scenario.
- ***SQL Transformation*** - Specifies a database connection for the database server that runs the SQL file associated with the scenario. The data processed by the SQL script is located in the database.

XML Transformation with XSLT

To create an **XML transformation with XSLT** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XML transformation with XSLT**.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XML transformation with XSLT**.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XML transformation with XSLT**.

  Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog box allows you to configure the options that control the transformation.

The dialog box contains the following tabs:

- ***XSLT***.
- ***FO Processors***.
- ***Output***.

The XSLT Tab

The **XSLT** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  Note: If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

  Note: If the transformer engine is one of the built-in XSLT 2.0 / 3.0 engines and *the name of an initial template* is specified in the scenario, the **XML URL** field can be empty. The **XML URL** field can also be empty if you use *external XSLT processors*. Otherwise, a value is mandatory in the **XML URL** field.

- **XSL URL** - Specifies the source XSL file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XSL URL** fields:

Insert Editor Variables

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse workspace

Opens a file browser dialog box allowing you to select a file from the local workspace.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the ***Data Source Explorer*** window.

Search for file

Allows you to find a file in the current project.

The rest of the options available in the **XSLT** tab allow you to further customize the transformation scenario:

- **Use "xml-stylesheet" declaration** - Use the stylesheet declared with an `xml-stylesheet` declaration instead of the stylesheet specified in the **XSL URL** field. By default, this checkbox is not selected and the transformation applies the XSLT stylesheet that is specified in the **XSL URL** field. If it is checked, the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction.
- **Transformer** - This drop-down menu presents all the transformation engines available to Oxygen XML Editor plugin for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog box is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).
  - **Advanced options** - Allows you to configure the advanced options of the Saxon HE / PE / EE engine for the current transformation scenario. To configure the same options globally, go to the *Saxon-HE/PE/EE preferences page*. For the current transformation scenario, these **Advanced options** override the options configured in the *Saxon-HE/PE/EE preferences page*. The **Initial mode and template** option is only available in the **Advanced options**. It is a Saxon-specific option that sets the name of the first XSLT template that starts the XSLT transformation or the initial mode of the transformation.
- **Parameters** - Opens *the **Configure parameters** dialog box*, allowing you to configure the XSLT parameters used in the current transformation. In this dialog box, you can also configure the parameters of additional stylesheets by using the **Additional XSLT stylesheets** button. If the XSLT transformation engine is custom-defined, you can not use this dialog box to configure the parameters sent to the custom engine. Instead, you can copy all parameters from the dialog box using contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line that starts the transformation process.
- **Extensions** - Opens *the dialog box for configuring the XSLT/XQuery extension jars or classes* that define extension Java functions or extension XSLT elements used in the transformation.
- **Additional XSLT stylesheets** - Opens *the dialog box for adding XSLT stylesheets* that are applied on the main stylesheet that is specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document.

Configure XSLT Parameters

The global parameters of the XSLT stylesheet used in a transformation scenario can be configured by using the **Parameters** button in the **XSLT** tab of a new or edited transformation scenario dialog box.

The table displays all the parameters of the current XSLT stylesheet, all imported and included stylesheets, and all *additional stylesheets*, along with their descriptions and current values. You can also add, edit, and remove parameters. Use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

If the **XPath** column is checked, the parameter value is evaluated as an XPath expression before starting the XSLT transformation.

For example, you can use expressions such as:

```
doc('test.xml')//entry
//person[@atr='val']
```

Note:

1. The **doc** function solves the argument relative to the XSL stylesheet location. You can use full paths or editor variables (such as *${cfdu}* [current file directory]) to specify other locations:
   `doc('${cfdu}/test.xml')//*`
2. You cannot use XSLT Functions. Only XPath functions are allowed.

The following actions are available for managing the parameters:

New

Opens the **Add Parameter** dialog box that allows you to add a new parameter to the list. An *editor variable* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is enabled, the parameter will be evaluated as an XPath expression.

Edit

Opens the **Edit Parameter** dialog box that allows you to edit the selected parameter. An *editor variable* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is enabled, the parameter will be evaluated as an XPath expression.

Unset

Resets the selected parameter to its default value. Available only for edited parameters with set values.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The bottom panel presents the following:

- The default value of the parameter selected in the table.
- A description of the parameter, if available.
- The system ID of the stylesheet that declares it.

XSLT/XQuery Extensions

The **Libraries** dialog box is used to specify the jars and classes that contain extension functions called from the XSLT or XQuery file of the current transformation scenario. You can use the **Add**, **Edit**, and **Remove** buttons to configure the extensions.

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 displayed in this dialog box. To change the order of the items, select the item to be moved and press the **Move up** or **Move down** buttons.

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.
- **XSLT result as input** - The FO processor is applied to the result of the XSLT transformation that is defined in the **XSLT** tab.
- **XML URL as input** - The FO processor is applied to the input XML file.
- **Method** - The output format of the FO processing. Available options depend on the selected processor type.
- **Processor** - Specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Prompt for file** - At the end of the transformation, a file browser dialog box is displayed for specifying the path and name of the file that stores the transformation result.
- **Save As** - The path of the file where the result of the transformation is stored. The path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, `.pdf` files are usually opened in the *Acrobat Reader* application).

  Note: To set the web browser that is used for displaying HTML/XHTML pages, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**.

  - **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin automatically opens the file specified in the **Save As** text field at the end of the transformation.
  - **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor plugin opens the file specified here. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.

- **Open in editor** - When this is enabled, the transformation result specified in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).
- **Show in results view as**
  - **XHTML** - Can only be enabled if **Open in Browser/System Application** is disabled. If this is checked, Oxygen XML Editor plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

    Important: When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you want to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

  - **XML** - If this is checked, Oxygen XML Editor plugin displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlighting*, specific for XML documents.
- **Image URLs are relative to** - If **Show in results view as XHTML** is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

The Oxygen XML Editor plugin Browser View

The Oxygen XML Editor plugin Browser view is automatically displayed in the views pane of the Eclipse window to display HTML output from XSLT transformations. It contains a tab for each file with HTML results displayed in the view.

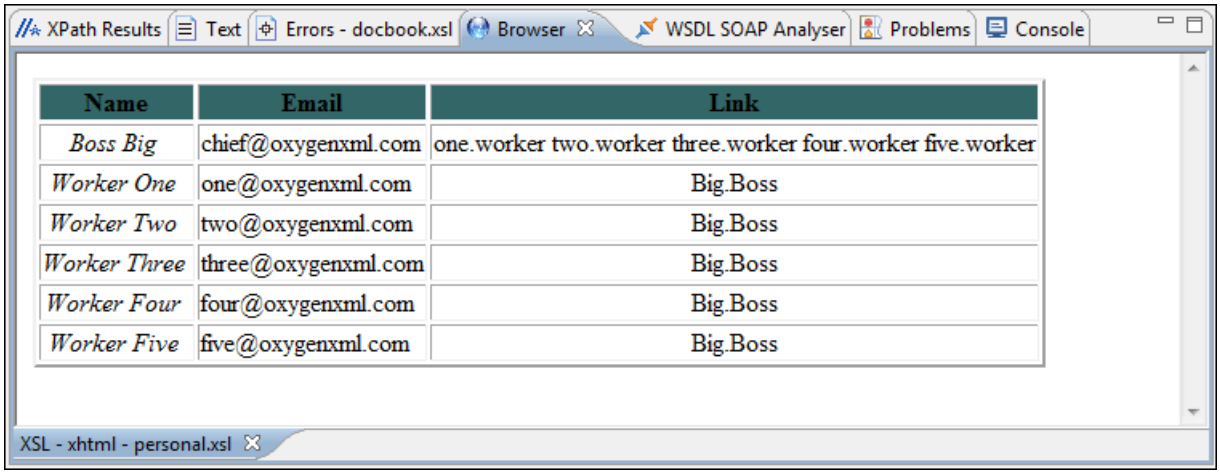


Figure 302: The Browser View

The Oxygen XML Editor plugin Text View

The Oxygen XML Editor plugin **Text** view is automatically displayed in the views pane of the Eclipse window to display text output from XSLT transformations, FO processor info, warnings, and error messages. It contains a tab for each file with text results displayed in the view.

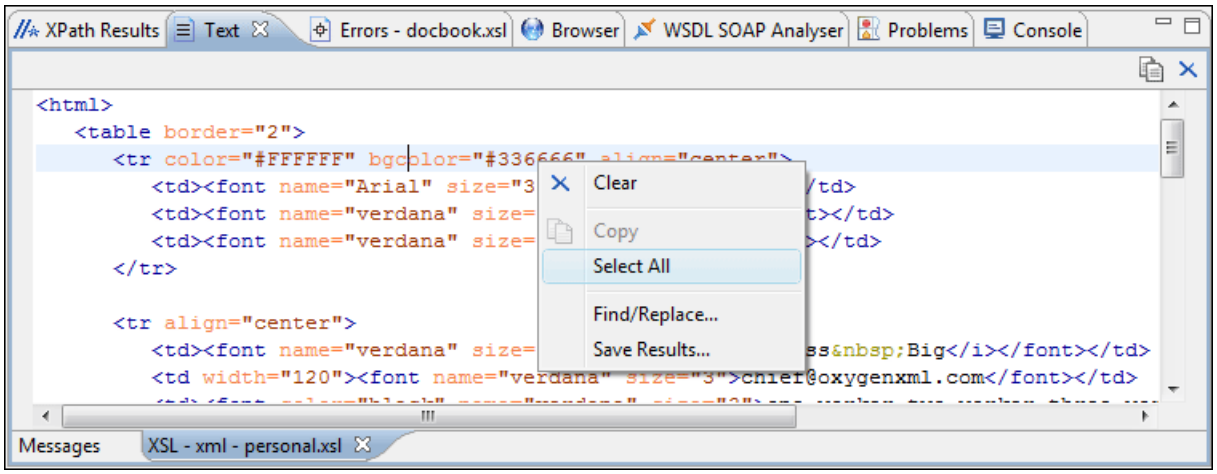


Figure 303: The Text View

Additional XSLT Stylesheets

The list of additional XSLT stylesheets can be edited in the dialog box opened by the **Additional XSLT Stylesheets** button in the **XSLT** tab of a new or edited transformation scenario dialog box. The following actions are available:

Add

Adds a stylesheet in the **Additional XSLT stylesheets** list using a file browser dialog box. You can type an *editor variable* in the file name field of the browser dialog box. The name of the stylesheet will be added in the list after the current selection.

Remove

Deletes the selected stylesheet from the **Additional XSLT stylesheets** list.

Up

Moves the selected stylesheet up in the list.

Down

Moves the selected stylesheet down in the list.

XML Transformation with XQuery

Use the **XML transformation with XQuery** scenario to apply a transformation in which an XQuery file queries an XML file for the output results.

To create an **XML transformation with XQuery** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XML transformation with XQuery**.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XML transformation with XQuery**.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XML transformation with XQuery**.

  Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog box allows you to configure the options that control the transformation.

The dialog box contains the following tabs:

- ***XQuery***.
- ***FO Processor***.
- ***Output***.

The XQuery Tab

The **XQuery** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  Note: If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

- **XQuery URL** - specifies the source XQuery file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XQuery URL** fields:

Insert Editor Variables

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse workspace

Opens a file browser dialog box allowing you to select a file from the local workspace.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the ***Data Source Explorer*** window.

Search for file

Allows you to find a file in the current project.

The rest of the options available in the **XQuery** tab allow you to further customize the transformation scenario:

- **Transformer** - This drop-down menu presents all the transformation engines available to Oxygen XML Editor plugin for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog box is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).;
  - **Advanced options** - *configure advanced options specific for the Saxon HE / PE / EE engine*.
- **Parameters** - Opens the **Configure parameters** dialog box for configuring the XQuery parameters. You can use buttons in this dialog box you can add, edit, or remove parameters. If the XQuery transformation engine is custom-defined you can not use this dialog box to set parameters. Instead, you can copy all parameters from the dialog box using contextual menu actions and edit the custom XQuery engine to include the necessary parameters in the command line that starts the transformation process.

  Note: Use the **Filter** text box to search for a specific term in the entire parameters collection.

- **Extensions** - Opens *the dialog box for configuring the XSLT/XQuery extension jars or classes* that define extension Java functions or extension XSLT elements used in the transformation.

XSLT/XQuery Extensions

The **Libraries** dialog box is used to specify the jars and classes that contain extension functions called from the XSLT or XQuery file of the current transformation scenario. You can use the **Add**, **Edit**, and **Remove** buttons to configure the extensions.

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 displayed in this dialog box. To change the order of the items, select the item to be moved and press the **Move up** or **Move down** buttons.

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.
- **XQuery result as input** - the FO processor is applied to the result of the XQuery transformation defined in the **XQuery** tab.
- **XML URL as input** - The FO processor is applied to the input XML file.
- **Method** - The output format of the FO processing. Available options depend on the selected processor type.
- **Processor** - Specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Present as a sequence** - Enabling this option will reduce the time necessary to fetch the full result, as it will only fetch the first chunk of the result.
- **Prompt for file** - At the end of the transformation, a file browser dialog box is displayed for specifying the path and name of the file that stores the transformation result.
- **Save As** - The path of the file where the result of the transformation is stored. The path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.

- **Open in Browser/System Application** - If enabled, Oxygen XML Editor plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, `.pdf` files are usually opened in the *Acrobat Reader* application).

  Note: To set the web browser that is used for displaying HTML/XHTML pages, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**.

  - **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin automatically opens the file specified in the **Save As** text field at the end of the transformation.
  - **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor plugin opens the file specified here. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.

- **Open in editor** - When this is enabled, the transformation result specified in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).
- **Show in results view as**
  - **XHTML** - Can only be enabled if **Open in Browser/System Application** is disabled. If this is checked, Oxygen XML Editor plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

    Important: When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you want to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

  - **XML** - If this is checked, Oxygen XML Editor plugin displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlighting*, specific for XML documents.
- **Image URLs are relative to** - If **Show in results view as XHTML** is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

DITA OT Transformation

To create a **DITA OT Transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **DITA OT Transformation**.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **DITA OT Transformation**.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **DITA OT Transformation**.

  Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **DITA Transformation Type** dialog box that presents the list of possible outputs.

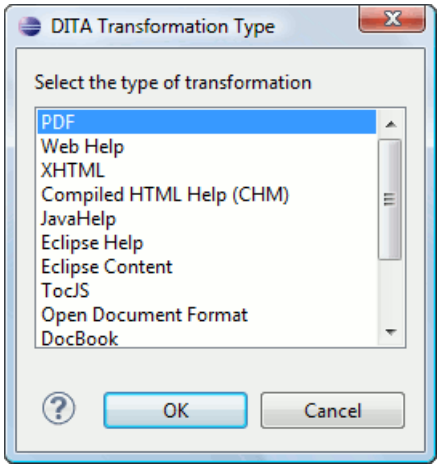


Figure 304: DITA Transformation Type Dialog Box

Select the desired type of output and click **OK**. This opens the **New Scenario** dialog box, which allows you to configure the options that control the transformation.

The lower part of the dialog box contains the following tabs (only those that are appropriate for the chosen output type will be displayed):

- **Skins** (Available for **WebHelp** and **WebHelp with Feedback** output types).
- **FO Processor** (Available for PDF output types).
- **Parameters**
- **Filters**
- **Advanced**
- **Output**

For information about creating an entirely new DITA OT transformation, see *Creating a DITA OT Customization Plugin* on page 478 and *Installing a Plugin in the DITA Open Toolkit* on page 480.

The Skins Tab

A *skin* is a collection of CSS properties that can alter the look of the output by changing colors, font types, borders, margins, and paddings. This allows you to rapidly adapt the look and feel of the output for your organization.

Oxygen XML Editor plugin provides a set of predefined *skins* for the **DITA Map WebHelp** and **DITA Map WebHelp with Feedback** transformation scenarios.

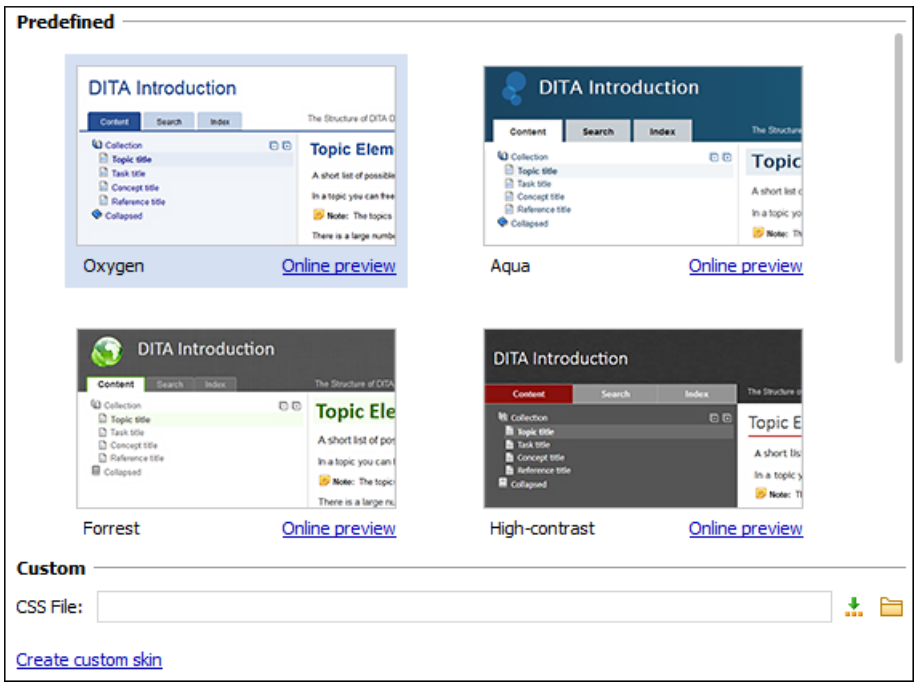


Figure 305: The Skins Tab

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 (notice the light blue border around the skin preview). If you want to obtain an output without any customization, deselect the currently selected skin.

To see how the *skin* looks when applied on a sample documentation project that is stored on the Oxygen XML Editor plugin 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 referenced in this parameter take precedence over the properties declared in the skin set in the **Skins tab**.

The FO Processor Tab

This tab allows you to select an FO Processor, when you choose to generate PDF output.

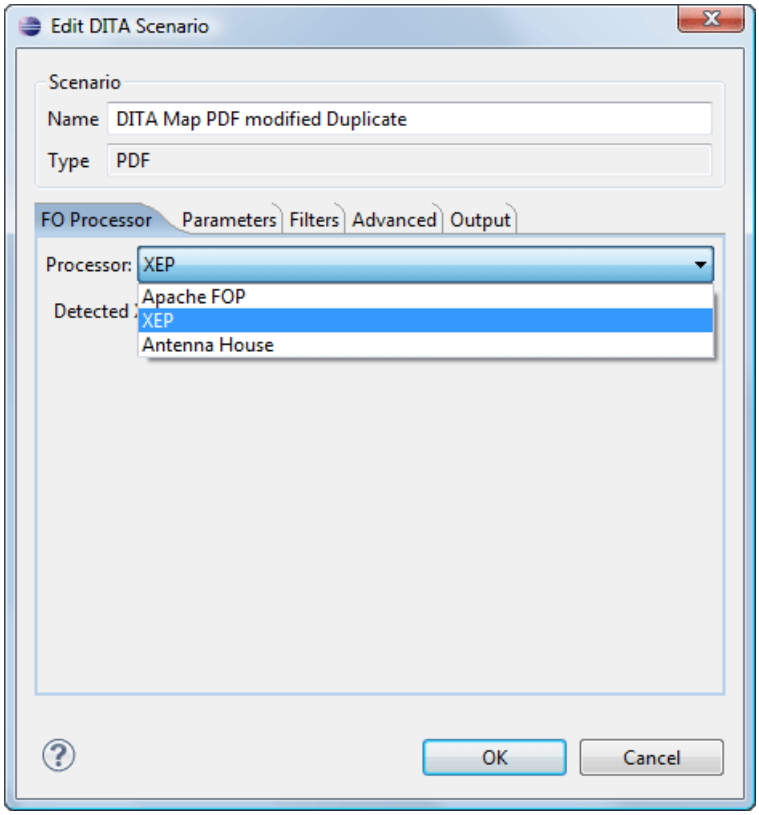


Figure 306: FO Processor Configuration Tab

You can choose the following processors:

- **Apache FOP** - The default processor that comes bundled with .
- **XEP** - The *RenderX* XEP processor.

  If XEP is already installed, displays the detected installation path under the drop-down menu.

  XEP is considered installed if it was detected in one of the following sources:

  - XEP was configured as an external FO Processor in the ***FO Processors*** *option page*.
  - The system property *com.oxygenxml.xep.location* was set to point to the XEP executable file for the platform (for example: `xep.bat` on Windows).
  - XEP was installed in the `/plugins/org.dita.pdf2/lib` directory of the installation directory.
- **Antenna House** - The *Antenna House* (AH Formatter) processor.

  If Antenna House is already installed, displays the detected installation path under the drop-down menu.

  Antenna House is considered installed if it was detected in one of the following sources:

  - Environment variable set by Antenna House installation (the newest installation version will be used).
  - Antenna House was added as an external FO Processor in the preferences pages.

  To further customize the PDF output obtained from the Antenna House processor:

  - **Edit** the transformation scenario.
  - Open the ***Parameters*** *tab*.
  - Add the `env.AXF_OPT` parameter and point to Antenna House configuration file.

The Parameters Tab

The **Parameters**tab allows you to configure the parameters sent to the DITA-OT build file.

The table displays all the parameters that the DITA-OT documentation specifies as available for each chosen type of transformation (for example: XHTML or PDF), along with their description and current values. You can find more information about each parameter in the *DITA OT Documentation*. You can also add, edit, and remove parameters. Use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

Depending on the type of a parameter, its value can be one of the following:

- A simple text field for simple parameter values.
- A combo box with some predefined values.
- A file chooser and an *editor variable* selector to simplify setting a file path as the value of a parameter.

Note: To input parameter values at runtime, use the *ask editor variable* in the **Value** column.

The following actions are available for managing parameters:

New

Opens the **Add Parameter** dialog box that allows you to add a new parameter to the list. An *editor variable* can be inserted in the text box using the **Insert Editor Variables** button.

Edit

Opens the **Edit Parameter** dialog box that allows you to change the value of the selected parameter by selecting it from a list of allowed values.

Unset

Resets the selected parameter to its default value. Available only for edited parameters with set values.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The Filters Tab

The **Filters** tab allows you to add filters to remove certain content elements from the generated output.

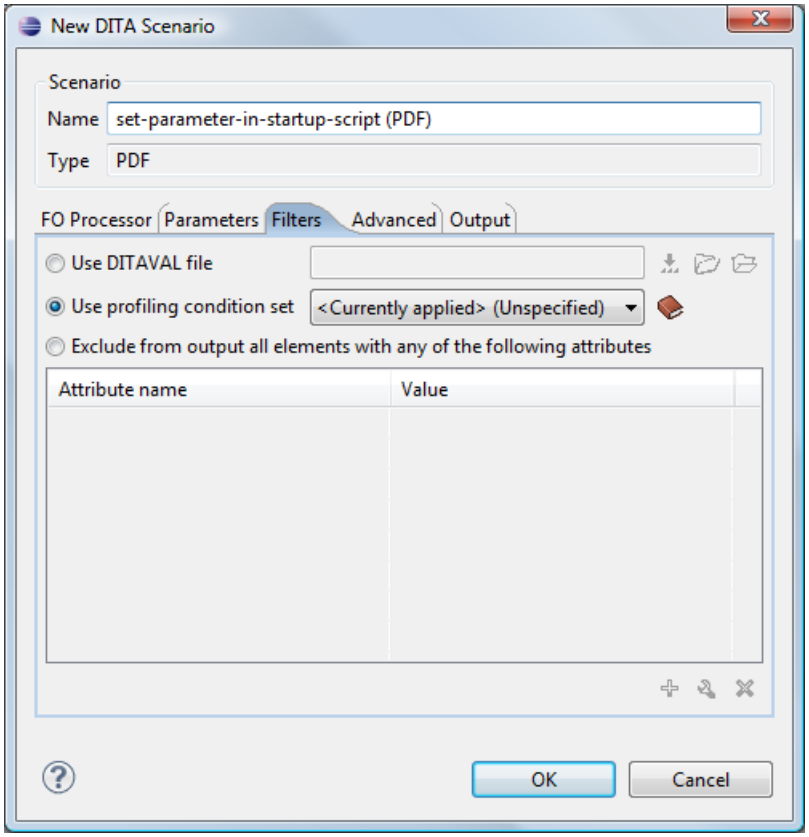


Figure 307: Edit Filters tab

There are three ways to define filters:

- **Use DITAVAL file** - If you already have a DITAVAL file associated with the DITA map, you can specify the file to be used when filtering content. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button. You can find out more about constructing a DITAVAL file in the *DITA OT Documentation*.
- **Use profiling condition set** - Sets the *profiling condition set* that will apply to your transformation.
- **Exclude from output all elements with any of the following attributes** - By using the **New**, **Edit**, or **Delete** buttons at the bottom of the pane, you can configure a list of attributes (name and value) to exclude all elements that contain any of these attributes from the output.

The Advanced Tab

The **Advanced** tab allows you to specify advanced options for the transformation scenario.

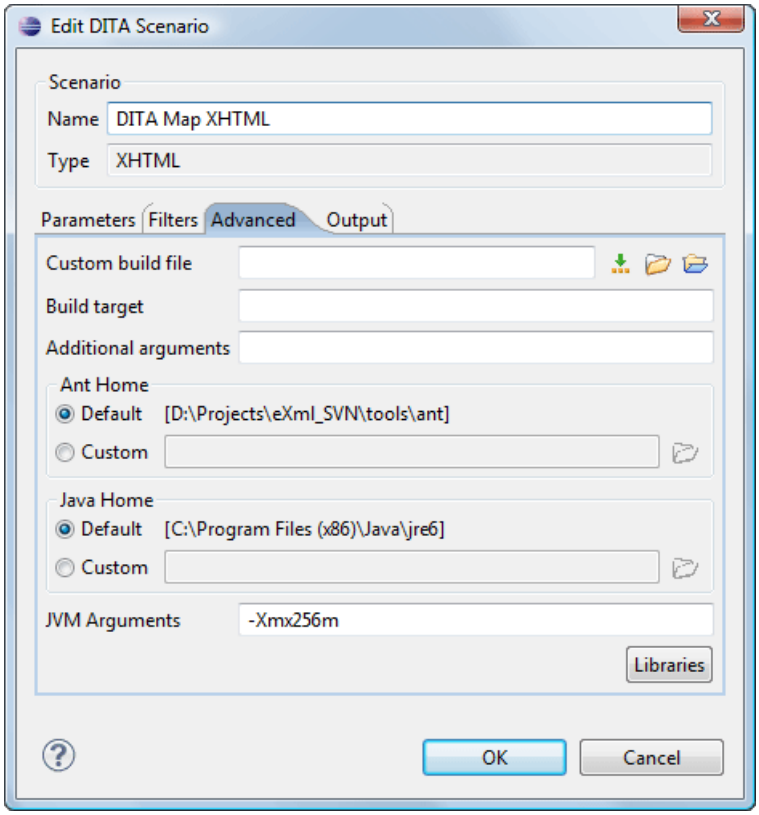


Figure 308: Advanced Settings Tab

You can specify the following parameters:

- **Custom build file** - If you use a custom DITA-OT build file, you can specify the path to the customized build file. If empty, the `build.xml` file from the *dita.dir* parameter that is configured in the **Parameters** tab is used. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button.
- **Build target** - Optionally, you can specify a build target for the build file. If no target is specified, the default `init` target is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the Ant transformation (such as -verbose).
- **Ant Home** - You can choose between the default or custom Ant installation to run the transformation.
- **Java Home** - You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by Oxygen XML Editor plugin.

  Note: It may be possible that the used Java version is incompatible with the DITA Open Toolkit engine. For example, DITA OT 1.8.5 and older requires Java 1.6 or later, while DITA OT 2.0 and newer requires Java 1.7 or newer. Thus, if you encounter related errors running the publishing, consider installing a Java VM that is supported by the DITA OT publishing engine and using it in the **Java Home** text field.

- **JVM Arguments** - This parameter allows you to set specific parameters for the Java Virtual Machine used by Ant. For example, if it is set to -Xmx384m, the transformation process is allowed to use 384 megabytes of memory. When performing a large transformation, you may want to increase the memory allocated to the Java Virtual Machine. This will help avoid Out of Memory error messages (**OutOfMemoryError**).
- **Libraries** - By default, Oxygen XML Editor plugin adds (as high priority) libraries that are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can use this button to specify additional libraries (jar files or additional class paths) to be used by the Ant transformer.

The Output Tab

The **Output** tab allows you to configure options that are related to the location where the output is generated.

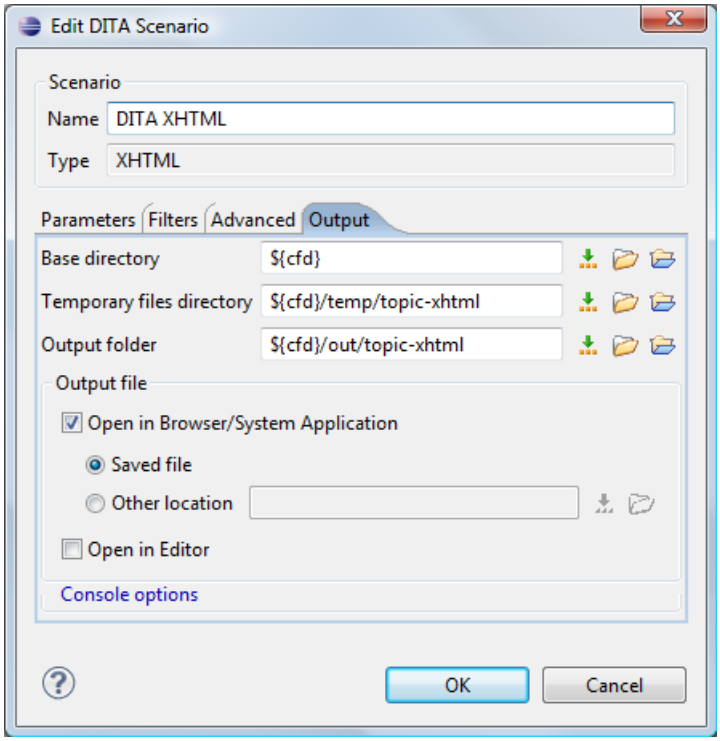


Figure 309: Output Settings Tab

You can specify the following parameters:

- **Base directory** - All the relative paths that appear as values in parameters are considered relative to the base directory. The default value is the directory where the transformed map is located. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.
- **Temporary files directory** - This directory is used to store pre-processed temporary files until the final output is obtained. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.
- **Output folder** - The folder where the content of the final output is stored. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.

  Note: If the DITA map or topic is opened from a remote location or a ZIP file, the parameters must specify absolute paths.

- **Open in Browser/System Application** - If enabled, Oxygen XML Editor plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, `.pdf` files are usually opened in the *Acrobat Reader* application).

  Note: To set the web browser that is used for displaying HTML/XHTML pages, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**.

  - **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin automatically opens the file specified in the **Save As** text field at the end of the transformation.
  - **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor plugin opens the file specified here. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.

Troubleshooting DITA Transformation Errors

If a DITA transformation results in errors or warnings, the information is displayed in the message panel at the bottom of the editor. The information includes the severity, description of the problem, the name of the resource, and the path of the resource.

To help prevent and solve DITA transformation problems, follow these steps:

1. *Validate the DITA map* by using the **Validate and Check for Completeness** action that is available on the **DITA Maps Manager** toolbar and in the **DITA Maps** menu.
2. If this action results in validation errors, solve them prior to executing the transformation. Also, you should pay attention to the warning messages because they may identify problems in the transformation.
3. *Run the DITA transformation scenario*.
4. If the transformation results in errors or warnings, they are displayed in the **Transformation problems** message panel at the bottom of the editor. The following information is presented to help you troubleshoot the problems:
   - **Severity** - The first column displays the following icons that indicate the severity of the problem:
     - **Informational** - The transformation encountered a condition of which you should be aware.
     - **Warning** - The transformation encountered a problem that should be corrected.
     - **Error** - The transformation encountered a more severe problem, and the output is affected or cannot be generated.
   - **Info** - You can click the **See More** icon to open a web page that contains details about DITA-OT error messages.
   - **Description** - A description of the problem.
   - **Resource** - The name of the transformation resource.
   - **System ID** - The path of the transformation resource.
5. Use this information or other resources from the online DITA-OT community to solve the transformation problems before re-executing the transformation scenario.

ANT Transformation

An Ant transformation scenario is usually associated with an Ant build script. Oxygen XML Editor plugin runs an Ant transformation scenario as an external process that executes the Ant build script with the built-in Ant distribution (Apache Ant version 1.8.2) that comes with the application, or optionally with a custom Ant distribution configured in the scenario.

To create an Ant transformation scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **ANT transformation**.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **ANT transformation**.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **ANT transformation**.

  Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog box allows you to configure the options that control the transformation.

The dialog box contains the following tabs:

- *The **Options** tab*.

- *The **Parameters** tab*.
- *The **Output** tab*.

The Options Tab

The **Options** tab allows you to specify the following options:

- **Working directory** - The path of the current directory of the Ant external process. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button.
- **Build file** - The Ant script file that is the input of the Ant external process. An *editor variable* can be inserted for the file path by using the **Insert Editor Variables** button.
- **Build target** - Optionally, you can specify a build target for the Ant script file. If no target is specified, the Ant target that is specified as the default in the Ant script file is used.
- **Additional arguments** - You can specify additional command-line arguments to be passed to the Ant transformation (such as -verbose).
- **Ant Home** - You can choose between the default or custom Ant installation to run the transformation.
- **Java Home** - You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by Oxygen XML Editor plugin.
- **JVM Arguments** - This parameter allows you to set specific parameters for the Java Virtual Machine used by Ant. For example, if it is set to -Xmx384m, the transformation process is allowed to use 384 megabytes of memory. When performing a large transformation, you may want to increase the memory allocated to the Java Virtual Machine. This will help avoid Out of Memory error messages (**OutOfMemoryError**).
- **Libraries** - By default, Oxygen XML Editor plugin adds (as high priority) libraries that are not transformation-dependent and also patches for certain DITA Open Toolkit bugs. You can use this button to specify additional libraries (jar files or additional class paths) to be used by the Ant transformer.

The Parameters Tab

The **Parameters** tab allows you to configure the parameters that are accessible as Ant properties in the Ant build script.

The table displays all the parameters that are available in the Ant build script, along with their description and current values. You can also add, edit, and remove parameters. Use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with the name in bold.

Depending on the type of a parameter, its value can be one of the following:

- A simple text field for simple parameter values.
- A combo box with some predefined values.
- A file chooser and an *editor variable* selector to simplify setting a file path as the value of a parameter.

Note: To input parameter values at runtime, use the *ask editor variable* in the **Value** column.

The following actions are available for managing parameters:

New

Opens the **Add Parameter** dialog box that allows you to add a new parameter to the list. An *editor variable* can be inserted in the text box using the **Insert Editor Variables** button.

Edit

Opens the **Edit Parameter** dialog box that allows you to change the value of the selected parameter by selecting it from a list of allowed values.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

The Output Tab

The **Output** tab contains the following options:

- **Open** - Allows you to specify the file to open automatically when the transformation is finished. Usually, this is the output file of the Ant process. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.

- **In System Application** - The file specified in the **Open** text box is opened in the system application that is set in the operating system as the default application for that type of file (for example, `.pdf` files are usually opened in the *Acrobat Reader* application).
- **In Editor** - The file specified in the **Open** text box is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor).

- The **Show console output** option allows you to specify when to display the console output log. The following options are available:
  - **When build fails** - displays the console output log if the build fails.
  - **Always** - displays the console output log, regardless of whether or not the build fails.

XSLT Transformation

To create an **XSLT transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XSLT transformation**.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XSLT transformation**.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XSLT transformation**.

  Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog box allows you to configure the options that control the transformation.

The dialog box contains the following tabs:

- ***XSLT***
- ***FO Processors***
- ***Output***

The XSLT Tab

The **XSLT** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  Note: If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

  Note: If the transformer engine is one of the built-in XSLT 2.0 / 3.0 engines and *the name of an initial template* is specified in the scenario, the **XML URL** field can be empty. The **XML URL** field can also be empty if you use *external XSLT processors*. Otherwise, a value is mandatory in the **XML URL** field.

- **XSL URL** - Specifies the source XSL file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XSL URL** fields:

Insert Editor Variables

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse workspace

Opens a file browser dialog box allowing you to select a file from the local workspace.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the ***Data Source Explorer*** window.

Search for file

Allows you to find a file in the current project.

The rest of the options available in the **XSLT** tab allow you to further customize the transformation scenario:

- **Use "xml-stylesheet" declaration** - Use the stylesheet declared with an `xml-stylesheet` declaration instead of the stylesheet specified in the **XSL URL** field. By default, this checkbox is not selected and the transformation applies the XSLT stylesheet that is specified in the **XSL URL** field. If it is checked, the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction.
- **Transformer** - This drop-down menu presents all the transformation engines available to Oxygen XML Editor plugin for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog box is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).
  - **Advanced options** - Allows you to configure the advanced options of the Saxon HE / PE / EE engine for the current transformation scenario. To configure the same options globally, go to the *Saxon-HE/PE/EE preferences page*. For the current transformation scenario, these **Advanced options** override the options configured in the *Saxon-HE/PE/EE preferences page*. The **Initial mode and template** option is only available in the **Advanced options**. It is a Saxon-specific option that sets the name of the first XSLT template that starts the XSLT transformation or the initial mode of the transformation.
- **Parameters** - Opens *the **Configure parameters** dialog box*, allowing you to configure the XSLT parameters used in the current transformation. In this dialog box, you can also configure the parameters of additional stylesheets by using the **Additional XSLT stylesheets** button. If the XSLT transformation engine is custom-defined, you can not use this dialog box to configure the parameters sent to the custom engine. Instead, you can copy all parameters from the dialog box using contextual menu actions and edit the custom XSLT engine to include the necessary parameters in the command line that starts the transformation process.
- **Extensions** - Opens *the dialog box for configuring the XSLT/XQuery extension jars or classes* that define extension Java functions or extension XSLT elements used in the transformation.
- **Additional XSLT stylesheets** - Opens *the dialog box for adding XSLT stylesheets* that are applied on the main stylesheet that is specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document.

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.
- **XSLT result as input** - The FO processor is applied to the result of the XSLT transformation that is defined in the **XSLT** tab.
- **XML URL as input** - The FO processor is applied to the input XML file.
- **Method** - The output format of the FO processing. Available options depend on the selected processor type.

- **Processor** - Specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Prompt for file** - At the end of the transformation, a file browser dialog box is displayed for specifying the path and name of the file that stores the transformation result.
- **Save As** - The path of the file where the result of the transformation is stored. The path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, `.pdf` files are usually opened in the *Acrobat Reader* application).

  Note: To set the web browser that is used for displaying HTML/XHTML pages, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**.

  - **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin automatically opens the file specified in the **Save As** text field at the end of the transformation.
  - **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor plugin opens the file specified here. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.

- **Open in editor** - When this is enabled, the transformation result specified in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).
- **Show in results view as**
  - **XHTML** - Can only be enabled if **Open in Browser/System Application** is disabled. If this is checked, Oxygen XML Editor plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

    Important: When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you want to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

  - **XML** - If this is checked, Oxygen XML Editor plugin displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlighting*, specific for XML documents.
- **Image URLs are relative to** - If **Show in results view as XHTML** is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

XProc Transformation

A sequence of transformations described by an XProc script can be executed with an XProc transformation scenario. To create an **XProc transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XProc transformation**.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XProc transformation**.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XProc transformation**.

  Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are

associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog box allows you to configure the options that control the transformation.

The lower part of the dialog box contains the following tabs:

- ***XProc***
- ***Inputs***
- ***Parameters***
- ***Outputs***
- ***Options***

The XProc Tab

The **XProc** tab contains the following options:

- **XProc URL** - Specifies the source XSL file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter value in the **XProc URL**:

 **Insert Editor Variables**

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field.

 **Browse for local file**

Opens a local file browser dialog box allowing you to select a local file.

 **Browse workspace**

Opens a file browser dialog box allowing you to select a file from the local workspace.

 **Browse for remote file**

Opens an URL browser dialog box allowing you to select a remote file.

 **Browse for archived file**

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

 **Browse Data Source Explorer**

Opens the ***Data Source Explorer*** window.

 **Search for file**

Allows you to find a file in the current project.

- **Processor** - Allows you to select the XProc engine. You can select the built-in *Calabash* engine or a custom engine that is *configured in the **Preferences** dialog box*.

The Inputs Tab

The **Inputs** tab contains a list with the ports that the XProc script uses to read input data. Use the **Filter** text box to search for a specific term in the entire ports collection.

Each input port has an assigned name in the XProc script. The XProc engine reads data from the URL specified in the **URL** column. The *built-in editor variables* and *custom editor variables* can be used to specify the URL.

The following actions are available for managing the input ports:

New

Opens an **Edit** dialog box that allows you to add a new port and its URL.

Edit

Opens an **Edit** dialog box that allows you to modify the selected port and its URL.

Delete

Removes the selected port from the list. It is enabled only for new ports that have been added to the list.

The Parameters Tab

The **Parameters** tab presents a list of ports and parameters collected from the XProc script. The tab is divided into three sections:

- *List of Ports* - In this section you can use the **New** and **Delete** buttons to add or remove ports.
- *List of Parameters* - This section presents a list of parameters for each port and includes columns for the parameter name, namespace URI, and its value. Use the **Filter** text box to search for a specific term in the entire parameters collection. You can use the **New** and **Delete** buttons to add or remove parameters. You can edit the value of each cell in this table by double-clicking the cell. You can also sort the parameters by clicking the column headers.
- *Editor Variable Information* - The *built-in editor variables* and *custom editor variables* can be used for specifying the URI. The message pane at the bottom of the dialog box provides more information about the editor variables that can be used.

The Outputs Tab

The **Outputs** tab displays a list of output ports (along with the URL) collected from the XProc script. Use the **Filter** text box to search for a specific term in the entire ports collection. You can also sort the columns by clicking the column headers.

The following actions are available for managing the output ports:

New

Opens an **Edit** dialog box that allows you to add a new output port and its URL. An *editor variable* can be inserted for the URL by using the **Insert Editor Variables** button. There is also a **Show in transformation results view** option that allows you to select whether or not the results will be displayed in the output results view.

Edit

Opens an **Edit** dialog box that allows you to edit an existing output port and its URL. An *editor variable* can be inserted for the URL by using the **Insert Editor Variables** button. There is also a **Show in transformation results view** option that allows you to select whether or not the results will be displayed in the output results view.

Delete

Removes the selected output port from the list. It is enabled only for new ports that have been added to the list.

Additional options that are available at the bottom of this tab include:

Open in Editor

If this option is selected, the XProc transformation result is automatically opened in an editor panel.

Open in Browser/System Application

If this option is selected, you can specify a file to be opened at the end of the XProc transformation in the browser or system application that is associated with the file type. An *editor variable* can be inserted for the path by using the **Insert Editor Variables** button.

Results

The result of the XProc transformation can be displayed as a sequence in an output view with two sections:

- A list with the output ports on the left side.
- The content that correspond to the selected output port on the right side.

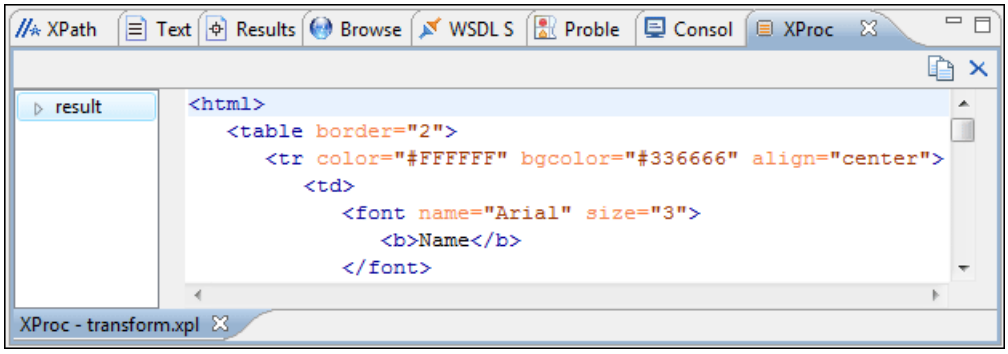


Figure 310: XProc Transformation Results View

The Options Tab

The **Options** tab displays a list of the options collected from the XProc script. The tab is divided into two sections:

- *List of Options* - This section presents a list of options and includes columns for the option name, namespace URI, and its value. Use the **Filter** text box to search for a specific term in the entire options collection. You can use the **New** and **Delete** buttons to add or remove options. You can edit the value of each cell in this table by double-clicking the cell. You can also sort the parameters by clicking the column headers. The names of edited options are displayed in bold.
- *Editor Variable Information* - The *built-in editor variables* and *custom editor variables* can be used for specifying the URI. This section provides more information about the editor variables that can be used.

Configuring Calabash with XEP

To generate PDF output from your XProc pipeline (when using the Calabash XProc processor), follow these steps:

1. Open the `[OXYGEN INSTALLATION DIRECTORY]/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 Editor plugin.

Integration of an External XProc Engine

The Javadoc documentation of the XProc API is available for download from the application website as a zip file *`xprocAPI.zip`*. To create an XProc integration project, follow these steps:

1. Move the `oxygen.jar` file from `[OXYGEN_DIR]/lib` to the `lib` folder of your project.
2. Implement the `ro.sync.xml.transformer.xproc.api.XProcTransformerInterface` interface.
3. Create a Java archive (jar) from the classes you created.
4. Create a `engine.xml` file according with the `engine.dtd` file. The attributes of the `engine` element are as follows:
    1. `name` - The name of the XProc engine.
    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 the integration.
    5. `engineVersion` - The version of the integrated engine.
    6. `vendor` - The name of the vendor / implementer.

7. `supportsValidation` - `true` if the engine supports validation, `false` otherwise.

   The `engine` element has only one child, `runtime`. The `runtime` element contains several `library` elements with the `name` attribute containing the relative or absolute location of the libraries necessary to run this integration.

5. Create a folder with the name of the integration in the `[OXYGEN_DIR]/lib/xproc`.
6. Place the `engine.xml` and all the libraries necessary to run the new integration in that folder.

XQuery Transformation

To create an **XQuery transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XQuery transformation**.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XQuery transformation**.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **XQuery transformation**.

  Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog box allows you to configure the options that control the transformation.

The lower part of the dialog box contains the following tabs:

- ***XQuery***
- ***FO Processor***
- ***Output***

The XQuery Tab

The **XQuery** tab contains the following options:

- **XML URL** - Specifies the source XML file. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, then the file is used directly from its remote location.

  Note: If the transformer engine is Saxon 9 and a custom URI resolver is configured in **Preferences** for Saxon 9, the XML input of the transformation is passed to that URI resolver.

- **XQuery URL** - specifies the source XQuery file that the transformation will use. This URL is resolved through the catalog resolver. If the catalog does not have a mapping for the URL, the file is used directly from its remote location.

You can use the following browsing buttons to enter values in the **XML URL** and **XQuery URL** fields:

Insert Editor Variables

Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse workspace

Opens a file browser dialog box allowing you to select a file from the local workspace.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the ***Data Source Explorer*** window.

Search for file

Allows you to find a file in the current project.

The rest of the options available in the **XQuery** tab allow you to further customize the transformation scenario:

- **Transformer** - This drop-down menu presents all the transformation engines available to Oxygen XML Editor plugin for performing a transformation. These are the built-in engines and *the external engines defined in the Custom Engines preferences page*. The engine you choose in this dialog box is used as the default transformation engine. Also, if an XSLT or XQuery document does not have an associated validation scenario, this transformation engine is used in the validation process (if it provides validation support).;
  - **Advanced options** - *configure advanced options specific for the Saxon HE / PE / EE engine*.
- **Parameters** - Opens the **Configure parameters** dialog box for configuring the XQuery parameters. You can use buttons in this dialog box you can add, edit, or remove parameters. If the XQuery transformation engine is custom-defined you can not use this dialog box to set parameters. Instead, you can copy all parameters from the dialog box using contextual menu actions and edit the custom XQuery engine to include the necessary parameters in the command line that starts the transformation process.

  Note: Use the **Filter** text box to search for a specific term in the entire parameters collection.
- **Extensions** - Opens *the dialog box for configuring the XSLT/XQuery extension jars or classes* that define extension Java functions or extension XSLT elements used in the transformation.

The FO Processor Tab

The **FO Processor** tab contains the following options:

- **Perform FO Processing** - Specifies whether an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.
- **XQuery result as input** - the FO processor is applied to the result of the XQuery transformation defined in the **XQuery** tab.
- **XML URL as input** - The FO processor is applied to the input XML file.
- **Method** - The output format of the FO processing. Available options depend on the selected processor type.
- **Processor** - Specifies the FO processor. It can be the built-in Apache FOP processor or an *external processor*.

The Output Tab

The **Output** tab contains the following options:

- **Present as a sequence** - Enabling this option will reduce the time necessary to fetch the full result, as it will only fetch the first chunk of the result.
- **Prompt for file** - At the end of the transformation, a file browser dialog box is displayed for specifying the path and name of the file that stores the transformation result.
- **Save As** - The path of the file where the result of the transformation is stored. The path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.
- **Open in Browser/System Application** - If enabled, Oxygen XML Editor plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, `.pdf` files are usually opened in the *Acrobat Reader* application).

Note: To set the web browser that is used for displaying HTML/XHTML pages, *open the **Preferences** dialog box*, then go to **General** > **Web Browser**.

- **Saved file** - When **Open in Browser/System Application** is selected, this button can be used to specify that Oxygen XML Editor plugin automatically opens the file specified in the **Save As** text field at the end of the transformation.
- **Other location** - When **Open in System Application** is selected, this option can be used to specify that Oxygen XML Editor plugin opens the file specified here. The file path can include *special Oxygen XML Editor plugin editor variables* or *custom editor variables* by using the **Insert Editor Variables** button.

- **Open in editor** - When this is enabled, the transformation result specified in the **Save As** field is opened in a new editor panel with the appropriate built-in editor type (for example, if the result is an XML file it is opened in the built-in XML editor, or if it is an XSL-FO file it is opened with the built-in FO editor).
- **Show in results view as**
  - **XHTML** - Can only be enabled if **Open in Browser/System Application** is disabled. If this is checked, Oxygen XML Editor plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.

    Important: When transforming very large documents, you should be aware that enabling this feature results in a very long processing time, necessary for rendering the transformation result in the XHTML result viewer panel. This drawback is due to the built-in Java XHTML browser implementation. To avoid delays for large documents, if you want to see the XHTML result of the transformation, you should use an external browser by checking the **Open in browser** option.

  - **XML** - If this is checked, Oxygen XML Editor plugin displays the transformation result in an XML viewer panel at the bottom of the application window with *syntax highlighting*, specific for XML documents.
- **Image URLs are relative to** - If **Show in results view as XHTML** is checked, this text field specifies the path used to resolve image paths contained in the transformation result.

SQL Transformation

To create an **SQL transformation** scenario, use one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **SQL transformation**.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **SQL transformation**.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then click the **New** button and select **SQL transformation**.

Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

All three methods open the **New Scenario** dialog box. This dialog box allows you to configure the following options that control the transformation:

- **Name** - The unique name of the SQL transformation scenario.
-
- **SQL URL** - Allows you to specify the URL of the SQL script. You can use the following browsing buttons to enter value in this field:

  Insert Editor Variables

  Opens a pop-up menu allowing you to introduce special *Oxygen XML Editor plugin editor variables* or *custom editor variables* in the XML URL field.

Browse for local file

Opens a local file browser dialog box allowing you to select a local file.

Browse workspace

Opens a file browser dialog box allowing you to select a file from the local workspace.

Browse for remote file

Opens an URL browser dialog box allowing you to select a remote file.

Browse for archived file

Opens a zip archive browser dialog box allowing you to select a file from a zip archive.

Browse Data Source Explorer

Opens the ***Data Source Explorer*** window.

Search for file

Allows you to find a file in the current project.

- **Connection** - Allows you to select a connection from a drop-down list. To configure a connection, use the **Advanced options** button to open the *data source preferences page*.
- **Parameters** - Allows you to configure the parameters of the transformation.

Configure Transformation Scenario(s) Dialog Box

You can use the **Configure Transformation Scenarios(s)** dialog box to manage both the *built-in transformation scenarios* and the ones you create.

To open this dialog box, use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu.

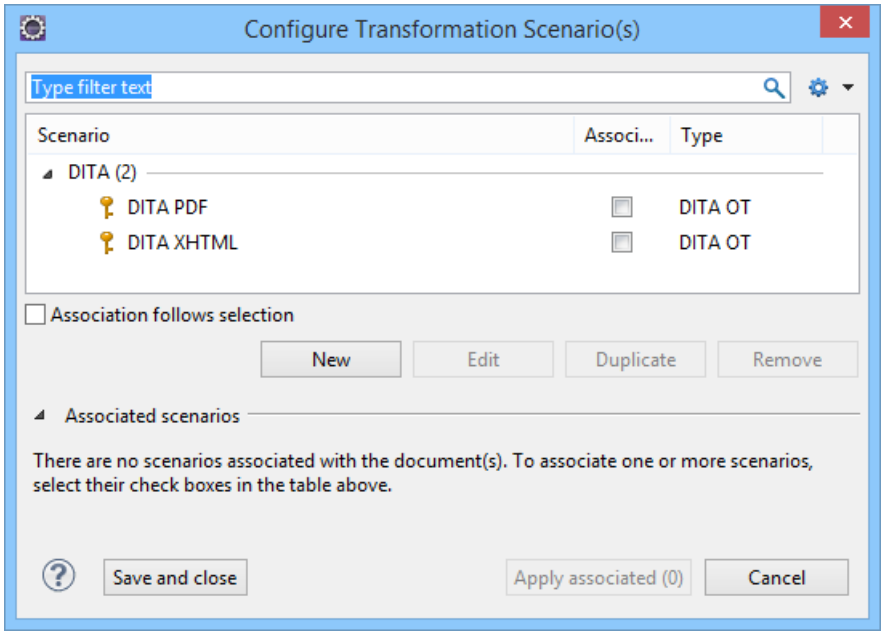


Figure 311: Configure Transformation Scenario(s) Dialog Box

The top section of the dialog box contains a filter that allows you to search through the scenarios list. The **Settings** button allows you to configure the following options:

- **Show all scenarios** - Select this option to display all the available scenarios, regardless of the document they are associated with.

- **Show only the scenarios available for the editor** - Select this option to only display the scenarios that Oxygen XML Editor plugin can apply for the current document type.
- **Show associated scenarios** - Select this option to only display the scenarios associated with the document you are editing.
- **Import scenarios** - This option opens the **Import scenarios** dialog box that allows you to select the `scenarios` file that contains the scenarios you want to import. If one of the scenarios you import is identical to an existing scenario, Oxygen XML Editor plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:
  - Keep or replace the existing scenario.
  - Keep both scenarios.

  Note: When you keep both scenarios, Oxygen XML Editor plugin adds `imported` to the name of the imported scenario.

- **Export selected scenarios** - Use this option to export transformation and validation scenarios individually. Oxygen XML Editor plugin creates a `scenarios` file that contains the scenarios that you export.

The middle section of the dialog box displays the scenarios that you can apply to the current document. You can view both the scenarios associated with the current document type and the scenarios defined at project level. The following columns are used to display the transformation scenarios:

- **Association** - The check-boxes in this column mark whether a transformation scenario is associated with the current document.
- **Scenario** - This column presents the names of the transformation scenarios.
- **Type** - Displays the type of the transformation scenario. For further details about the different types of transformation scenarios available in Oxygen XML Editor plugin see the *Defining a New Transformation Scenario* section.
- **Storage** - Displays where a transformation scenario is stored (the **Show Storage** option must be enabled.)

To sort each column you can left-click its header. The contextual menu of each header allows you to do the following:

- **Show Type** - Use this option to display the transformation type of each scenario.
- **Show Storage** - Use this option to display the storage location of the scenarios.
- **Group by Type** - Select this option to group the scenarios by their type.
- **Group by Storage** - Select this option to group the scenarios by their storage location.
- **Ungroup all** - Select this option to ungroup all the scenarios.
- **Reset Layout** - Select this option to restore the default settings of the layout.

The bottom section of the dialog box contains the following actions:

- **Association follows selection** - Enable this check-box to automatically associate selected transformation scenarios with the current document. This option can also be used for multiple selections.

  Note: When this option is enabled, the **Association** column is hidden.

- **New** - This button allows you to create a new transformation scenario, *depending upon its type*.
- **Edit** - This button opens the **Edit Scenario** dialog box that allows you to configure the options of the transformations scenario.

  Note: If you try to edit a transformation scenario associated with a defined document type, Oxygen XML Editor plugin displays a warning message to inform you that this is not possible and gives you the option to create a *duplicate transformation scenario* to edit instead.

- **Duplicate** - Use this button to create a *duplicate transformation scenario*.
- **Remove** - Use this button to remove transformation scenarios.

  Note: Removing scenarios associated with a defined document type is not allowed.

The **Edit**, **Duplicate**, **Remove**, **Import scenarios**, and **Export selected scenarios**actions are also available in the contextual menu of the transformation scenarios listed in the middle section of the dialog box.

Duplicating a Transformation Scenario

Use the following procedure to duplicate a transformation scenario. This is useful for creating a scenario that is similar to an existing one.

1. Open the **Configure Transformation** dialog by using the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu.
2. Create a copy of a scenario by selecting the scenario and clicking the **Duplicate** button.
3. Enter a new name in the**Name** field.
   a) You can choose to save the scenarios at project level by selecting the **Project Options** setting.
4. Click **OK** to save the scenario.

Editing a Transformation Scenario

Editing a transformation scenario is useful if you need to configure some of its parameters.

Oxygen XML Editor plugin allows you to configure existing transformation scenarios by using one of the following methods:

- Go to **Window** > **Show View** and select **Transformation Scenarios** to display this view. Then select the scenario and click the **Edit** button.
- Use the **Configure Transformation Scenario(s) (Alt + Shift + T, C (Meta + Alt + T, C on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then select the scenario and click the **Edit** button.
- Use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu. Then select the scenario and click the **Edit** button.

  Note: If a scenario is already associated with the edited document, selecting **Apply Transformation Scenario(s)** runs the associated scenario automatically. You can check whether transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

You can edit transformation scenarios that are defined at project level only. To edit a transformation scenario that is associated with a defined document type, duplicate it and edit the duplicated scenario.

Apply Batch Transformations

A transformation action can be applied on a batch of selected files *from the contextual menu of the **Project** view* without having to open the files involved in the transformation. You can apply the same scenario to a batch of files or multiple scenarios to a single file or batch of files.

1. (Optional, but recommended) Organize the files you want to transform in logical folders.
   a) Create a logical folder in the **Project** view by using the **New** > **Logical Folder** action from the contextual menu of the root file.
   b) Add files you want to transform to the logical folder by using the **Add Files** or **Add Edited File** actions from the contextual menu of the logical folder.

      Note: You can skip this step if the files are already in a dedicated folder that does not include any additional files or folders. You can also manually select the individual files in the **Project** view each time you want to transform them, but this can be tedious.

2. Right-click the newly created logical folder and select **Transform** > **Configure Transformation Scenario(s)** to select one or more transformation scenarios to be applied on all the files in the logical folder.

   Note: These types of transformation scenarios must be configured with the current file (`${cf}`) or current file URL (`${currentFileURL}`) editor variables for the input file. This ensures that each file becomes the current file when its turn arrives in the batch transformation process. Edit the transformation scenario to make sure the appropriate editor variable is assigned for the input file. For example, for a DocBook PDF transformation make sure the **XML URL** input box is set to the `${currentFileURL}` editor variable. For a DITA PDF transformation make sure the `args.input` parameter is set to the `${cf}` editor variable.

3. Now that logical folder has been associated with one or more transformation scenarios, whenever you want to apply the same batch transformation you can select **Transform** > **Transform with** from the contextual menu and the same previously associated scenario(s) will be applied.
4. If you want a different type of transformation to be applied to each file inside the logical folder, associate individual scenarios for each file and select **Transform** > **Apply Transformation Scenario(s)** from the contextual menu of the logical folder.

Built-in Transformation Scenarios

Oxygen XML Editor plugin included preconfigured built-in transformation scenarios that are used for common transformations. To obtain the desired output, use the **Apply Transformation Scenario(s) (Alt + Shift + T, T (Meta + Alt + T, T on OS X))** action from the **Transformation** toolbar or the **XML** menu and choose one of the built-in scenarios for the current document.

You can use the **Apply Transformation Scenario(s)** action even if the current document is not associated with a transformation scenario.

If the document contains an `xml-stylesheet` processing instruction that refers to an XSLT stylesheet (commonly used to display the document in web browsers), Oxygen XML Editor plugin prompts you to associate the document with a built-in transformation scenario.

The default transformation scenario is suggested based on the processing instruction from the edited document. The **XSL URL** field of the default transformation scenario contains the URL from the `href` attribute of the processing instruction. By default, the **Use xml-stylesheet declaration** check-box is enabled, Saxon is used as the transformation engine, and no FO processing is performed. The result of the transformation is store in a file with the same URL as the edited document, but the extension is changed to `html`. The name and path are preserved because the output file name is specified with the help of two *editor variables*: ${cfd} and ${cfn}.

Sharing the Transformation Scenarios

The transformation scenarios and their settings can be shared with other users by *exporting them to a specialized scenarios file* that can then be imported.

Transformation Scenarios View

You can manage the transformation scenarios by using the **Transformation Scenarios** view. To open this view, go to **Window** > **Show View** > **Transformation Scenarios**.

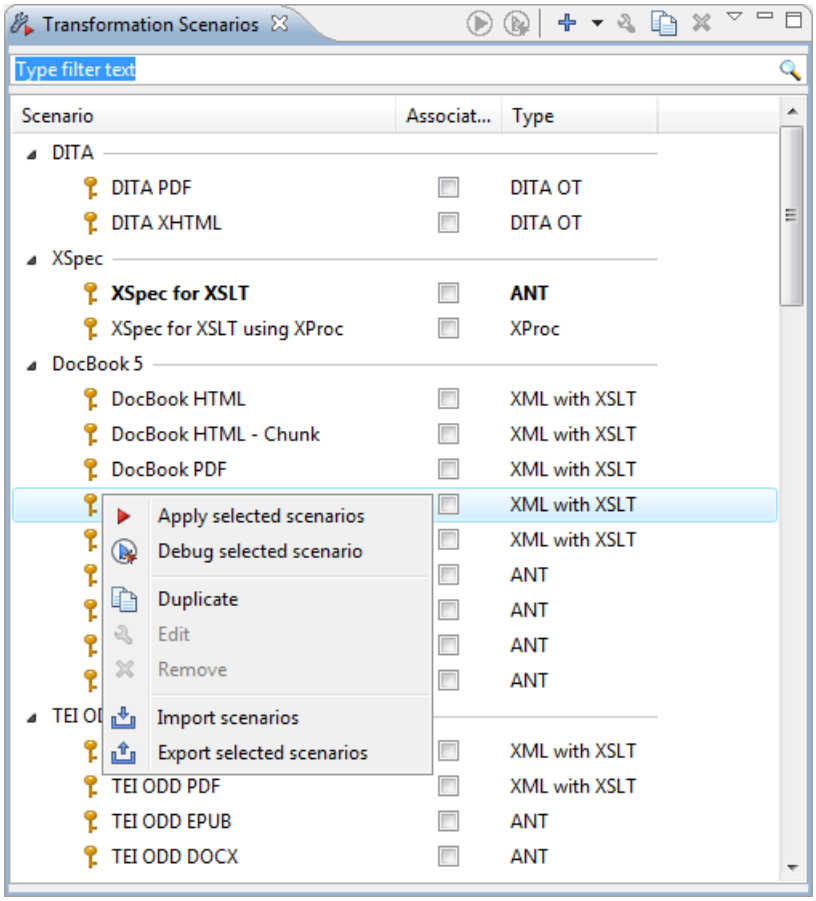


Figure 312: The Scenarios view

The following options are available in the contextual menu of the **Transformation Scenarios** view:

Apply selected scenarios

Select this option to run the current transformation scenario.

Debug selected scenario

Select this option to switch to the **Debugger** perspective and initialize it with the parameters from the scenario (the XML, XSLT, or XQuery input, the transformation engine, the XSLT parameters).

Duplicate

Adds a new scenario to the list that is a duplicate of the current scenario. It is useful for creating a scenario that is similar to an existing one.

Edit

Opens the dialog box for editing the parameters of a transformation scenario.

Remove

Removes the current scenario from the list. This action is also available by using the **Delete** key.

Import scenarios

This option opens the **Import scenarios** dialog box that allows you to select the `scenarios` file that contains the scenarios you want to import. If one of the scenarios you import is identical to an existing scenario, Oxygen XML Editor plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:

- Keep or replace the existing scenario.
- Keep both scenarios.

Note: When you keep both scenarios, Oxygen XML Editor plugin adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export transformation and validation scenarios individually. Oxygen XML Editor plugin creates a `scenarios` file that contains the scenarios that you export.

Along with the options available in the contextual menu, the **Transformation Scenarios** view toolbar contains a **New** drop-down button that contains a list of the scenarios you can create. Oxygen XML Editor plugin determines the most appropriate scenarios for the current type of file and displays them at the beginning of the list, followed by the rest of the scenarios.

The **Settings** drop-down menu allows you to configure the following options:

- **Show all scenarios** - Select this option to display all the available scenarios, regardless of the document they are associated with.
- **Show only the scenarios available for the editor** - Select this option to only display the scenarios that Oxygen XML Editor plugin can apply for the current document type.
- **Show associated scenarios** - Select this option to only display the scenarios associated with the document you are editing.
- **Import scenarios** - This option opens the **Import scenarios** dialog box that allows you to select the `scenarios` file that contains the scenarios you want to import. If one of the scenarios you import is identical to an existing scenario, Oxygen XML Editor plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:
  - Keep or replace the existing scenario.
  - Keep both scenarios.

    Note: When you keep both scenarios, Oxygen XML Editor plugin adds `imported` to the name of the imported scenario.

- **Export selected scenarios** - Use this option to export transformation and validation scenarios individually. Oxygen XML Editor plugin creates a `scenarios` file that contains the scenarios that you export.
- **Show Type** - Use this option to display the transformation type of each scenario.
- **Show Storage** - Use this option to display the storage location of the scenarios.
- **Group by Type** - Select this option to group the scenarios by their type.
- **Group by Storage** - Select this option to group the scenarios by their storage location.
- **Ungroup all** - Select this option to ungroup all the scenarios.
- **Reset Layout** - Select this option to restore the default settings of the layout.

Oxygen XML Editor plugin supports multiple scenarios association. To associate multiple scenarios with a document, enable the check-boxes in front of each scenario. You can also associate multiple scenarios with a document from the **Configure Transformation Scenario(s)** or **Configure Validation Scenario(s)** dialog boxes.

The **Transformation Scenarios** presents both global scenarios and project scenarios. By default, Oxygen XML Editor plugin presents the items in the **Transformation Scenarios** in the following order: scenarios matching the current framework, scenarios matching the current project, scenarios matching other frameworks. You can group the scenarios depending on the columns in the **Transformation Scenarios** view. Right click the name of a column to choose how to group the scenarios. The following grouping options are available:

- **Group by Type** - Select this option to group the scenarios by their type.
- **Group by Storage** - Select this option to group the scenarios by their storage location.

Debugging PDF Transformations

To debug a DITA PDF transformation scenario using the XSLT Debugger follow these steps:

1. *Open the **Preferences** dialog box*, go to **XML** > **XML Catalog**, click **Add**, and select the file located at `DITA_OT_DIR\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 `DITA_OT_DIR\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/DITA_OT_DIR/plugins/org.dita.pdf2/cfg`.
9. Debug the transformation scenario.

XSLT Processors

This section explains how to configure an XSLT processor and extensions for such a processor in Oxygen XML Editor plugin.

Supported XSLT Processors

Oxygen XML Editor plugin includes the following XSLT processors:

- **Xalan 2.7.1** - *Xalan-Java* is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements XSL Transformations (XSLT) Version 1.0 and XML Path Language (XPath) Version 1.0.
- **Saxon 6.5.5** - *Saxon 6.5.5* is an XSLT processor that implements the Version 1.0 XSLT and XPath with a number of powerful extensions. This version of Saxon also includes many of the new features that were first defined in the XSLT 1.1 working draft, but for conformance and portability reasons these are not available if the stylesheet header specifies `version="1.0"`.
- **Saxon 9.6.0.7 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 are present in Saxon-PE.
- **Saxon 9.6.0.7 Enterprise Edition (EE)** - *Saxon EE* is the schema-aware edition of Saxon and it is one of the built-in processors included in Oxygen XML Editor plugin. Saxon EE includes an XML Schema processor, and schema-aware XSLT, XQuery, and XPath processors.

  The validation in schema aware transformations is done according to the W3C XML Schema 1.0 or 1.1. This can be *configured in Preferences*.

  Note: Oxygen XML Editor plugin implements a Saxon framework that allows you to create Saxon configuration files. Two templates are available: **Saxon collection catalog** and **Saxon configuration**. Both of these templates support content completion, element annotation, and attribute annotation.

  Note: Saxon can use the *ICU-J localization library* (`saxon9-icu.jar`) to add support for sorting and date/number formatting in a wide variety of languages. This library is not included in the Oxygen XML Editor plugin installation kit. However, Saxon will use the default collation and localization support available in the currently used JRE. To enable this capability follow these steps:

  1. Download Saxon 9.6.0.7 Professional Edition (PE) or Enterprise Edition (EE) from *http://www.saxonica.com*.

2. Unpack the downloaded archive.
3. Create a new XSLT transformation scenario (or edit an existing one). In the **XSLT** tab, click the **Extensions** button to open the list of additional libraries used by the transformation process.
4. Click **Add** and browse to the folder where you unpacked the downloaded archive and choose the `saxon9-icu.jar` file.

Note that the `saxon9-icu.jar` should NOT be added to the application library folder because it will conflict with another version of the ICU-J library that comes bundled with Oxygen XML Editor plugin.

- **Saxon-CE (Client Edition)** is Saxonica's implementation of XSLT 2.0 for use on web browsers. Oxygen XML Editor plugin provides support for editing stylesheets that contain Saxon-CE extension functions and instructions. This support improves the validation, content completion, and syntax highlighting.

  Note: Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Editor plugin only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about *executing such a stylesheet on Saxonica's website*.

  Note: A specific template, named **Saxon-CE stylesheet**, is available in the New From Templates wizard.

- **Xsltproc (libxslt)** - *Libxslt* is the XSLT C library developed for the Gnome project. `Libxslt` is based on *libxml2*, the XML C library developed for the Gnome project. It also implements most of the EXSLT set of processor-portable extensions, functions, and some of Saxon's evaluate and expression extensions. The *libxml2* version included in Oxygen XML Editor plugin is 2.7.6 and the `Libxslt` version is 1.1.26.

  Oxygen XML Editor plugin uses `Libxslt` through its command line tool (`Xsltproc`). The XSLT processor is included in the distribution kit of the stand-alone version for Windows and Mac OS X. Since there are differences between various Linux distributions, on Linux you must install `Libxslt` on your machine as a separate application and set the PATH variable to contain the `Xsltproc` executable.

  If you do not have the `Libxslt` library already installed, you should copy the following files from Oxygen XML Editor plugin stand-alone installation directory to the root of the `com.oxygenxml.editor_17.1` plugin:

  - on Windows: `xsltproc.exe`, `zlib1.dll`,`libxslt.dll`,`libxml2.dll`, `libexslt.dll`,`iconv.dll`
  - on Linux: `xsltproc`,`libexslt.so.0`, `libxslt.so.1`,`libxsml2.so.2`
  - on Mac OS X: `xsltproc.mac`, `libexslt`, `libxslt`, `libxml`

  Note: The Xsltproc processor can be configured from the ***XSLTPROC** options page*.

  Caution: Known problem: file paths containing spaces are not handled correctly in the LIBXML processor. For example, the built-in XML catalog files of the predefined document types (DocBook, TEI, DITA, etc) are not handled properly by LIBXML if Oxygen XML Editor plugin is installed in the default location on Windows (C:\Program Files). This is because the built-in XML catalog files are stored in the `[OXYGEN_DIR]/frameworks` subdirectory of the installation directory, which in this case contains at least a space character.

- **MSXML 4.0** - *MSXML 4.0* is available only on Windows platforms. It can be used for *transformation* and *validation of XSLT stylesheets*.

  Oxygen XML Editor plugin uses the Microsoft XML parser through its command line tool *`msxsl.exe`*.

  Since `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 platforms. It can be used for *transformation* and *validation of XSLT stylesheets*.

  Oxygen XML Editor plugin performs XSLT transformations and validations using .NET Framework's XSLT implementation (`System.Xml.Xsl.XslTransform` class) through the **nxslt** command line utility. The **nxslt** version included in Oxygen XML Editor plugin is 1.6.

You should have the .NET Framework version 1.0 already installed on your system. Otherwise, you will get the following warning: `MSXML.NET requires .NET Framework version 1.0 to be installed. Exit code: 128`.

You can get the .NET Framework version 1.0 from the *Microsoft website*.

- **.NET 1.0** - A transformer based on the `System.Xml` 1.0 library available in the .NET 1.0 and .NET 1.1 frameworks from Microsoft (*http://msdn.microsoft.com/xml/*). It is available only on Windows.

  You should have the .NET Framework version 1.0 or 1.1 already installed on your system. Otherwise, you will get the following warning: `MSXML.NET requires .NET Framework version 1.0 to be installed. Exit code: 128`.

  You can get the .NET Framework version 1.0 from the *Microsoft website*.

- **.NET 2.0** - A transformer based on the `System.Xml` 2.0 library available in the .NET 2.0 framework from *Microsoft*. It is available only on Windows.

  You should have the .NET Framework version 2.0 already installed on your system. Otherwise, you will get the following warning: `MSXML.NET requires .NET Framework version 2.0 to be installed. Exit code: 128`.

  You can get the .NET Framework version 2.0 from the *Microsoft website*.

Configuring Custom XSLT Processors

You can configure and run XSLT and XQuery transformations with processors other than *the ones which come with the Oxygen XML Editor plugin distribution*.

Note: You can not use these custom engines in *the Debugger perspective*.

The output messages of a custom processor are displayed in an output view at the bottom of the application window. If an output message follows *the format of an Oxygen XML Editor plugin linked message*, clicking it highlights the location of the message in an editor panel containing the file referenced in the message.

Configuring the XSLT Processor Extensions Paths

The Xalan and Saxon processors support the use of extension elements and extension functions. Unlike a literal result element, which the stylesheet simply transfers to the result tree, an extension element performs an action. The extension is usually used because the XSLT stylesheet fails in providing adequate functions for accomplishing a more complex task.

For more information about how to use extensions, see the following links:

- **Xalan** - *http://xml.apache.org/xalan-j/extensions.html*
- **Saxon 6.5.5** - *http://saxon.sourceforge.net/saxon6.5.5/extensions.html*
- **Saxon 9.6.0.7** - *http://www.saxonica.com/documentation9.5/index.html#!extensibility*

To set an XSLT processor extension (a directory or a `jar` file), use *the **Extensions** button* in the **Edit scenario** dialog box.

Note: The old way of setting an extension (using the parameter -Dcom.oxygenxml.additional.classpath) was deprecated, and instead you should use the extension mechanism of the XSLT transformation scenario.

XSL-FO Processors

This section explains how to apply XSL-FO processors when transforming XML documents to various output formats in Oxygen XML Editor plugin.

The Built-in XSL-FO Processor

The Oxygen XML Editor plugin installation package is distributed with the *Apache FOP* that is a Formatting Objects processor for rendering your XML documents to PDF. *FOP* is a print and output independent formatter driven by XSL

Formatting Objects. *FOP* is implemented as a Java application that reads a formatting object tree and renders the resulting pages to a specified output.

Other FO processors can be configured in *the **Preferences** dialog box*.

Add a Font to the Built-in FO Processor - 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. (This is not necessary for DITA PDF transformations, skip to the 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 box*, go to **XML** > **XSLT/FO/XQuery** > **FO Processors**, and enter the path of the FOP configuration file in the **Configuration file** text field.
2. Set the font on the document content.

   This is done usually with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

   - For DocBook documents you can start with the predefined scenario called **DocBook PDF**, *edit the XSLT parameters* and set the font name (in our example the font family name is **Arial Unicode MS**) to the parameters body.font.family and title.font.family.
   - For TEI documents you can start with the predefined scenario called **TEI PDF**, *edit the XSLT parameters* and set the font name (in our example **Arial Unicode MS**) to the parameters bodyFont and sansFont.
   - For DITA transformations to PDF using DITA-OT you should modify the following two files:
     - *DITA_OT_DIR*`/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)
     - *DITA_OT_DIR*`/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 FO Processor

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 Editor plugin install directory.
   c) Create the following script file in the Oxygen XML Editor plugin 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 Editor plugin 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) Run the script.

      On Linux and OS X, run the command `sh ttfConvert.sh` from the command line. On Windows, run the command `ttfConvert.bat` from the command line or double-click the file `ttfConvert.bat`.

3. Register the font in FOP configuration. (This is not necessary for DITA PDF transformations, skip to the 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 box*, go to **XML** > **XSLT/FO/XQuery** > **FO Processors**, and enter the path of the FOP configuration file in the **Configuration file** text field.

4. Set the font on the document content.

This is usually done with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

For DocBook documents, you can start with the predefined scenario called **DocBook PDF**, *edit the XSLT parameters*, and set the font name (in our example **Arialuni**) to the parameters body.font.family and title.font.family.
For TEI documents, you can start with the predefined scenario called **TEI PDF**, *edit the XSLT parameters*, and set the font name (in our example **Arialuni**) to the parameters bodyFont and sansFont.
For DITA to PDF transformations using DITA-OT modify the following two files:

- *DITA_OT_DIR*`/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)
- *DITA_OT_DIR*`/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 FO Processor (XML with XSLT and FO)

Adding Hyphenation Support for XML with XSLT Transformation Scenarios

You can extend the functionality of the built-in FO processor by dropping additional libraries in the `[OXYGEN_DIR]/lib/fop` directory. To add support for hyphenation:

1. Download the pre-compiled JAR from *OFFO* .
2. Copy the `fop-hyph.jar` file into the `[OXYGEN_DIR]/lib/fop` folder.
3. Restart Oxygen XML Editor plugin.

Adding Support for PDF Images

1. Download the *`fop-pdf-images`* JAR libraries.
2. Copy the libraries into the `[OXYGEN_DIR]/lib` folder.
3. Restart Oxygen XML Editor plugin.

Adding Libraries to the Built-in FO Processor (DITA-OT)

To use additional libraries with the DITA-OT publishing engine, you need to edit the transformation scenario and add the path to the new libraries in the **Libraries** section of the **Advanced** tab.

Adding Hyphenation Support for DITA-OT Transformation Scenarios

1. Download the pre-compiled JAR from *OFFO* .
2. Edit the DITA-OT transformation scenario and switch to the **Advanced** tab. Click the **Libraries** button and add the path to the `fop-hyph.jar` library.

Adding Support for PDF Images

1. Download the *`fop-pdf-images`* JAR libraries.
2. Edit the DITA-OT transformation scenario and switch to the **Advanced** tab. Click the **Libraries** button and add the path to the libraries.

Output Formats

Oxygen XML Editor plugin allows you to use transformation scenarios to publish XML content in various output formats (such as WebHelp, PDF, CHM, EPUB, JavaHelp, Eclipse Help, XHTML, etc.)

For transformations that are not included in your installed version of Oxygen XML Editor plugin, simply install the tool chain required to perform the specific transformation and process the files in accordance with the processor instructions. A multitude of target formats are possible. The basic condition for transformation to any format is that your source document is well-formed.

Note: You need to use the appropriate stylesheet according to the source definition and the desired output. For example, if you want to transform into an HTML format using a DocBook stylesheet, your source XML document should conform with the DocBook DTD.

For more information, see the *Transformation Scenarios* on page 718 section.

WebHelp System Output

Oxygen XML Editor plugin allows you to obtain WebHelp output from DocBook and DITA documents. This section contains information about the WebHelp system, its variants, and ways to customize it to better fit your specific needs.

WebHelp System Description

WebHelp is a form of online help that consists of a series of web pages (XHTML format). Its advantages include platform independence and continuous content update, since it can be viewed using a regular web browser.

Layout

The layout of the WebHelp system is comprised of two parts:

- The left section that contains separate tabs for **Content**, **Search**, and **Index**.

  Note: If your documents contain no `indexterm` elements, the **Index** tab is not generated.

  Note: You can enhance the appearance of the selected item in the Table of Contents. See the *Customizing WebHelp chapter* for more details.

- The right section where help pages are displayed.

You can navigate through the content of your output using the arrows in the upper-right part of the page. These arrows allow you to move to the parent, previous, and next topic. The parents of the currently opened topic are also presented at the top of the page.

Note: You can edit the `args.hide.parent.link` parameter to hide the **Parent**, **Next**, and **Previous** links.

You can use the **Collapse all** button that is displayed in the **Content** tab to collapse all the topics presented in the Table of Contents.

The top-right corner of the page contains the following options:

- **With Frames** - Displays the output using HTML frames to render two separate sections (a section that displays the Table of Contents in the left side and a section that displays the content of a topic in the right side).
- **Print this page** - Opens a dialog box with various printing options and a print preview.

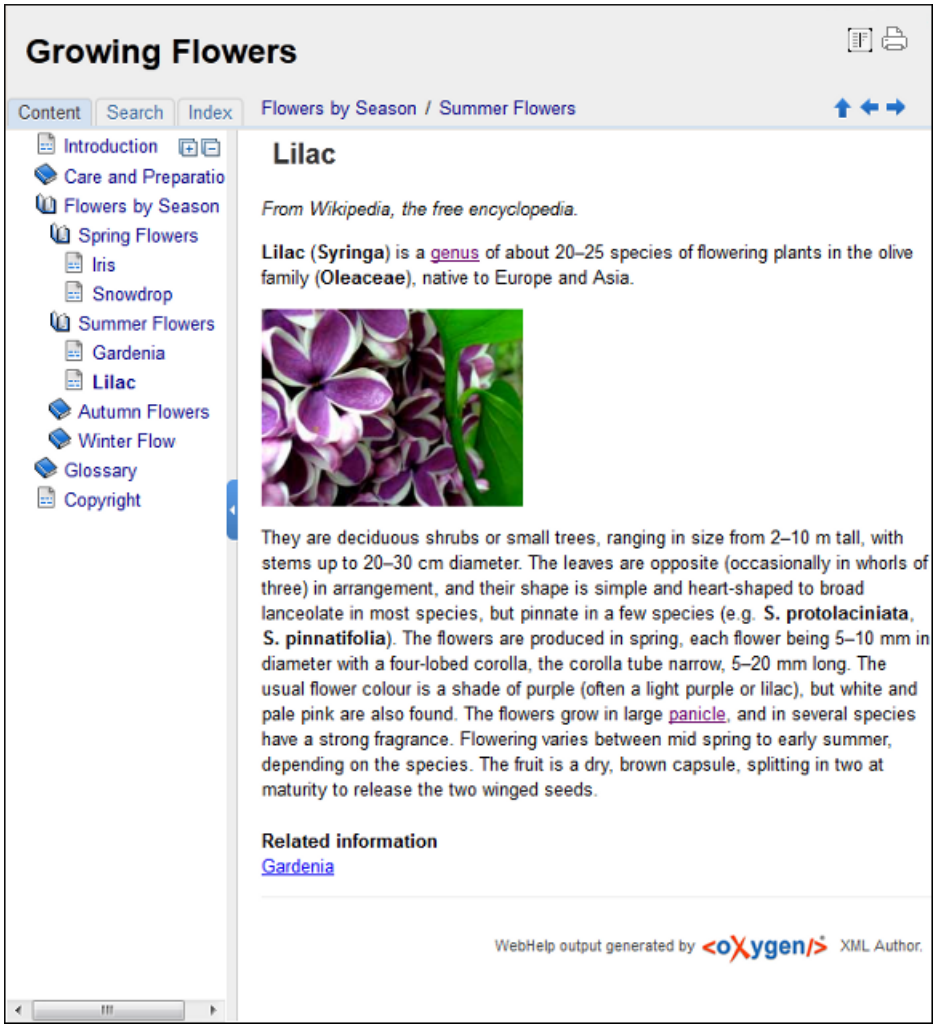


Figure 313: WebHelp Output

Search Feature

The **Search** feature is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on the following:

- The number of keywords found in a single page (the higher the number, the better).
- The context (for example, a word found in a title scores better than a word found in unformatted text). The search ranking order, sorted by relevance is as follows:
  - The search phrase is included in a meta keyword
  - The search phrase is in the title of the page
  - The search phrase is in bold text in a paragraph
  - The search phrase is in normal text in a paragraph

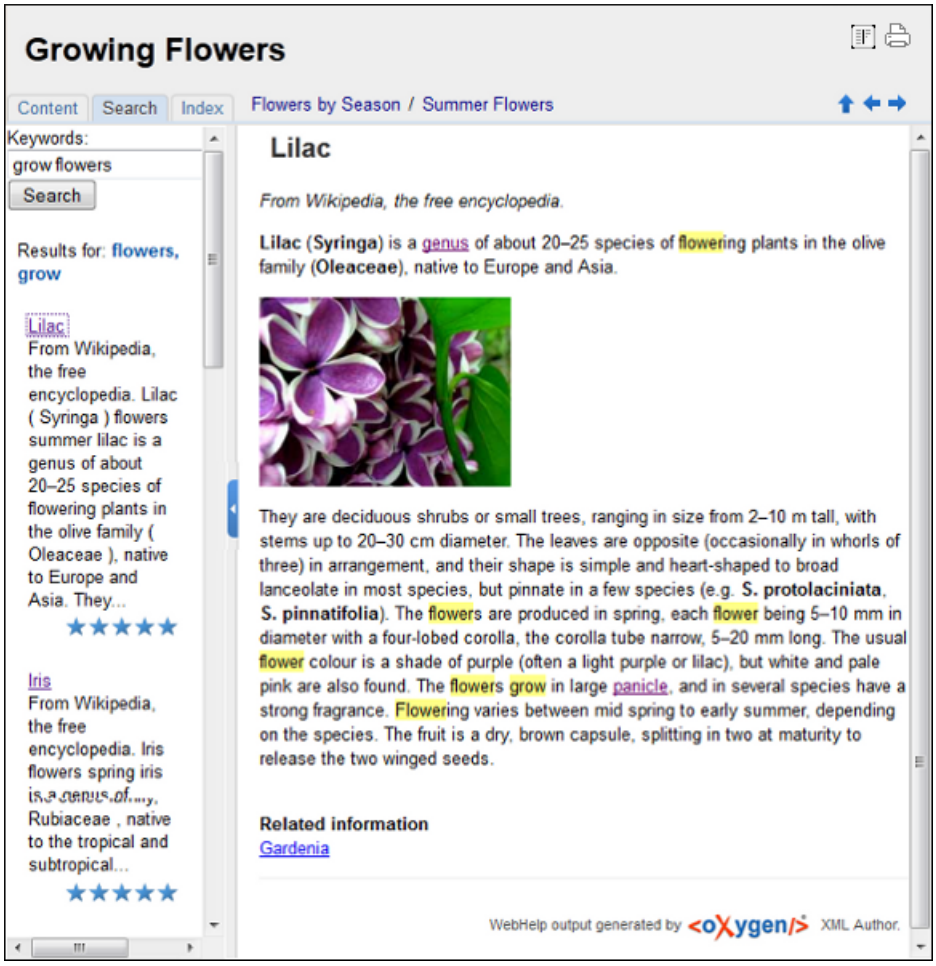


Figure 314: WebHelp Search with Stemming Enabled

Rules that are applied during a search include:

- Boolean searches are supported using the following operators: *and*, *or*, *not*. When there are two adjacent search terms without an operator, *or* is used as the default search operator (for example, *grow flowers* is the same as *grow or flowers*).
- The space character separates keywords (an expression such as *grow flowers* counts as two separate keywords: *grow* and *flowers*).
- Do not use quotes to perform an exact search for multiple word expressions (an expression such as *"grow flowers"*, returns no results since it searches for two separate words).
- `indexterm` and `keywords` DITA elements are an effective way to increase the ranking of a page (for example, content inside *keywords* elements weighs twice as much as content inside an *H1* HTML element).
- Words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters count as a single word.
- Always search for words containing three or more characters (shorter words, such as *to* or *of* are ignored). This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

HTML tag elements are also assigned a scoring value and these values are evaluated for the search results. For information about editing these values, see the *Editing Scoring Values of Tag Elements in Search Results* on page 782 topic.

This output format is compatible with most of the most recent versions of the following common browsers:

- Internet Explorer (IE 8 or newer)
- Chrome
- Firefox

- Safari
- Opera

Important: Due to some security restrictions in Google Chrome, WebHelp pages loaded from the local system (through URLs of the `file:///...` format) may not work properly. We recommend that you load WebHelp pages in Google Chrome only from a web server (with a URL such as `http://your.server.com/webhelp/index.html` or `http://localhost/web_pages/index.html`).

Warning: Due to some restrictions in web browsers in regards to JavaScript code, the *frameless* version (`index.html` start page) of the WebHelp system should only be loaded from a web server (with a URL such as `http://your.server.com/webhelp/index.html` or `http://localhost/web_pages/index.html`). When loading WebHelp pages from the local file system, the *frameset* version (`index_frames.html` start page) of the WebHelp system should be used instead (`file:///...`).

WebHelp with Feedback System Description

WebHelp with Feedback is a form of online help system that consists of a series of web pages (XHTML format). Its advantages include platform independence, continuous content update, and a feedback mechanism that allows your authors and audience to interact with one another through comments.

Layout

The layout of the feedback-enabled WebHelp system is similar to the layout of the basic WebHelp and the left section is identical. However, the bottom of the right section contains a **comments** bar. You can click on the **Log in** button on the right side of this bar to be authenticated as a user for the WebHelp system and your user name will be included in any comments that you add. If you do not have a user name, you can click on the **Sign Up** button to create a new user. Under the **comments** bar, you can click the **Add New Comment** button to add a comment, regardless of whether or not you are logged in.

Note: You can enhance the appearance of the selected item in the Table of Contents. See the *Customizing WebHelp chapter* for more details.

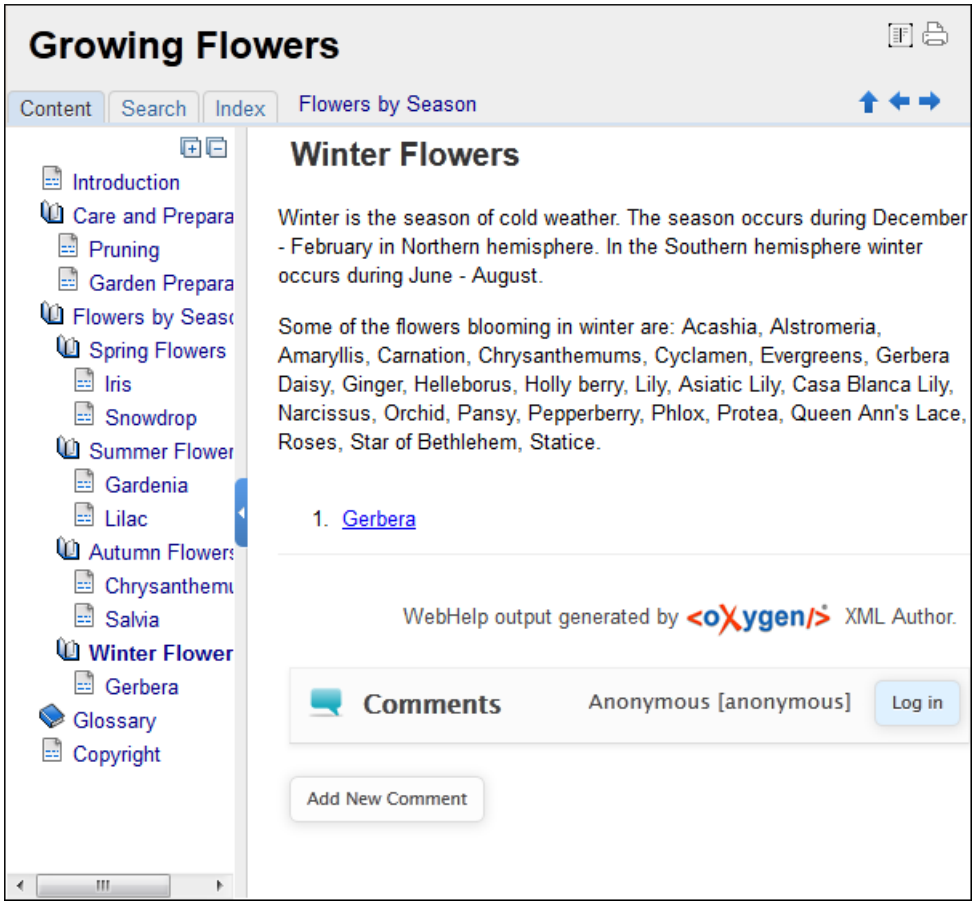


Figure 315: The Layout of the Feedback-Enabled WebHelp System

After you log in, your name and user name are displayed in the **Comments** bar together with the **Log off** and **Edit** buttons. Click the **Edit** button to open the **User Profile** dialog box. In this dialog box, 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 email address** - You can use this field to edit the initial email address that you used to create your profile.
- You can choose to receive an email in the following situations:
  - When a comment is left on a page that you commented on.
  - When a comment is left on any topic in the WebHelp 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 if you want to save the changes you make.

Search Feature

The **Search** feature is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. These scores are then translated into a 5-star rating scheme. The search results are sorted depending on the following:

- The number of keywords found in a single page (the higher the number, the better).
- The context (for example, a word found in a title scores better than a word found in unformatted text). The search ranking order, sorted by relevance is as follows:

- The search phrase is included in a meta keyword
- The search phrase is in the title of the page
- The search phrase is in bold text in a paragraph
- The search phrase is in normal text in a paragraph

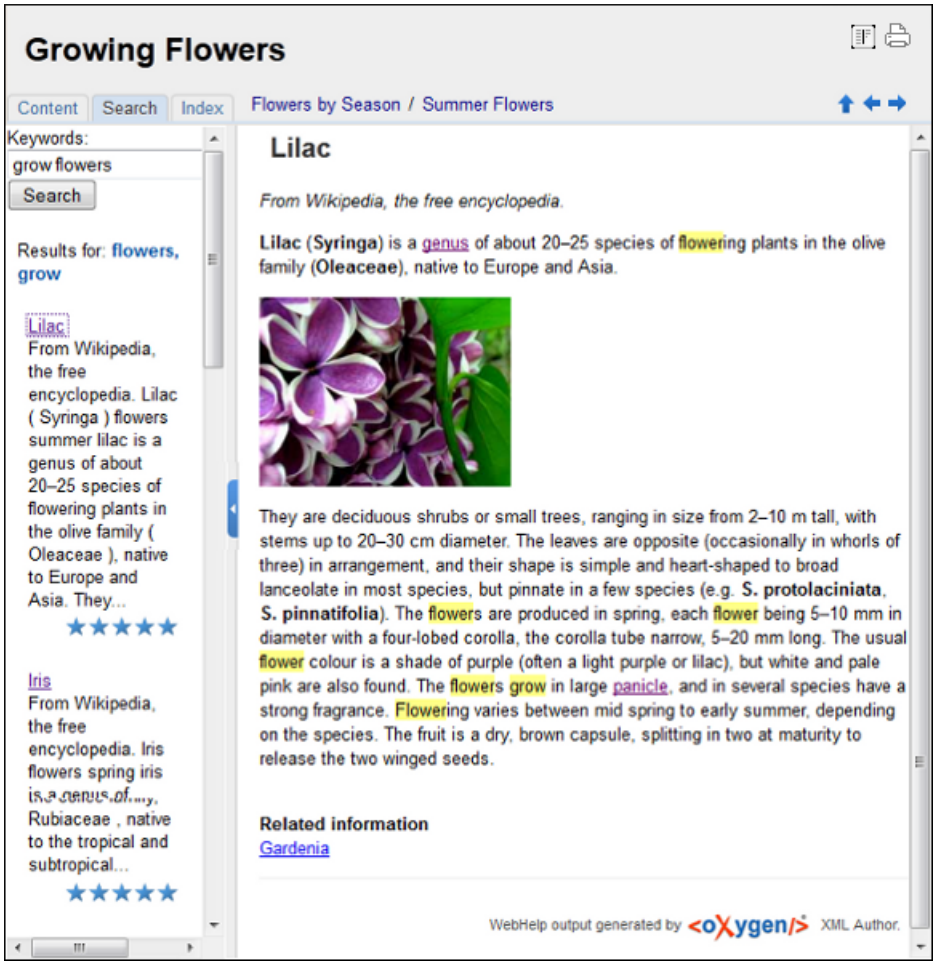


Figure 316: WebHelp Search with Stemming Enabled

Rules that are applied during a search include:

- Boolean searches are supported using the following operators: *and*, *or*, *not*. When there are two adjacent search terms without an operator, *or* is used as the default search operator (for example, *grow flowers* is the same as *grow or flowers*).
- The space character separates keywords (an expression such as *grow flowers* counts as two separate keywords: *grow* and *flowers*).
- Do not use quotes to perform an exact search for multiple word expressions (an expression such as *"grow flowers"*, returns no results since it searches for two separate words).
- `indexterm` and `keywords` DITA elements are an effective way to increase the ranking of a page (for example, content inside *keywords* elements weighs twice as much as content inside an *H1* HTML element).
- Words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters count as a single word.
- Always search for words containing three or more characters (shorter words, such as *to* or *of* are ignored). This rule does not apply to CJK (Chinese, Japanese, Korean) languages.

HTML tag elements are also assigned a scoring value and these values are evaluated for the search results. For information about editing these values, see the *Editing Scoring Values of Tag Elements in Search Results* on page 782 topic.

This output format is compatible with most of the most recent versions of the following common browsers:

- Internet Explorer (IE 8 or newer)
- Chrome
- Firefox
- Safari
- Opera

Deployment of the WebHelp With Feedback System

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Editor plugin requires a standard server deployment. You can request this from your server admin and it needs the following system components:

- A Web server (such as *Apache Web Server*)
- A *MySQL* or *MariaDB* database server
- A database admin tool (such as *phpMyAdmin*)
- PHP Version 5.1.6 or later

Oxygen XML WebHelp system supports most of the recent versions of the following browsers: Chrome, Firefox, Internet Explorer, Safari, Opera.

Create WebHelp with Feedback Database

The WebHelp system needs a database to store user details and the actual feedback, and a user added to it with all privileges. After this is created, you should have the following information:

- Database name
- Username
- Password

Exactly how you create the database and user depends on your web host and your particular needs.

> For example, the following procedure uses *phpMyAdmin* to create a MySQL database for the feedback system and a MySQL user with privileges for that database. The feedback system uses these credentials to connect to the database.
>
> Using *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 it any name you want (for example *comments*).
> 4. Create a user with connection privileges for this database.
> 5. 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

If you have a web server configured with PHP and MySQL, you can deploy the WebHelp output by following these steps:

1. Connect to your server using an FTP client.
2. Locate the home directory (from now on, referred to as *DOCUMENT_ROOT*) of your server.
3. Copy the transformation output folder into the `DOCUMENT_ROOT` folder.
4. Rename it to something relevant, for example, `myProductWebHelp`.

5. Open the output folder (for example, `http://[YOUR_SERVER]/myProductWebHelp/`). You are redirected to the WebHelp installation wizard. Proceed with the installation as follows:
   a. Verify that the prerequisites are met.
   b. Press **Start Installation**.
   c. Configure the **Deployment Settings** section. Default values are provided, but you should adjust them as needed.

      Tip: You can change some of the options later. The installation creates a `config.php` file in `[OXYGEN_WEBHELP]/resources/php/config/config.php` where all your configuration options are stored.

   d. Configure the **MySql Database Connection Settings** section. Use the information (database name, username, password) from the *Create WebHelp with 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. Therefore, it is useful the first time you install the WebHelp with Feedback system, but you do not want to select this option on subsequent deployments.

   e. If you are using a domain (such as *OpenLDAP* or *Active Directory*) to manage users in your organization, check the **Enable LDAP Autehntication** option. This will allow you to configure the LDAP server, which will provide information and credentials for users who will access the WebHelp system. Also, this will allow you to choose which of the domain users will have administrator privileges.
   f. 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. If 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.

   g. Press **Next Step**.
   h. Remove the installation folder from your web server.

      Important: When you publish subsequent iterations of your WebHelp system, you will not upload the `/install` folder in the output, as you only need it uploaded the first time you create the installation. On subsequent uploads, you will just upload the other output files.

   i. In your Web browser, go to your WebHelp system main page.

Testing Your WebHelp with Feedback System

To test your system, create a user and post a comment. Check to see if the notification emails are delivered to your email inbox.

Note: To read debug messages generated by the system:

1. Enable *JavaScript* logging by doing one of the following:
   - Open the `log.js` file, locate the `var log= new Log(Level.NONE);` line, and change the logging level to: `Level.INFO`, `Level.DEBUG`, `Level.WARN`, or `Level.ERROR`.
   - Append `?log=true` to the WebHelp URL.
2. Inspect the PHP and Apache server log files.

Documentation Product ID and Version

When you run a WebHelp with Feedback transformation scenario, by default you are prompted for a documentation product ID and version number. This is needed when multiple WebHelp systems are deployed on the same server. Think of your WebHelp output as a *product*. If you have 3 different WebHelp outputs, you have 3 different *products* (each with their own unique documentation product ID). This identifier is included in a configuration file so that comments are tied to a particular output (product ID and version number).

Note: The WebHelp with Feedback installation includes a configuration option **Display comments from other products** that allows you to choose to have comments be visible in other specified *products*.

Refreshing the Content of a WebHelp with Feedback Installation

It is common to update the content of an existing installation of a WebHelp with Feedback system on a regular basis. In this case, reinstalling the whole system is not a viable option since it might result in the loss of the comments associated with your topics. Also, reconfiguring the system every time you want to refresh it may be time consuming.

However, you can refresh just the content without losing the comments or the initial system configuration, by following these steps:

1. Execute the transformation scenario that produces the WebHelp with Feedback output directory.
2. Locate the `oxygen-webhelp\resources\php\config\config.php` file and delete it.
3. Locate the `oxygen-webhelp\install` directory and delete it.
4. Copy the remaining structure of the output folder and paste it into your WebHelp with Feedback system installation directory, overwriting the existing content.

Managing Users and Comments in a WebHelp with Feedback System

When you installed the WebHelp with Feedback system the first time (assuming the ***Create new database structure option*** was enabled), you should have been prompted to create an administrator account (or a user named `administrator` was created by default). As an administrator, you have access to manage comments posted in your feedback-enabled WebHelp system. You can also manage the user information (such as role, status, or notification options).

To manage comments and user information, follow these steps:

1. At the bottom of each specific topic there is a **Comments** navigation bar and on the right side there is a **Log in** button. Click this button and log in with your administrator credentials. This gives you access to an **Admin Panel** button.
2. Click the **Admin Panel** button to display an administration panel inside the topic.

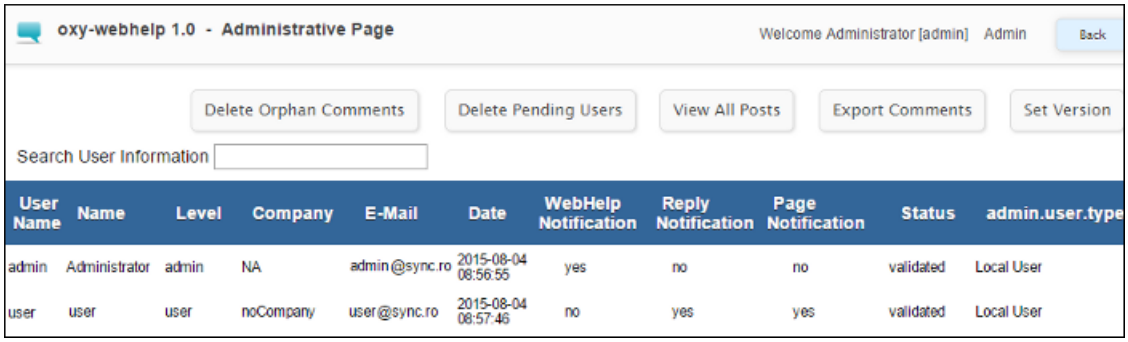


Figure 317: The Administrative Page

3. Use this panel to manage the following options:

   Delete Orphaned Comments

   Allows you to delete comments that are no longer associated with a topic in your WebHelp system.

   Delete Pending Users

   Allows you to delete user accounts that you do not wish to activate.

View All Posts

Allows you to view all the comments that are associated with topics in your WebHelp system.

Export Comments

Allows you to export all posts associated with topics in your WebHelp system into an XML file.

Set Version

Use this action to display comments starting with a particular version.

Manage User Information

To edit the details for a user, click on the corresponding row. This opens a window that allows you to customize the following information associated with the user:

Name

The full name of the user.

Level

Use this field to modify the privilege level (role) for the selected user. You can choose from the following:

- **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** where 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

The name of the organization associated with the user.

E-Mail

The contact email address for the user. 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 your feedback-enabled WebHelp system.

Reply Notification

When enabled, the user receives notifications when comments are posted as a reply to one of their comments.

Page Notification

When enabled, the user receives notifications when comments are posted on a topic where they previously posted a comment.

Date

The date the user registered is displayed.

Status

Use this drop-down list to change the status of the user. You can choose from the following:

- **Created** - The user is created but does not yet have any rights for the feedback-enabled WebHelp system.
- **Validated** - The user is able to use the feedback-enabled WebHelp system.
- **Suspended** - The user has no rights for the feedback-enabled WebHelp system.

Warning: The key used for identifying the page a comment is attached to is the relative file path to the output page. Since the output file and folder names mirror the source, any change to the file name (or its folder) in the source will affect the comments associated with that WebHelp page. If you change the file name or path, the comment history for that topic will become orphaned (a change to the topic ID does not affect the comment history).

WebHelp Mobile System Description

To further improve its ability to create online documentation, Oxygen XML Editor plugin offers support to transform DocBook And DITA documents into *Mobile WebHelp* systems. This feature generates an output that works on multiple platforms (Android, iOS, BlackBerry, Windows Mobile) and is specially designed for mobile devices. All the specific

touch screen gestures are supported. The functionality of the desktop WebHelp layout is preserved, is organized in an intuitive layout, and offers table of contents, search capabilities, and index navigation.

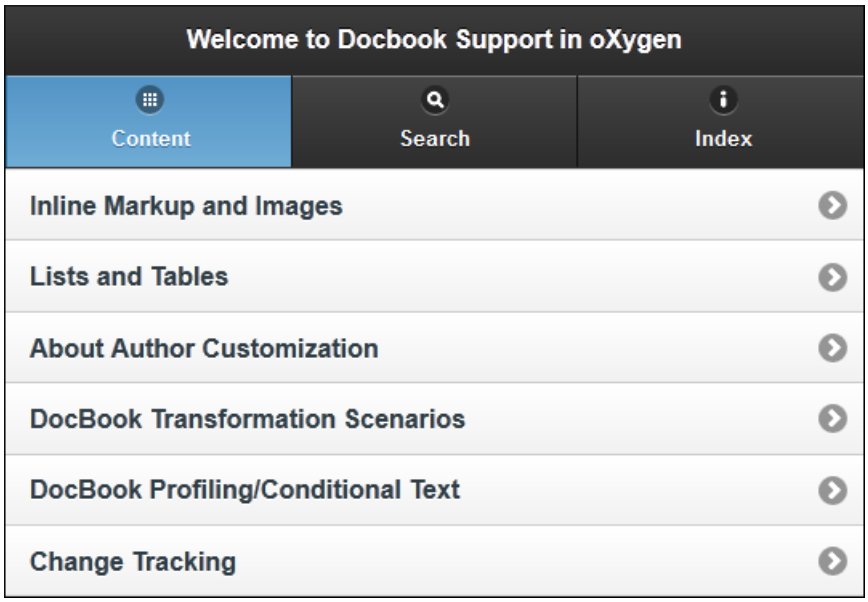


Figure 318: Mobile WebHelp

Context-Sensitive WebHelp System

Context-sensitive help systems assist users by providing specific informational topics for certain components of a user interface, such as a button or window. This mechanism works based on mappings between a unique ID defined in the topic and a corresponding HTML page.

When WebHelp output is generated by Oxygen XML Editor plugin, the transformation process produces an XML mapping file called `context-help-map.xml` and copies it in the output folder of the transformation. This XML file maps an ID to a corresponding HTML page, as in the following example:

```
<map productID="oxy-webhelp" productVersion="1.1">
  <appContext helpID="annotations-view" path="topics/annotations-view.html"/>
  <appContext helpID="button-editor" path="concepts/button-editor.html"/>
    . . .
</map>
```

- `helpID` - Unique ID provided by a topic from two possible sources:
  - The `resourceid` element set to it in the `prolog` section:

    ```
    <prolog>
        <resourceid id="context-sensitive-help-system"/>
    </prolog>
    ```

    Note: If you need different parts of the application (for instance, dialog boxes, views, or editing modes) to open the same contextual help topic, all of the context ID values should be included in the same DITA topic file. For example, if you need both a dialog box and a view to open the same WebHelp page, you can assign different resource IDs in the same DTIA topic.

    ```
    <prolog>
      <resourceid id="dialog1"/>
      <resourceid id="view1"/>
    </prolog>
    ```

  - The `id` attribute set on the topic root element.

    Important: You should ensure that these defined IDs are unique in the context of the entire DITA project. If the IDs are not unique, the transformation scenario will display warning messages in the transformation console output. In this case, the help system will not work properly.

- `path` - Path to a corresponding WebHelp page. This path is relative to the location of the `context-help-map.xml` mapping file.
- `productID` - ID of the product for which you are writing documentation. Applicable only if you are using **WebHelp with Feedback** transformations.
- `productVersion` - Version of the product for which you are writing documentation. Applicable only if you are using **WebHelp with Feedback** transformations.

There are two ways of implementing context-sensitive help in your system:

- The XML mapping file can be loaded by a PHP script on the server side. The script receives the `contextId` value and will look it up in the XML file.
- Invoke one of the WebHelp system files `index.html` or `index_frames.html` and pass the `contextId` parameter with a specific value. The WebHelp system will automatically open the help page associated with the value of the `contextId` parameter.

The following example will open a *frameless* version of the WebHelp system showing the page associated with the id `dialog1ID`:

```
index.html?contextId=dialog1ID
```

The following example will open a *frameset* version of the WebHelp system showing the page associated with the id `view1ID`:

```
index_frames.html?contextId=view1ID
```

Customizing WebHelp Systems

To change the overall appearance of the WebHelp output, you can use the visual *WebHelp Skin Builder tool*, which does not require knowledge of CSS language.

If you are familiar with CSS and coding, this section includes topics that explain how you can customize your WebHelp system, such as how to improve the appearance of the Table of Contents, add logo images in the title area, remove the navigation buttons, and add custom headers and footers.

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 Editor plugin website and allows you to experiment with different styles and colors over an inert documentation sample.

To be able to use the **Skin Builder**, you need:

- An Internet connection and unrestricted access to Oxygen XML Editor plugin website.
- A late version web browser.

To start the **Skin Builder**, do one of the following:

- For DocBook or DITA WebHelp systems, use a web browser to go to *http://www.oxygenxml.com/webhelp-skin-builder*.
- For DITA WebHelp systems, you can click the **Online preview** link in the ***Skins tab*** of a DITA OT transformation scenario. In the upper section of the preview, click the **Select Skin** button, then choose **Customize Skin**.

The Skin Builder Layout

The left side panel of the *Skin Builder* is divided into 3 sections:

- **Actions** - Contains the following two buttons:

  - **Import** - Opens an **Import CSS** dialog box that allows you to load a CSS stylesheet and apply it over the documentation sample.
  - **Export** - Saves all properties as a CSS file.
- **Settings** - Includes a **Highlight selection** option that 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 such as 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

1. The starting point can be either one of the predefined skins or a CSS stylesheet applied over the sample using the **Import** button.
2. Use the elements in the **Customize** section to set properties that modify the look of the skin. By default, all customizable elements display a single property, but you can make more visible by clicking the + **Add** button and choosing from the available properties.

   Note: If you want to revert a particular property to its initial value, press the **Reset** button.
3. When you are happy with the skin customizations 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

1. Start Oxygen XML Editor plugin.
2. Load the DITA map you want to produce as a WebHelp output.
3. Edit a *DITA Map to WebHelp*-type transformation scenario. Set the previously exported CSS file in the **Custom** section of the **Skins** tab.
4. Run the transformation to obtain the WebHelp output.

Apply a Customization Skin to a DocBook to WebHelp Transformation Scenario

1. Start Oxygen XML Editor plugin.
2. Load the DocBook file you want to produce as a WebHelp output.
3. In the **Parameters** tab, set the `webhelp.skin.css` parameter to point to the previously exported CSS.
4. To customize the logo, use the following parameters: `webhelp.logo.image` (not available for the WebHelp Mobile transformation) and `webhelp.logo.image.target.url` (not available for the WebHelp Mobile transformation).
5. Run the transformation to obtain the WebHelp output.

Customizing WebHelp Output with a Custom CSS

By creating your own custom CSS stylesheet, you can customize the look and style of WebHelp output to fit your specific needs.

To use a custom CSS in WebHelp output, follow these steps:

1. *Edit the WebHelp transformation scenario* and open the **Parameters** tab.
   a. For a DITA transformation, set the `args.css` parameter to the path of your custom CSS file. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.
   b. For a DocBook transformation, set the `html.stylesheet` parameter to the path of your custom CSS file.
2. Run the transformation scenario.

Changing the Style of WebHelp Mobile Pages

You can change the style for your WebHelp Mobile pages by setting a custom theme created with a third-party tool.

To create a custom theme for WebHelp Mobile pages, use the following procedure:

1. Create a custom theme (the result will be a CSS stylesheet). Use a designer tool, such as the *ThemeRoller for jQuery Mobile*, to create your own custom theme and then download the resulting CSS stylesheet.

   Tip: If you are using ThemeRoller for jQuery Mobile, make sure you use a type C swatch.

2. *Edit the WebHelp transformation scenario* and open the **Parameters** tab.
   a. For a DITA transformation, set the `args.css` parameter to the path of your custom CSS file. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.
   b. For a DocBook transformation, set the `html.stylesheet` parameter to the path of your custom CSS file.
3. Make sure that the output folder is empty.
4. Run the transformation scenario.

Integrating Social Media and Google Tools in WebHelp Output

Oxygen XML Editor plugin includes support for integrating some of the most popular social media sites in WebHelp output.

How to Add a Facebook Like Button in WebHelp Output

See how you can add a Facebook widget into your WebHelp output.

To add a Facebook™ *Like* widget to your WebHelp output, follow these steps:

1. Go to the *Facebook Developers* website.
2. Fill-in the displayed form, then click the **Get Code** button.
   A dialog box that contains code snippets is displayed.
3. Copy the two code snippets and paste them into a <*div*> element inside an XML file called `facebook-widget.xml`.

   Make sure you follow these rules:

   - The file must be well-formed.
   - The code for each `script` element must be included in an XML comment.
   - The start and end tags for the XML comment must be on a separate line.

   The content of the XML file should look like this:

```
<div id="facebook">
    <div id="fb-root"/>
    <script>
        <!--
            (function(d, s, id) {
        var js, fjs = d.getElementsByTagName(s)[0];
        if (d.getElementById(id)) return;
        js = d.createElement(s); js.id = id;
        js.src = "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
        fjs.parentNode.insertBefore(js, fjs);
        }(document, 'script', 'facebook-jssdk'));
        -->
    </script>
    <div class="fb-like" data-layout="standard" data-action="like" data-show-faces="true"
```

```
        data-share="true"/>
</div>
```

4. In Oxygen XML Editor plugin, click the **Configure Transformation Scenario(s)** action from the toolbar.
5. Select an existing WebHelp transformation scenario (depending on your needs, it may be with or without feedback, or the mobile variant) and click the **Duplicate** button to open the **Edit Scenario** dialog box.
6. Switch to the **Parameters** tab and edit the `webhelp.footer.file` parameter to reference the `facebook-widget.xml` file that you created earlier.
7. Click **Ok**.
8. Run the transformation scenario.

How to Add Tweet Button in WebHelp Output

See how you can add a Twitter widget into your WebHelp output.

To add a Twitter™ *Tweet* widget to your WebHelp output, follow these steps:

1. Go to the *Tweet button generator* page.
2. Fill-in the displayed form.
   The **Preview and code** area displays the code.
3. Copy the code snippet displayed in the **Preview and code** area and paste it into a `div` element inside an XML file called `tweet-button.xml`.

   Make sure you follow these rules:

   - The file must be well-formed.
   - The code for each `script` element must be included in an XML comment.
   - The start and end tags for the XML comment must be on a separate line.

   The content of the XML file should look like this:

```
<div id="twitter">
  <a href="https://twitter.com/share" class="twitter-share-button">Tweet</a>
  <script>
    <!--
      !function (d, s, id) {
      var
      js, fjs = d.getElementsByTagName(s)[0], p = /^http:/.test(d.location) ? 'http': 'https';
      if (! d.getElementById(id)) {
      js = d.createElement(s);
      js.id = id;
      js.src = p + '://platform.twitter.com/widgets.js';
      fjs.parentNode.insertBefore(js, fjs);
      }
      }
      (document,
      'script', 'twitter-wjs');
     -->
  </script>
</div>
```

4. In Oxygen XML Editor plugin, click the **Configure Transformation Scenario(s)** action from the toolbar.
5. Select an existing WebHelp transformation scenario (depending on your needs, it may be with or without feedback, or the mobile variant) and click the **Duplicate** button to open the **Edit Scenario** dialog box.
6. Switch to the **Parameters** tab and edit the `webhelp.footer.file` parameter to reference the `tweet-button.xml` file that you created earlier.
7. Click **Ok**.
8. Run the transformation scenario.

How to Add a Google+ Button in WebHelp Output

See how you can add a Google+ widget into your WebHelp output.

To add a *Google+* widget to your WebHelp output, follow these steps:

1. Go to the *Google Developers* website.
2. Fill-in the displayed form.

The preview area on the right side displays the code and a preview of the widget.

3. Copy the code snippet displayed in the preview area and paste it into a `div` element inside an XML file called `google-plus-button.xml`.

   Make sure that the content of the file is well-formed.

   The content of the XML file should look like this:

```
<div id="google-plus">
  <!-- Place this tag in your head or just before your close body tag. -->
  <script src="https://apis.google.com/js/platform.js" async defer></script>

  <!-- Place this tag where you want the +1 button to render. -->
  <div class="g-plusone" data-annotation="inline" data-width="300"></div>
</div>
```

4. In Oxygen XML Editor plugin, click the **Configure Transformation Scenario(s)** action from the toolbar.
5. Select an existing WebHelp transformation scenario (depending on your needs, it may be with or without feedback, or the mobile variant) and click the **Duplicate** button to open the **Edit Scenario** dialog box.
6. Switch to the **Parameters** tab and edit the `webhelp.footer.file` parameter to reference the `google-plus-button.xml` file that you created earlier.
7. Click **Ok**.
8. Run the transformation scenario.

How to Integrate Google Search in WebHelp Output

See how you can integrate a Google Custom Search Engine into WebHelp output.

You can integrate Google Search into your WebHelp output.

To enable your WebHelp system to use Google Search, follow these steps:

1. Go to the Google Custom Search Engine page using your Google account.
2. Press the **Create a custom search engine** button.
3. Follow the on-screen instructions to create a search engine for your site. At the end of this process you should obtain a code snippet.

   A Google Search script looks like this:

```
<script>
 (function()  {
  var cx =
    '000888210889775888983:8mn4x_mf-yg';
  var gcse = document.createElement('script');
  gcse.type = 'text/javascript';
  gcse.async = true;
  gcse.src = (document.location.protocol == 'https:' ?
    'https:' : 'http:') +         '//www.google.com/cse/cse.js?cx=' + cx;
  var s = document.getElementsByTagName('script')[0];
  s.parentNode.insertBefore(gcse, s);
 }
 )();
</script>
```

4. Save the script into a well-formed HTML file called `googlecse.html`.
5. In Oxygen XML Editor plugin, click the **Configure Transformation Scenario(s)** action from the toolbar.
6. Select an existing WebHelp transformation scenario (depending on your needs, it may be with or without feedback, or the mobile variant) and click the **Duplicate** button to open the **Edit Scenario** dialog box.
7. Switch to the **Parameters** tab and edit the `webhelp.google.search.script` parameter to reference the `googlecse.html` file that you created earlier.
8. You can also use the `webhelp.google.search.results` parameter to choose where to display the search results.
   a) Create an HTML file that contains the Google Custom Search Engine element `gcse:searchresults-only`. It is recommended that you set the `linkTarget` attribute value to `frm` for *frameless* versions of the WebHelp

system or to `contentWin` for *frameset* versions of WebHelp. The default value is `_blank` and if you do not specify a value the search results will be loaded in a new window.

b) Set the value of the `webhelp.google.search.results` parameter to the location of the file you just created. If this parameter is not specified the following code is used: `<gcse:searchresults-only linkTarget="frm"></gcse:searchresults-only>`.

9. Click **Ok**.
10. Run the transformation scenario.

How to Integrate Google Analytics in WebHelp Output

See how you can integrate Google Analytics into WebHelp output.

To enable your WebHelp system to benefit from Google Analytics reports, follow these steps:

1. Create a new Google Analytics account (if you do not already have one) and log on.
2. Choose the Analytics solution that fits the needs of your website.
3. Follow the on-screen instructions to obtain a *Tracking Code* that contains your *Tracking ID*.

   A Tracking Code looks like this:

```
<script>
  (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
  (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
  m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
  })(window,document,'script','//www.google-analytics.com/analytics.js','ga');

  ga('create', 'UA-XXXXXXXX-X', 'auto');
  ga('send', 'pageview');
</script>
```

4. Save the Tracking Code (obtained in the previous step) in a new HTML file called `googleAnalytics.html`.
5. In Oxygen XML Editor plugin, click the **Configure Transformation Scenario(s)** action from the toolbar.
6. Select an existing WebHelp transformation scenario (depending on your needs, it may be with or without feedback, or the mobile variant) and click the **Duplicate** button to open the **Edit Scenario** dialog box.
7. Switch to the **Parameters** tab and edit the `webhelp.footer.file` parameter to reference the `googleAnalytics.html` file that you created earlier.
8. Click **Ok**.
9. Run the transformation scenario.

Localizing the Interface of WebHelp Output

You can localize the interface of WebHelp output for DITA or DocBook transformations.

How to Localize the Interface of WebHelp Output (for DITA Map Transformations)

Static labels that are used in the WebHelp output are kept in translation files in the *DITA_OT_DIR*`/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization` folder. 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.

To localize the interface of the WebHelp output for DITA map transformations, follow these steps:

1. Look for the *strings-[lang1]-[lang2].xml* file in *DITA_OT_DIR*`/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization` directory (for example, the Canadian French file would be: *strings-fr-ca.xml*). If it does not exist, create one starting from the *strings-en-us.xml* file.
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. Make sure that the new XML file that you created in the previous two steps is listed in the file *DITA_OT_DIR*`/plugins/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization/strings.xml`. In our example for the Canadian French file, it should be listed as: `<lang xml:lang="fr-ca" filename="strings-fr-ca.xml"/>`.

4. Edit any of the **DITA Map to WebHelp** transformation scenarios (with or without feedback, or the mobile version) and set the args.default.language parameter to the code of the language you want to localize (for example, *fr-ca* for Canadian French).
5. Run the transformation scenario to produce the WebHelp output.

How to Localize the Interface of WebHelp Output (for DocBook Transformations)

Static labels that are used in the WebHelp output are kept in translation files in the `[OXYGEN_DIR]/frameworks/docbook/xsl/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization` folder. 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.

To localize the interface of the WebHelp output for DocBook transformations, follow these steps:

1. Look for the *strings-[lang1]-[lang2].xml* file in `[OXYGEN_DIR]/frameworks/docbook/xsl/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization` directory (for example, the Canadian French file would be: *strings-fr-ca.xml*). If it does not exist, create one starting from the *strings-en-us.xml* file.
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. Make sure that the new XML file that you created in the previous two steps is listed in the file `[OXYGEN_DIR]/frameworks/docbook/xsl/com.oxygenxml.webhelp/oxygen-webhelp/resources/localization/strings.xml`. In our example for the Canadian French file, it should be listed as: `<lang xml:lang="fr-ca" filename="strings-fr-ca.xml"/>`.
4. Edit any of the DocBook to WebHelp transformation scenarios (with or without feedback, or the mobile version) and set the l10n.gentext.default.language parameter to the code of the language you want to localize (for example, *fr-ca* for Canadian French).
5. Run the transformation scenario to produce the WebHelp output.

Using the Oxygen XML WebHelp Plugin to Automate Output

Oxygen XML WebHelp plugin allows you to use a command line interface script to obtain WebHelp output from DITA and DocBook documents. Note that the Oxygen XML WebHelp plugin is a standalone product with its own licensing terms and cannot be used with a Oxygen XML Editor plugin license.

The WebHelp output files created with the Oxygen XML WebHelp plugin are the same as the output files produced when you run DITA or DocBook to WebHelp transformation scenarios from within Oxygen XML Editor plugin.

When an automated process is required due to the amount of output needed, do the following:

1. Install the Oxygen XML WebHelp plugin.
2. Acquire a Oxygen XML WebHelp license from *http://www.oxygenxml.com/buy_webhelp.html*.
3. Integrate the Oxygen XML WebHelp plugin with *DITA* or *DocBook*.

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 the following transformation types to the DITA Open Toolkit:

- **webhelp** - The transformation that produces *WebHelp* output for desktop.
- **webhelp-feedback** - The transformation that produces feedback-enabled *WebHelp* for desktop.
- **webhelp-mobile** - The transformations that produces *WebHelp* output for mobile devices.

Integrating the Oxygen XML WebHelp Plugin with the DITA Open Toolkit

This topic includes the procedure for 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 as follows:

- Java Virtual Machine 1.6 or later
- DITA Open toolkit 1.6.x, 1.7.x, 1.8, or 2.0 (Full Easy Install, includes Saxon 9.1.0.8 libraries)
- 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, 1.7.x, 1.8.x, or 2.x.
3. Go to *Oxygen XML WebHelp website*, download the latest installation kit, and unzip it.
4. Copy the `com.oxygenxml.webhelp` and `com.oxygenxml.highlight` directories inside the `plugins` directory of the DITA OT distribution. The `com.oxygenxml.highlight` directory adds syntax highlight capabilities to your WebHelp output for `codeblock` sections that contain source code snippets (XML, Java, JavaScript etc.).
5. If you are using DITA-OT version 2.x, the WebHelp plugin contains a `plugin_2.x.xml` which needs to be renamed to `plugin.xml`.
6. In the home directory of the DITA Open Toolkit, run `ant -f integrator.xml`.
7. Go to *http://sourceforge.net/projects/saxon/files/Saxon-B/9.1.0.8/*, download the `processor.saxonb9-1-0-8j.zip` file (contains the Saxon 9.1.0.8), and unzip it to a destination of your choosing.

Licensing the Oxygen XML WebHelp Plugin for DITA OT

This topic explains how to register the license for the Oxygen XML WebHelp plugin for the DITA Open Toolkit.

To register the license for the Oxygen XML WebHelp plugin for the DITA Open Toolkit, follow these steps:

1. Open the `[OXYGEN_DIR]/frameworks/dita/DIT-OT/plugins/com.oxygenxml.webhelp` directory and create a file called `licensekey.txt`.
2. In this file, copy your license key which you purchased for your Oxygen XML WebHelp plugin.

   The WebHelp transformation process reads the Oxygen XML Editor plugin license key from this file. In case the file does not exit, or it contains an invalid license, an error message will be displayed.

Upgrading the Oxygen XML WebHelp Plugin for DITA OT

This topic describes the procedure for upgrading a Oxygen XML WebHelp plugin for the DITA Open Toolkit.

To upgrade your Oxygen XML WebHelp plugin for the DITA Open Toolkit, follow these steps:

1. Navigate to the `plugins` directory of your DITA OT distribution and delete the old Oxygen XML WebHelp plugin files (`oxygen_custom.xsl`, `oxygen_custom_html.xsl`) and directories (`com.oxygenxml.highlight`, `com.oxygenxml.webhelp`).
2. Go to *Oxygen XML WebHelp website*, download the latest installation kit, and unzip it.
3. Copy the `com.oxygenxml.webhelp` and `com.oxygenxml.highlight` directories inside the `plugins` directory of the DITA OT distribution. The `com.oxygenxml.highlight` directory adds syntax highlight capabilities to your WebHelp output for `codeblock` sections that contain source code snippets (XML, Java, JavaScript etc.).
4. If you are using DITA-OT version 2.x, the WebHelp plugin contains a `plugin_2.x.xml` which needs to be renamed to `plugin.xml`.
5. In the home directory of the DITA Open Toolkit, run `ant -f integrator.xml`.

Running an External DITA Transformation Using the Oxygen XML WebHelp Plugin

This topic explains how to run an external DITA to WebHelp 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.

Warning: You can also invoke the DITA OT WebHelp publishing using a startup script but you will lose certain small fixes and patches that Oxygen XML Editor plugin added to the automated DITA OT processing.

The `dita.bat` and the `dita.sh` files are located in the home directory of the Oxygen XML WebHelp Plugin. Before using them to generate a WebHelp system, customize them to match the paths to the JVM, DITA Open Toolkit, and Saxon engine, and also to set the transformation type. To do this, open the script file and edit the following variables and parameters:

- `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.

  Note: The `dita.bat` and `dita.sh` scripts reference the `dost-patches-DITA-1.8.jar` JAR file containing DITA OT 1.8-specific patches. If you use DITA OT 1.7, update that reference to `dost-patches-DITA-1.7.jar`. If you use DITA OT 2.0, no patches are needed, so just remove the reference.

- `SAXON_9_DIR` - Specifies the path to the directory on your disk where you unzipped the Saxon 9 archive files.

  Note: If you are integrating the WebHelp plugin in a DITA OT distribution that comes with the included Saxon libraries, you can point directly to the Saxon libraries in the DITA OT distribution like this:

  ```
  set SAXON_9_DIR=C:\path\to\DITA-OT1.8.5\lib\saxon
  ```

  Otherwise, you can download the Saxon 9.1 JAR files from *https://sourceforge.net/projects/saxon/files/Saxon-B/9.1.0.8/saxonb9-1-0-8j.zip/download*.

- `TRANSTYPE` - Specifies the type of the transformation you want to apply. 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 optional parameter `-Dargs.filter` can be used to specify a filter file to be used to include, exclude, or flag content.

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.sitemap.base.url`

Base URL for all the `loc` elements in the generated `sitemap.xml` file. The value of a `loc` element is computed as the relative file path from the `href` attribute of a `topicref` element from the DITA map, appended to this base URL value. The `loc` element is mandatory in `sitemap.xml`. If you leave this parameter set to its default empty value, then the `sitemap.xml` file is not generated.

`-Dwebhelp.sitemap.change.frequency`

The value of the `changefreq` element in the generated `sitemap.xml` file. The `changefreq` element is optional in `sitemap.xml`. If you leave this parameter set to its default empty value, then the `changefreq` element is not added in `sitemap.xml`. Allowed values: <empty string> (default), `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, `never`.

`-Dwebhelp.sitemap.priority`

The value of the `priority` element in the generated `sitemap.xml` file. It can be set to any fractional number between 0.0 (least important priority) and 1.0 (most important priority), such as: 0.3, 0.5, 0.8, etc. The `priority` element is optional in `sitemap.xml`. If you leave this parameter set to its default empty value, then the `priority` element is not added in `sitemap.xml`.

`-Dwebhelp.copyright`

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

-Dwebhelp.footer.file

Path to an XML file that includes the footer content for your WebHelp output pages. You can use this parameter to integrate social media features (such as widgets for Facebook™, Twitter™, Google Analytics, or Google+™). The file must be well-formed, each widget must be in separate `div` or `span` element, and the code for each `script` element is included in an XML comment (also, the start and end tags for the XML comment must be on a separate line). The following code exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
    <div id="fb-root"/>
    <script>
            <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0]; if (d.getElementById(id))
 return;
             js = d.createElement(s); js.id = id; js.src =
"//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
            fjs.parentNode.insertBefore(js, fjs); }(document, 'script', 'facebook-jssdk')); -->
    </script>
   <div data-share="true" data-show-faces="true" data-action="like" data-layout="standard" class="fb-like"/>
</div>
```

-Dwebhelp.footer.include

Specifies whether or not to include footer in each WebHelp page. Possible values: `yes`, `no`. If set to `no`, no footer is added to the WebHelp pages. If set to `yes` and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then the default Oxygen XML Editor plugin footer is inserted in each WebHelp page.

-Dwebhelp.logo.image (not available for the WebHelp Mobile transformation)

Specifies a path to an image displayed as a logo in the left side of the output header.

-Dwebhelp.logo.image.target.url (not available for the WebHelp Mobile transformation)

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

-Dwebhelp.search.ranking (not available for the WebHelp Mobile transformation)

If this parameter is set to `false` then the *5-star rating mechanism* is no longer included in the search results that are displayed on the **Search** tab (default setting is `true`).

-Dwebhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Editor plugin uses for indexing Japanese content in the WebHelp pages. This indexer does not come bundled with Oxygen XML Editor plugin or the Oxygen XML WebHelp plugin. To use it, you need to download it from *http://mvnrepository.com/artifact/org.apache.lucene/lucene-analyzers-kuromoji/4.0.0* and add it in the *DITA_OT_DIR*`/plugins/com.oxygenxml.webhelp/lib` directory.

-Dwebhelp.google.search.script

Specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google. The value must be an URL.

-Dwebhelp.google.search.results

URL value that specifies the location of a well-formed XHTML file containing the Google Custom Search Engine element `gcse:searchresults-only`. You can use all supported attributes for this element. It is recommend to set the `linkTarget` attribute to `frm` for frameless (*iframe*) version of WebHelp or to `contentWin` for the frameset version of WebHelp. The default value for this attribute is `_blank` and the search results will be loaded in a new window. If this parameter is not specified, the following code will be used `<gcse:searchresults-only linkTarget="frm"></gcse:searchresults-only>`

-Dwebhelp.product.id (available only for the WebHelp with Feedback transformation)

This parameter specifies a short name for the documentation target, or product (for example, `mobile-phone-user-guide`, `hvac-installation-guide`).

Note: You can deploy documentation for multiple products on the same server.

Note: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

`-Dwebhelp.product.version` (available only for the WebHelp with Feedback transformation)

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.

Note: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

`-Dwebhelp.skin.css` (not available for the WebHelp Mobile transformation)

Path to a CSS file that sets the style theme in the output WebHelp pages. It can be one of the predefined skin CSS from the `OXYGEN_INSTALL_DIR\frameworks\docbook\xsl\com.oxygenxml.webhelp\predefined-skins` directory, or it can be a custom skin CSS generated with the *Oxygen Skin Builder* web application.

Note: Note that the `fix.external.refs.com.oxygenxml` parameter is not supported in Oxygen XML WebHelp plugin.

If 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

This topic explains where to find instructions for configuring the database that contains the user comments for a DITA WebHelp with Feedback system.

If you run the **webhelp-feedback** transformation using the WebHelp plugin, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `installation.html` file, located at `[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 the following transformation types to the DocBook XSL distribution:

- **webhelp** - The transformation that produces *WebHelp* output for desktop.
- **webhelp-feedback** - The transformation that produces feedback-enabled *WebHelp* for desktop.
- **webhelp-mobile** - The transformations that produces *WebHelp* output for mobile devices.

Integrating the Oxygen XML WebHelp Plugin with the DocBook XSL Distribution

This topic includes the procedure for 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.

Licensing the Oxygen XML WebHelp Plugin for DocBook

This topic explains how to register the license for the Oxygen XML WebHelp plugin for the DocBook XSL distribution.

To register the license for 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 Editor plugin license key from this file. If the file does not exit, or it contains an invalid license, an error message is displayed.

Upgrading the Oxygen XML WebHelp Plugin for DocBook

This topic describes the procedure for upgrading a Oxygen XML WebHelp plugin for DocBook.

To upgrade your Oxygen XML WebHelp plugin for DocBook, follow these steps:

1. Navigate to the `[OXYGEN_DIR]/frameworks/docbook/xsl` directory and delete the old Oxygen XML WebHelp plugin files (`oxygen_custom.xsl`, `oxygen_custom_html.xsl`) and directory (`com.oxygenxml.webhelp`).
2. Go to *Oxygen XML WebHelp website*, download the latest installation kit, and unzip it.
3. Copy the `com.oxygenxml.webhelp` directory in `[OXYGEN_DIR]/frameworks/docbook/xsl`.

Running an External DocBook Transformation Using the WebHelp Plugin

This topic explains how to run an external DocBook to WebHelp transformation using the Oxygen XML 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 apply. 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 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.

If 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

This topic explains where to find instructions for configuring the database that contains the user comments for a DocBook WebHelp with Feedback system.

If you run the **webhelp-feedback** transformation using the WebHelp plugin, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `installation.html` file, located at `[OUTPUT_DIR]/oxygen-webhelp/resources/installation.html`. The `installation.html` file is created by the transformation process.

Localizing the Email Notifications of the WebHelp with Feedback System

The WebHelp with Feedback system uses emails to notify users when comments are posted. These emails are based on templates stored in the WebHelp directory. The default messages are in English, French, German, and Japanese. These messages are copied into the WebHelp system deployment directory during the execution of the corresponding transformation scenario.

Suppose that you want to localize the emails into Dutch (*nl*). Follow these steps:

DocBook to WebHelp with Feedback

1. Create the following directory:

   `[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl`

2. 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.
3. 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.

4. Start Oxygen XML Editor plugin and edit the *WebHelp with Feedback* transformation scenario.
5. 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

6. Run the transformation scenario to obtain the *WebHelp with Feedback* output.

DITA to WebHelp with Feedback

1. Create the following directory:

   DITA_OT_DIR`\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl`

2. Copy all English template files from *DITA_OT_DIR*`\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\en` and paste them into the directory you just created.
3. Edit the HTML files from the *DITA_OT_DIR*`\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl` directory and translate the content into Dutch.
4. Start Oxygen XML Editor plugin and edit the *WebHelp with Feedback* transformation scenario.
5. 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

6. Run the transformation scenario to obtain the *WebHelp with Feedback* output.

Editing Scoring Values of Tag Elements in Search Results

The WebHelp **Search** feature is enhanced with a rating mechanism that computes scores for every page that matches the search criteria. HTML tag elements are assigned a scoring value and these values are evaluated for the search results. Oxygen XML Editor plugin includes a properties file that defines the scoring values for tag elements and this file can be edited to customize the values according to your needs.

To edit the scoring values of HTML tag element for enhancing WebHelp search results, follow these steps:

1. Edit the scoring properties file for DITA or DocBook WebHelp systems. The properties file includes instructions and examples to help you with your customization.
   a) For DITA WebHelp systems, edit the following file: *DITA_OT_DIR*`\plugins\com.oxygenxml.webhelp\indexer\scoring.properties`.
   b) For DocBook WebHelp system, edit the following file: `[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\indexer\scoring.properties`.

   The values that can be edited in the `scoring.properties` file:

```
h1 = 10
h2 = 9
h3 = 8
h4 = 7
h5 = 6
h6 = 5
b = 5
strong = 5
em = 3
i=3
u=3
div.toc=-10
title=20
div.ignore=ignored
meta_keywords = 20
meta_indexterms = 20
meta_description = 25
shortdesc=25
```

2. Save your changes to the file.
3. Re-run your WebHelp system transformation scenario.

Adding Videos in the Output

Videos can be included and played in all HTML5-based output formats (such as *WebHelp*). For example, to add a YouTube video in the WebHelp output generated from DITA or DocBook documents, follow the procedures below.

Adding Videos to WebHelp Generated from DITA Maps

1. Edit the DITA topic to reference the video using an `object` element, as in the following example:

```
<object outputclass="video">
  <param name="src" value="http://www.youtube.com/watch/v/VideoName"/>
</object>
```

2. Apply a *WebHelp* or *WebHelp with Feedback* transformation scenario to obtain the output.

Adding Videos to WebHelp Generated from DocBook

1. Edit the DocBook document and reference the video using an `mediaobject` element, as in the following example:

```
<mediaobject>
  <videoobject>
    <videodata fileref="http://www.youtube.com/watch/v/VideoName"/>
  </videoobject>
</mediaobject>
```

2. Apply a *WebHelp* or *WebHelp with Feedback* transformation scenario to obtain the output.

Changing the Icons in a WebHelp Table of Contents

You can change the icons that appear in a WebHelp table of contents by assigning new image files in a custom CSS file. By default, the icons for the WebHelp table of contents are defined with the following CSS codes (the first example is the icon that appears for a collapsed menu and the second for an expanded menu):

```
.hasSubMenuClosed{
    background: url('../img/book_closed16.png') no-repeat;
    padding-left: 16px;
    cursor: pointer;
}
```

```
.hasSubMenuOpened{
    background: url('../img/book_opened16.png') no-repeat;
    padding-left: 16px;
    cursor: pointer;
}
```

To assign different icons use the following procedure:

1. Create a custom CSS file that assigns your desired icons to the `.hasSubMenuClosed` and `.hasSubMenuOpened` properties.

```
.hasSubMenuClosed{
    background: url('TOC-my-closed-button.png') no-repeat;
}
```

```
.hasSubMenuOpened{
    background: url('TOC-my-opened-button.png') no-repeat;
}
```

2. It is recommended that you store the image files in the same directory as the default icons.
   a) For DITA transformations: *DITA_OT_DIR*`\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\img\`.
   b) For DocBook transformations: `[OXYGEN_INSTALL_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\img\`.
3. *Edit the WebHelp transformation scenario* and open the **Parameters** tab.

a) For a DITA transformation, set the `args.css` parameter to the path of your custom CSS file. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.
b) For a DocBook transformation, set the `html.stylesheet` parameter to the path of your custom CSS file.

4. Run the transformation scenario.

Customize the Appearance of Selected Items in the Table of Contents

The appearance of selected items in the Table of Contents can be enhanced.

For example, to highlight the background of the selected item, follow these steps:

1. Locate the `toc.css` file in the following directory:
   a. For DITA transformations: *`DITA_OT_DIR`*`\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\css`.
   b. For DocBook transformations: `[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\css`.
2. Edit that CSS file, find the `menuItemSelected` class, and change the value of the `background` property.

Note: You can also overwrite the same value from your own custom CSS and then specify the path to your CSS in the transformation scenario (see step 3 in the *Changing the Icons in a WebHelp Table of Contents* topic.

Adding a Logo Image in the Title Area

You can customize the title area of your WebHelp output by using a custom CSS.

For example, to add a logo image before the title, you could use the following code:

```
#header  h1:before {
    display:inline;
    content:url('oxygen-webhelp/resources/img/myLogoImage.gif');
}
```

In the example above, **myLogoImage.gif** is an image file that you need to store in an images directory from which it will be copied automatically by the WebHelp transformation to the output directory.

It is recommended that you store the image files in the same directory as the default icons.

- For DITA transformations: *`DITA_OT_DIR`*`\plugins\com.oxygenxml.webhelp\oxygen-webhelp\resources\img\`.
- For DocBook transformations: `[OXYGEN_INSTALL_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\img\`.

Then you need to specify the path of your custom CSS in the transformation scenario.

Edit the WebHelp transformation scenario and open the **Parameters** tab.

- For a DITA transformation, set the `args.css` parameter to the path of your custom CSS file. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.
- For a DocBook transformation, set the `html.stylesheet` parameter to the path of your custom CSS file.

Removing the *Previous*/*Next* Links from WebHelp Pages

The **Previous** and **Next** links that are created at the top area of each WebHelp page can be hidden with a CSS code.

To remove these links from WebHelp pages, follow these steps:

1. Add the following CSS code in a custom CSS stylesheet:

```
.navparent, .navprev, .navnext {
    visibility:hidden;
}
```

2. *Edit the WebHelp transformation scenario* and open the **Parameters** tab.

a. For a DITA transformation, set the `args.css` parameter to the path of your custom CSS file. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.
b. For a DocBook transformation, set the `html.stylesheet` parameter to the path of your custom CSS file.

3. Run the transformation scenario.

Customizing the Header and Footer

In the transformation scenario, you can use the `args.hdr` and `args.ftr` parameters to point to resources that contain your custom HTML `<div>` blocks. These are included in the header and footer of each generated topic.

As a practical example, to hide the horizontal separator line between the content and footer, follow these steps:

1. Create a custom CSS file that contains the following snippet:

```
.footer_separator {
  display:none;
}
```

2. *Edit the WebHelp transformation scenario* and open the **Parameters** tab.

a. For a DITA transformation, set the `args.css` parameter to the path of your custom CSS file. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.
b. For a DocBook transformation, set the `html.stylesheet` parameter to the path of your custom CSS file.

3. Run the transformation scenario.

Adding a *Favicon* in WebHelp Systems

You can add a custom *favicon* to your WebHelp system by simply editing a template in an XSL file. To add a *favicon*, follow these steps:

1. Edit the following XSL file for DITA or DocBook WebHelp systems:
   a) For DITA WebHelp systems, edit the following file: *`DITA_OT_DIR`*`\plugins\com.oxygenxml.webhelp\xsl\createMainFiles.xsl`.
   b) For DocBook WebHelp system, edit the following file: `[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\xsl\createMainFiles.xsl`.
2. Locate the following template in the XSL file: `<xsl:template name-"create-toc-common-file">`.
3. Add two `link` elements inside the `head` element, as in the following example:

```
<link rel="icon" href="/favicon.ico" type="image/x-icon" />
<link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" />
```

4. Save your changes to the file.
5. Re-run your WebHelp system transformation scenario.

Change Numbering Styles for Ordered Lists

Ordered lists (`ol`) are usually numbered in XHTML output using numerals. If you want to change the numbering to alphabetical, follow these steps:

1. Define a custom `outputclass` value and set it as an attribute of the ordered list, as in the following example:

```
<ol outputclass="number-alpha">
    <li>A</li>
    <li>B</li>
    <li>C</li>
</ol>
```

2. Add the following code snippet in a custom CSS file:

```
ol.number-alpha{
    list-style-type:lower-alpha;
}
```

3. *Edit the WebHelp transformation scenario* and open the **Parameters** tab.
   a. For a DITA transformation, set the `args.css` parameter to the path of your custom CSS file. Also, set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.
   b. For a DocBook transformation, set the `html.stylesheet` parameter to the path of your custom CSS file.
4. Run the transformation scenario.

WebHelp Runtime Additional Parameters

A deployed WebHelp system can accept the following GET parameters:

- `log` - The value can be `true` or `false` (default value). When set to `true`, it enables JavaScript debugging.
- `contextId` - The WebHelp JavaScript engine will look up the value of this parameter in the mapping file and load the corresponding HTML help page. For more information, see the *Context-Sensitive WebHelp System* topic.

  Note: You can use an *anchor* in the `contextId` parameter to jump to a specific section in a document. For example, `contextId=topicID#anchor`.

- `toc.visible` - The value can be `true` (default value) or `false`. When set to `false`, the table of contents will be collapsed when you load the WebHelp page.
- `searchQuery` - You can use this parameter to perform a search operation when WebHelp is loaded. For example, if you want to open WebHelp showing all search results for *growing flowers*, the URL should look like this: `http://localhost/webhelp/index.html?searchQuery=growing%20flowers`.

Disable Caching in WebHelp Pages

Sometimes, a set of WebHelp pages needs to be updated often in order to deliver the latest version of the documentation to your users. In such cases, a WebHelp page should always be requested from the server upon re-loading it in a Web browser on the client side, rather than re-using an outdated *cached* version in the browser.

To disable caching in WebHelp pages, follow this procedure:

1. Edit the following XSL file for DITA or DocBook WebHelp systems:
   - For DITA WebHelp systems, edit the following file: *DITA_OT_DIR*`\plugins\com.oxygenxml.webhelp\xsl\createMainFiles.xsl`.
   - For DocBook WebHelp system, edit the following file: `[OXYGEN_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\xsl\createMainFiles.xsl`.
2. Locate the following template in the XSL file: `<xsl:template name-"create-toc-common-file">` and add the following code snippet:

```
<meta http-equiv="Pragma" content="no-cache" />
<meta http-equiv="Expires" content="-1" />
```

Note: The code should look like this:

```
  <html>
    <head>
      <xsl:if test="$withFrames">
      <base target="contentwin"/>
      </xsl:if>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

          <!-- Disable caching of WebHelp pages in web browser. -->

      <meta http-equiv="Pragma" content="no-cache" />
      <meta http-equiv="Expires" content="-1" />
....
```

3. Save your changes to the file.
4. Re-run your WebHelp system transformation scenario.

Adding a Button in Code Snippet Areas

This task will get you started on how to add an action (such as a button or link) in code snippet areas that are displayed in WebHelp output. You can then attach your code that does the actual processing for the action.

Follow these steps:

1. Open the *`DITA_OT_DIR`*`\plugins\org.dita.xhtml\xsl\xslhtml\dita2htmlImpl.xsl` file.
2. Locate the `<xsl:template match="*[contains(@class, ' topic/pre ')]" mode="pre-fmt">` template to check the default behavior of this template.
3. Open the *`DITA_OT_DIR`*`\plugins\com.oxygenxml.webhelp\xsl\dita\desktop\fixup.xsl` file.
4. Create a `<xsl:template match="*[contains(@class, ' topic/pre ')]" mode="pre-fmt">` template to override the default processing.
5. This new template will include your code for creating the button, which will have the action code that does the actual processing attached to it (this can be written in JavaScript, for example).

   Example of a *Select all* button:

```
<xsl:template match="*[contains(@class, ' topic/pre ')]" mode="pre-fmt">
    <button onclick="SelectText(element)">Select all</button>
    <script>
      function SelectText(element) {
        var text = document.getElementById(element);
        var range = document.body.createTextRange();
        range.moveToElementText(text);
        range.select();
      }
    </script>
  </xsl:template>
```

Flag DITA Content

Flagging content involves defining a set of images that will be used for marking content across your information set.

To flag DITA content, you need to create a filter file that defines properties that will be applied on elements to be flagged. Generally, flagging is supported for block-level elements (such as paragraphs), but not for phrase-level elements within a paragraph. This ensures that the images that will flag the content are easily scanned by the reader, instead of being buried in text.

Follow this procedure:

1. Create a DITA filter file in the directory where you want to add the file. Give the file a descriptive name, such as `audience-flag-build.ditaval`.
2. Define the property of the element you want to be flagged. For example, if you want to flag elements that have the `audience` attribute set to `programmer`, the content of the DITAVAL file should look like in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<val>
  <prop att="audience" val="programmer" action="flag" img="D:\resource\delta.gif" alt="sample alt text"/>
</val>
```

   Note that for an element to be flagged, at least one attribute-value pair needs to have a property declared in the DITAVAL file.

3. Specify the DITAVAL file in the **Filters** tab of the transformation scenario.
4. Run the transformation scenario.

Chapter 11

Querying Documents

Topics:

This chapter shows how to query XML documents in Oxygen XML Editor plugin with XPath expressions and the XQuery language.

Running XPath Expressions

This section covers the views, toolbars, and dialog boxes in Oxygen XML Editor plugin that are 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*.

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 Editor plugin executes the expressions over the XML document in the associated scenario.

To open the **XPath/XQuery Builder** view, go to **Window** > **Show View** > **XPath/XQuery Builder**.

The upper part of the view contains the following actions:

- A drop-down menu 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 Editor plugin uses Saxon to execute XPath 3.0 expressions. Since Saxon implements a part of the 3.0 functions, when using a function that is not implemented, Oxygen XML Editor plugin 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 menu next to the star button to see all your favorites. Oxygen XML Editor plugin automatically groups favorites in folders named after the method of execution

- **History** drop-down menu - 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 the following three options:
  - **Update on caret move** - When enabled and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed.
  - **Evaluate as you type** - When you select this option, the XPath expression you are composing is evaluated in real time.

    Note: The **Evaluate as you type** option and the automatic validation are disabled when the scope is other than **Current file**.

  - **Options** - Opens the Preferences page of the currently selected processing engine.
- **XPath scope** menu - Oxygen XML Editor plugin allows you to define a scope on which the XPath operation will be executed. You can choose where the XPath expression will be executed:
  - **Current file** - Current selected file only.
  - **Enclosing project** - All the files of the project that encloses the current edited file.
  - **Workspace selected files** - The files selected in the workspace. The files are collected from the last selected resource provider view (**Navigator**, **Project Explorer** or **Package Explorer**).
  - **All opened files** - All files opened in the application.
  - **Current DITA Map hierarchy** - All resources referenced in the currently selected DITA map, opened in the **DITA Maps Manager** view.
  - **Opened archive** - Files open in the *Archive Browser* view.
  - **Working sets** - The selected working sets.

  At the bottom of the scope menu there are available the following scope configuration actions:
  - **Configure XPath working sets** - Allows you to configure and manage collections of files and folders, encapsulated in logical containers called *working sets*.
  - **XPath file filter** - You can filter the files from the selected scope on which the XPath expression will be executed. By default the XPath expression will be executed only on XML files, but you can also define a set of patterns that will filter out files from the current scope.

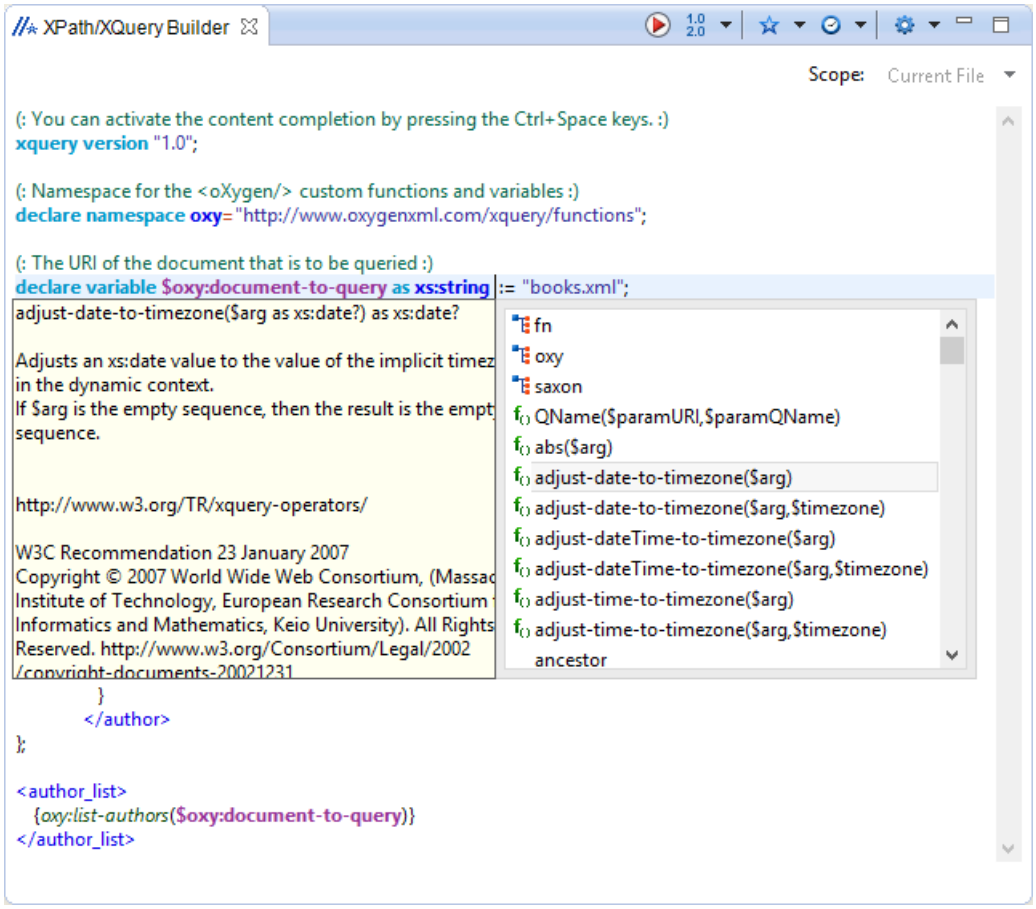


Figure 319: The XPath/XQuery Builder View

When you hover your cursor over the XPath/XQuery version icon 1.0 2.0, a tooltip is displayed to let you know what engine Oxygen XML Editor plugin currently uses.

While you edit an XPath or XQuery expression, Oxygen XML Editor plugin assists you with the following features:

- Content Completion Assistant - It offers context-dependent proposals and takes into account the cursor position in the document you are editing. The set of functions proposed by the **Content Completion Assistant** also depends on the engine version. Select the engine version from the drop-down menu available in the toolbar.
- Syntax Highlight - Allows you to identify the components of an expression. To customize the colors of the components of the expression, *open the **Preferences** dialog box* and go to ***Editor > Syntax Highlight***.
- Automatic validation of the expression as you type.

  Note: When you type invalid syntax a red serrated line underlines the invalid fragments.

- Function signature and documentation balloon, when the cursor is located inside a function.

XPath Expression Results

When you run an XPath expression, Oxygen XML Editor plugin displays the results of its execution in the **XPath Results** view.

This view contains the following columns:

- **Description** - Holds the result thatOxygen XML Editor plugin displays when you run an XPath expression.
- **XPath location** - Holds the path to the matched node.
- **Resource** - Holds the name of the document on which you run the XPath expression.

- **System ID** - Holds the path to the document itself.
- **Location** - Holds the location of the result in the document.

To arrange the results depending on a column, click its header. If no information regarding location is available, Oxygen XML Editor plugin displays **Not available** in the **Location** column. Oxygen XML Editor plugin displays the results in a valid XPath expression format.

```
- /node[value]/node[value]/node[value] -
```

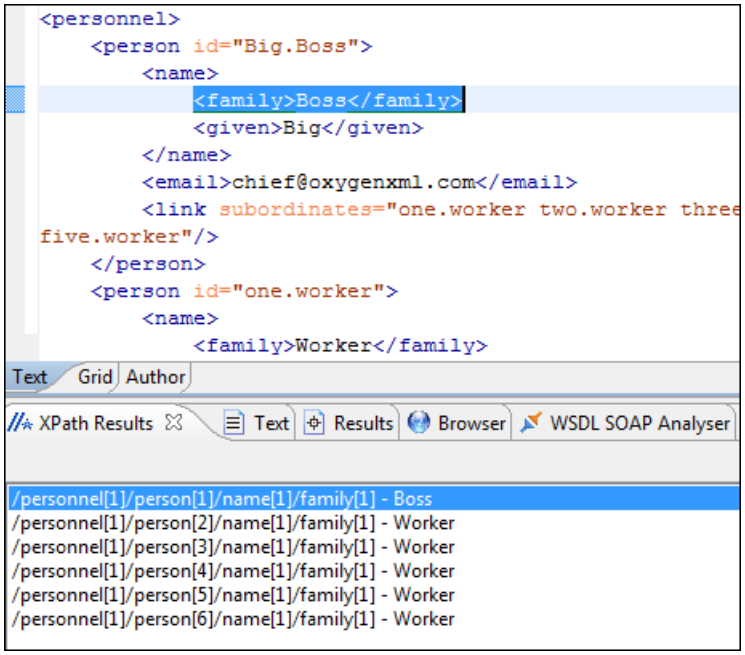


Figure 320: XPath results highlighted in editor panel with character precision

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 Editor plugin adds a result in the **Results View** for each `example` node found in any `sect2` node.

For example, if the result of the above XPath expression is:

```
- /chapter[1]/sect1[3]/sect2[7]/example[1]
```

it means that in the edited file the `example` node is located in the first chapter, third section level one, seventh section level 2.

Catalogs

The evaluation of the XPath expression tries to resolve the locations of documents referenced in the expression through the *XML catalogs*. These catalogs are *configured in the Preferences* pages and *the current XInclude preferences*.

As an example, consider 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 Editor plugin binds this namespace to the first free prefix from the list: `default`, `default1`, `default2`, and so on. For example, if you define the default namespace `xmlns="something"` and the prefix `default` is not associated with another namespace, you can match tags without prefix in an XPath expression typed in the XPath toolbar by using the prefix `default`. To find all the `level` elements when you define a default namespace in the root element, use this expression: `//default:level` in the XPath toolbar.

Working with XQuery

This section explains how to edit and run XQuery queries in Oxygen XML Editor plugin.

What is XQuery

XQuery is the query language for XML and is officially defined by *a W3C Recommendation document*. The many benefits of XQuery include:

- XQuery allows you to work in one common model no matter what type of data you're working with: relational, XML, or object data.
- XQuery is ideal for queries that must represent results as XML, to query XML stored inside or outside the database, and to span relational and XML sources.
- XQuery allows you to create many different types of XML representations of the same data.
- XQuery allows you to query both relational sources and XML sources, and create one XML result.

Syntax Highlight and Content Completion

To *create an XQuery document*, select **File** > **New** **(Ctrl (Meta on Mac OS)+N)** and when the **New** document wizard appears, select XQuery entry.

Oxygen XML Editor plugin provides syntax highlight for keywords and all known XQuery functions and operators. A content completion assistant is also available and can be activated with the **(Ctrl (Meta on Mac OS)+Space)** shortcut. The functions and operators are presented together with a description of the parameters and functionality. For some supported database engines like eXist and Berkeley DB, the content completion list offers the specific XQuery functions implemented by that engine. This feature is available when the XQuery file has an associated transformation scenario which uses one of these database engines or the XQuery validation engine is set to one of them via a validation scenario or in the *XQuery Preferences* page.

The extension functions built in the Saxon product are available on content completion if one of the following conditions are true:

- The edited file has a transformation scenario associated that uses as transformation engine Saxon 9.6.0.7 PE or Saxon 9.6.0.7 EE.
- The edited file has a validation scenario associated that use as validation engine Saxon 9.6.0.7 PE or Saxon 9.6.0.7 EE.

- The validation engine specified in *Preferences* is Saxon 9.6.0.7 PE or Saxon 9.6.0.7 EE.

If the Saxon namespace (*http://saxon.sf.net*) is mapped to a prefix the functions are presented using this prefix, the default prefix for the Saxon namespace (`saxon`) is used otherwise.

If you want to use a function from a namespace mapped to a prefix, just type that prefix and the content completion displays all the XQuery functions from that namespace. When the default namespace is mapped to a prefix the XQuery functions from this namespace offered by content completion are also prefixed, only the function name being used otherwise.

The content completion popup window presents all the variables and functions from both the edited XQuery file and its imports.

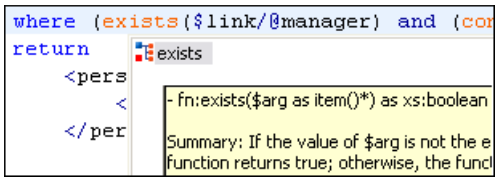


Figure 321: XQuery Content Completion

XQuery Outline View

The XQuery document structure is presented in the XQuery **Outline** view. The outline tree presents the list of all the components (namespaces, imports, variables, and functions) from both the edited XQuery file and its imports. It allows quick access to components and can be opened from **Window** > **Show View** > **Outline**.

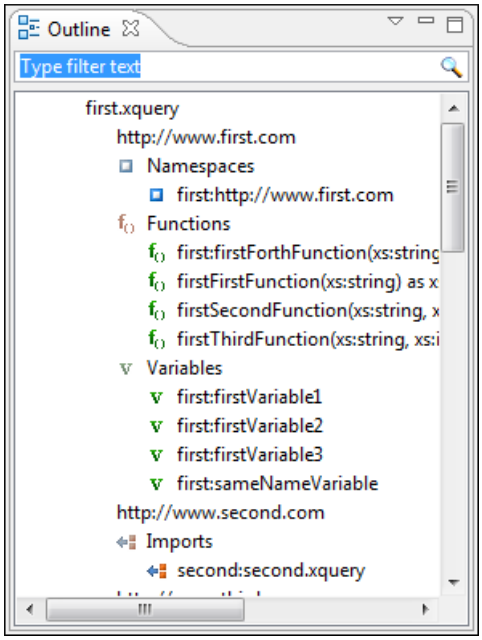


Figure 322: XQuery Outline View

The following actions are available in the **View menu** on the **Outline** view action bar:

Selection update on cursor move

Controls the synchronization between **Outline** view and source document. The selection in the **Outline** view can be synchronized with the cursor moves or the changes performed in the XQuery editor. Selecting one of the components from the **Outline** view also selects the corresponding item in the source document.

Sort

Allows you to alphabetically sort the XQuery components.

Show all components

Displays all collected components starting from the current file. This option is set by default.

Show only local components

Displays the components defined in the current file only.

Group by location/namespace/type

Allows you to group the components by location, namespace, and type. When grouping by namespace, the main XQuery module namespace is presented first in the **Outline** view.

If you know the component name, you can search it in the **Outline** view by typing its name in the filter text field from the top of the view or directly on the tree structure. When you type the component name in the filter text field you can switch to the tree structure using the arrow keys of the keyboard, **(Enter)**, **(Tab)**, **(Shift-Tab)**. To switch from tree structure to the filter text field, you can use **(Tab)**, **(Shift-Tab)**.

Tip: The search filter is case insensitive. The following wildcards are accepted:

- ***** - any string
- **?** - any character
- **,** - patterns separator

If no wildcards are specified, the string to search is used as a partial match (like `*textToFind*`).

The upper part of the **Outline** view contains a filter box that allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (*, ?) and separate multiple patterns with commas.

The XQuery Input View

You are able to drag and drop a node in the editing area to insert XQuery expressions quickly.

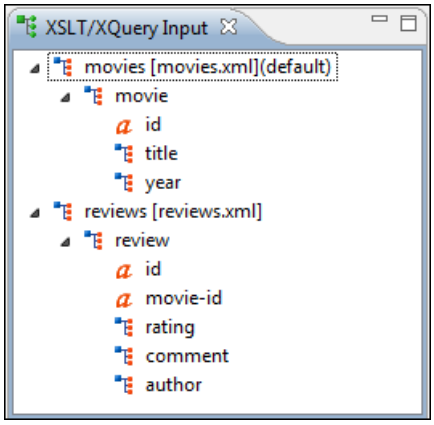


Figure 323: XQuery Input View

Create FLWOR by Drag and Drop

For the following XML documents:

```
<movies>
<movie id="1">
<title>The Green Mile</title>
<year>1999</year>
</movie>
<movie id="2">
<title>Taxi Driver</title>
<year>1976</year>
</movie>
</movies>
```

and

```
<reviews>
<review id="100" movie-id="1">
<rating>5</rating>
<comment>It is made after a great Stephen King book.
</comment>
<author>Paul</author>
</review>
<review id="101" movie-id="1">
<rating>3</rating>
<comment>Tom Hanks does a really nice acting.</comment>
<author>Beatrice</author>
</review>
<review id="104" movie-id="2">
<rating>4</rating>
<comment>Robert De Niro is my favorite actor.</comment>
<author>Maria</author>
</review>
</reviews>
```

and the following XQuery:

```
let $review := doc("reviews.xml")
for $movie in doc("movies.xml")/movies/movie
let $movie-id := $movie/@id
return
<movie id="{$movie/@id}">
{$movie/title}
{$movie/year}
<maxRating>
{

}
</maxRating>
</movie>
```

if you drag the **review** element and drop it between the braces, a pop-up menu will be displayed.

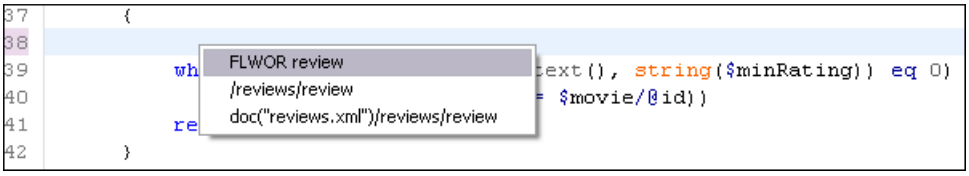


Figure 324: XQuery Input Drag and Drop Pop-up Menu

Select **FLWOR rating** and the result document will be:

```
{
    for $review in doc("reviews.xml")/reviews/review
    return
    where ((compare($rev/rating/text(), string($minRating)) eq 0)
            and ($rev/@movie-id = $movie/@id))
    return $rev/author
}
```

Figure 325: XQuery Input Drag and Drop Result

XQuery Validation

With Oxygen XML Editor plugin, you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.6.0.7 PE processor or the 9.6.0.7 EE, IBM DB2, eXist, Berkeley DB XML, Documentum xDb (X-Hive/DB) 10, or MarkLogic (version 5 or newer) if you installed them. Any other XQuery processor that offers an *XQJ API implementation* can also be used. This is in conformance with *the XQuery Working Draft*. The processor is used in two cases: validation of the expression and execution. Although the execution implies a validation, it is faster to check the expression syntactically, without executing it. The errors that occurred in the document are presented in

the messages view at the bottom of editor window, with a full description message. As with all error messages, if you click an entry, the line where the error appeared is highlighted.

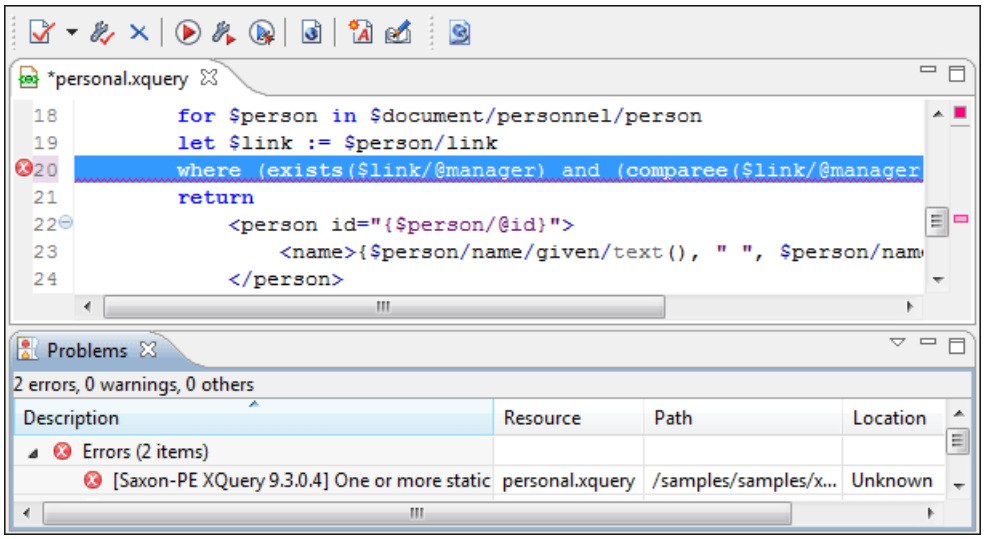


Figure 326: XQuery Validation

Note: If you choose a processor that does not support XQuery validation, Oxygen XML Editor plugin displays a warning when trying to validate.

When you open an XQuery document from a connection that supports validation (for example MarkLogic, or eXist), by default Oxygen XML Editor plugin uses this connection for validation. If you open an XQuery file using a MarkLogic connection, the validation better resolves imports.

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 box* ,or in *the **Scenarios** view*. The transformation can be performed on the XML document specified in the **XML URL** field, or, if this field is empty, the documents referenced from the query expression. The parameters of XQuery transforms must be set in *the **Parameters** dialog box*. Parameters that are in a namespace must be specified using the qualified name, for example a `param` parameter in the *http://www.oxygenxml.com/ns* namespace must be set with the name `{http://www.oxygenxml.com/ns}param`.

The transformation uses one of the Saxon 9.6.0.7 HE, Saxon 9.6.0.7 PE, Saxon 9.6.0.7 EE processors, a database connection (details can be found in the *Working with Databases* chapter - in the *XQuery transformation* section) or any XQuery processor that provides an XQJ API implementation.

The Saxon 9.6.0.7 EE processor supports also XQuery 3.0 transformations.

XQJ Transformers

This section describes the necessary procedures before running an XQJ transformation.

How to Configure an XQJ Data Source

Any transformer that offers an XQJ API implementation can be used when validating XQuery or transforming XML documents. An example of an XQuery engine that implements the XQJ API is *Zorba*.

1. If 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 Editor plugin after configuring the environment variables.
2. *Open the **Preferences** dialog box* and go to **Data Sources**.

3. Click the **New** button in the **Data Sources** panel.
4. Enter a unique name for the data source.
5. Select **XQuery API for Java(XQJ)** in the **Type** combo box.
6. Press the **Add** button to add XQJ API-specific files.

   You can manage the driver files using the **Add**, **Remove**, **Detect**, and **Stop** buttons.

   Oxygen XML Editor plugin detects any implementation of `javax.xml.xquery.XQDataSource` and presents it in **Driver class** field.
7. Select the most suited driver in the **Driver class** combo box.
8. Click the **OK** button to finish the data source configuration.

How to Configure an XQJ Connection

The steps for configuring an XQJ connection are the following:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for this connection.
4. Select one of the previously configured *XQJ data sources* in the **Data Source** combo box.
5. Fill-in the connection details.

   The properties presented in the connection details table are automatically detected depending on the selected data source.
6. Click the **OK** button.

Display Result in Sequence View

The result of an XQuery executed on a database can be very large and sometimes only a part of the full result is needed. To avoid the long time necessary for fetching the full result, select the **Present as a sequence** option in the **Output** tab of the **Edit scenario** dialog box. This option fetches only the first chunk of the result. Clicking the **More results available** label that is displayed at the bottom of the **Sequence** view fetches the next chunk of results.

The size of a chunk can be *set with the option **Size limit of Sequence view***. The **XQuery options** button from the **More results available** label provides a quick access to this option by opening *the **XQuery** panel of the **Preferences** dialog box* where the option can be modified.

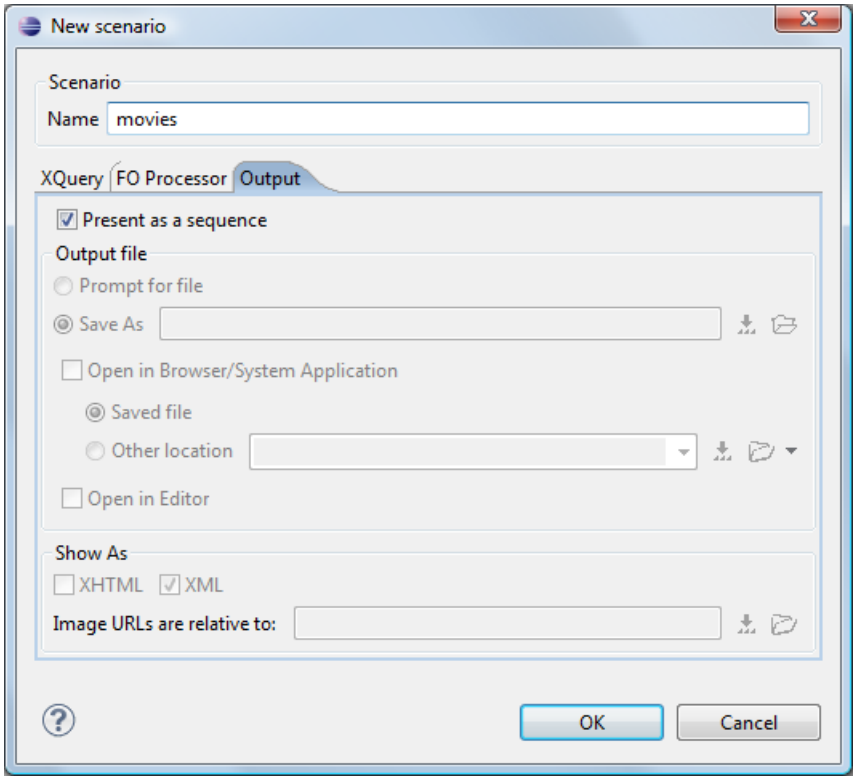


Figure 327: The XQuery transformation result displayed in Sequence view

A chunk of the XQuery transformation result is displayed in the **Sequence** view.

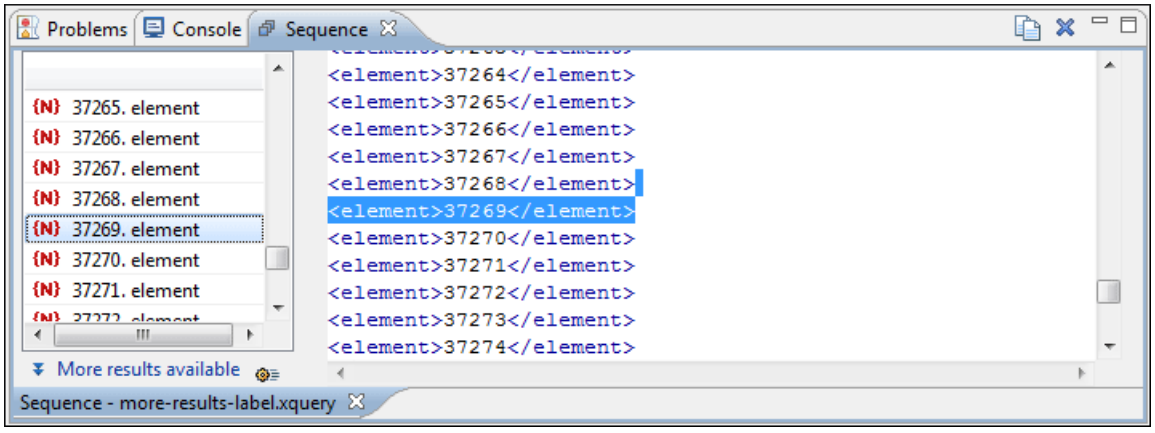


Figure 328: The XQuery transformation result displayed in Sequence view

Advanced Saxon HE/PE/EE XQuery Transformation Options

The XQuery transformation scenario allows you to configure advanced options that are specific for the Saxon HE (Home Edition), PE (Professional Edition), and EE (Enterprise Edition) engines. They are the same options as *the values set in the user preferences* but they are configured as a specific set of transformation options for each transformation scenario.

The advanced options for Saxon 9.6.0.7 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

- **Recoverable errors ("-warnings")**- Allows you to choose how dynamic errors are handled. The following options can be selected:
  - **recover silently ("silent")** - Continues processing without reporting the error.
  - **recover with warnings ("recover")** - Issues a warning but continues processing.

  - **signal the error and do not attempt recovery ("fatal")** - Issues an error and stops processing.

- **Strip whitespaces ("-strip")** - Can have one of the following values:
  - **All ("all")** - Strips *all* whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document.
  - **Ignore ("ignorable")** - Strips all *ignorable* whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
  - **None ("none")** - Strips *no* whitespace before further processing.
- **Optimization level ("-opt")** - This option allows optimization to be suppressed when reducing the compiling time is important, optimization conflicts with debugging, or optimization causes extension functions with side-effects to behave unpredictably.
- **Use linked tree model ("-tree:linked")** - This option activates the linked tree model.
- **Enable XQuery 3.0 support ("-qversion:(1.0|3.0)")** - If checked, Saxon runs the XQuery transformation with the XQuery 3.0 support (this option is enabled by default).
- **Initializer class** - Equivalent to the *-init* Saxon command-line argument. The value is the name of a user-supplied class that implements the `net.sf.saxon.lib.Initializer` interface. This initializer is called during the initialization process, and may be used to set any options required on the configuration programmatically. It is particularly useful for tasks such as registering extension functions, collations, or external object models, especially in Saxon-HE where the option cannot be set via a configuration file. Saxon only calls the initializer when running from the command line, but the same code may be invoked to perform initialization when running user application code.

  Important: The *advanced Saxon-HE/PE/EE options configured in a transformation scenario* override the Saxon-HE/PE/EE options defined globally.

The following advanced options are specific for Saxon 9.6.0.7 Professional Edition (PE) and Enterprise Edition (EE) only:

- **Use a configuration file ("-config")** - Sets a Saxon 9 configuration file that is used for XQuery transformation and validation
- **Allow calls on extension functions ("-ext")** - If checked, calls on external functions are allowed. Checking this option is recommended in an environment where untrusted stylesheets may be executed. It also disables user-defined extension elements and the writing of multiple output files, both of which carry similar security risks.

The advanced options that are specific for Saxon 9.6.0.7 Enterprise Edition (EE) are as follows:

- **Validation of the source file ("-val")** - Requests schema-based validation of the source file and of any files read using the `document()` or similar functions. It can have the following values:
  - **Schema validation ("strict")** - This mode requires an XML Schema and enables parsing the source documents with strict schema-validation enabled.
  - **Lax schema validation ("lax")** - If an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled but the validation will not fail if, for example, element declarations are not found.
  - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.
- **Validation errors in the results tree treated as warnings ("-outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.
- **Generate bytecode ("--generateByteCode:(on|off)")** - If you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such an action. For further details regarding this option, go to *http://www.saxonica.com/documentation9.5/index.html#!javadoc*.
- **Enable XQuery update ("-update:(on|off)")** - This option controls whether or not XQuery update syntax is accepted. The default value is off.
- **Backup files updated by XQuery ("-backup:(on|off)")** - If checked, backup versions for any XML files updated with XQuery Update are generated. This option is available when the **Enable XQuery update** option is enabled.

Updating XML Documents using XQuery

Using the bundled Saxon 9.6.0.7 EE XQuery processor Oxygen XML Editor plugin offers support for XQuery Update 1.0. The XQuery Update Facility provides expressions that can be used to make persistent changes to instances of the XQuery 1.0 and XPath 2.0 Data Model. Thus, besides querying XML documents, you can modify them using the various insert/delete/modify/create methods available in the *XQuery Update 1.0* standard.

Choose Saxon 9.6.0.7 EE as a transformer in the scenario associated with the XQuery files containing update statements and Oxygen XML Editor plugin will notify you if the update was successful.

Using XQuery Update to modify a tag name in an XML file

```
rename node doc("books.xml")//publisher[1]//book[1] as "firstBook"
```

Chapter 12

Debugging XSLT Stylesheets and XQuery Documents

Topics:

This chapter explains the user interface and how to use the debugger with XSLT and XQuery transformations.

XSLT/XQuery Debugging Overview

The **XSLT Debugger** and **XQuery Debugger** perspectives enable you to test and debug XSLT 1.0 / 2.0 / 3.0 stylesheets and XQuery 1.0 / 3.0 documents including complex XPath 2.0 / 3.0 expressions. The interface presents simultaneous views of the source XML document, the XSLT/XQuery document and the result document. As you go step by step through the XSLT/XQuery document the corresponding output is generated step by step, and the corresponding position in the XML file is highlighted. At the same time, special views provide various types of debugging information and events useful to understand the transformation process.

The following set of features allow you to test and solve XSLT/XQuery problems:

- Support for XSLT 1.0 stylesheets (using Saxon 6.5.5 and Xalan XSLT engines), XSLT 2.0 / 3.0 stylesheets and XPath 2.0 / 3.0 expressions that are included in the stylesheets (using Saxon 9.6.0.7 XSLT engine) and XQuery 1.0 / 3.0 (using Saxon 9.6.0.7 XQuery engine).
- Stepping capabilities: step in, step over, step out, run, run to cursor, run to end, pause, stop.
- Output to source mapping between every line of output and the instruction element / source context that generated it.
- Breakpoints on both source and XSLT/XQuery documents.
- Call stack on both source and XSLT/XQuery documents.
- Trace history on both source and XSLT/XQuery documents.
- Support for XPath expression evaluation during debugging.
- Step into imported/included stylesheets as well as included source entities.
- Available templates and hits count.
- Variables view.
- Dynamic output generation.

Layout

The XML and XSL files are displayed in *Text mode*. The other modes (*Author mode*, *Grid mode*) are available only in *the Editor perspective*.

The debugger perspective contains four sections:

- **Source document view (XML)** - Displays and allows the editing of XML files (documents).
- **XSLT/XQuery document view (XSLT/XQuery)** - Displays and allows the editing of XSL files(stylesheets) or XQuery documents.
- **Output document view** - Displays the output that results from inputting a document (XML) and a stylesheet (XSL) or XQuery document in the transformer. The transformation result is written dynamically while the transformation is processed.
- **Control view** - The control view is used to configure and control the debugging operations. It also provides a set of *Information views* types. This pane has two sections:
  - *Control toolbar*
  - *Information views*

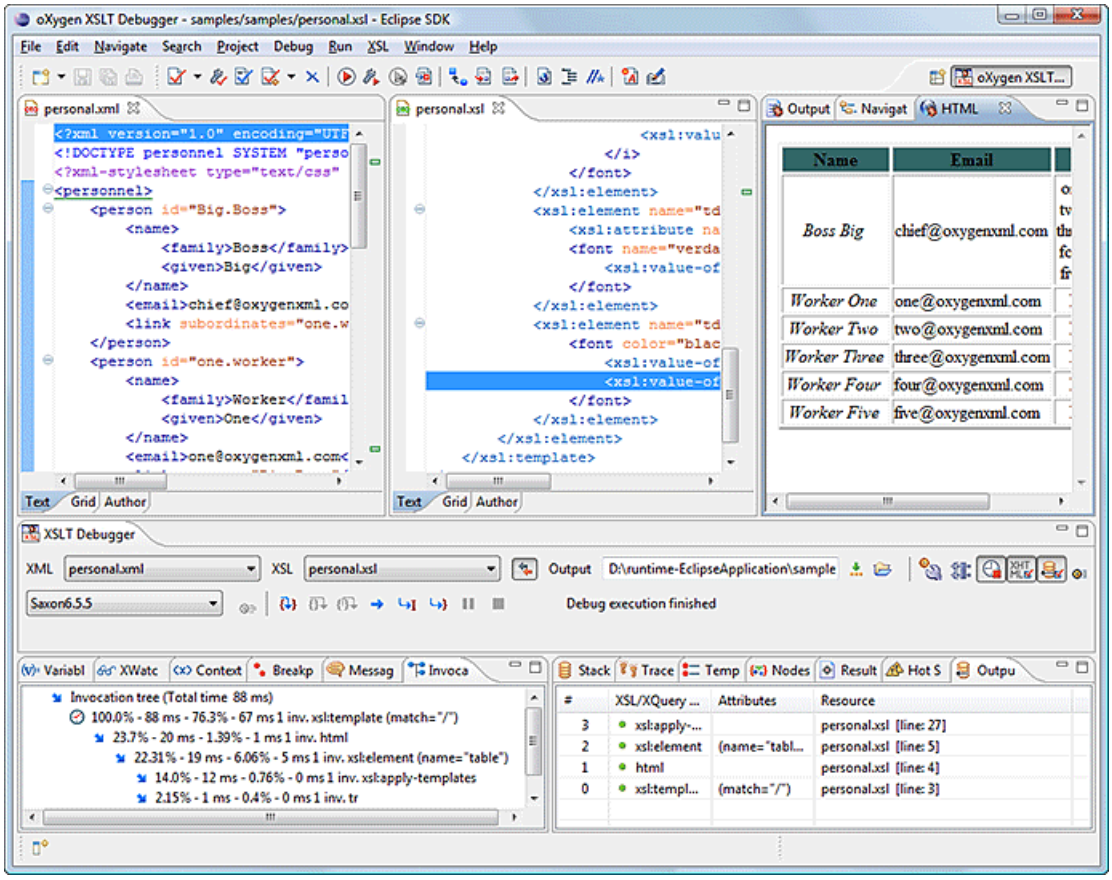


Figure 329: Debugger Mode Interface

XML documents and XSL stylesheets or XQuery documents that were opened in the Editor perspective are automatically sorted into the first two panes. When multiple files of each type are opened, the individual documents and stylesheets are separated using the familiar tab management system of the Editor perspective. Selecting a tab brings the document or stylesheet into focus and enables editing without the need to go back to the Editor perspective.

During debugging, the current execution node is highlighted in both document (XML) and XSLT/XQuery views.

Control Toolbar

The **Control** toolbar contains all the actions that you need to configure and control the debugging process. The following actions are described as they appear in the toolbar from left to right.

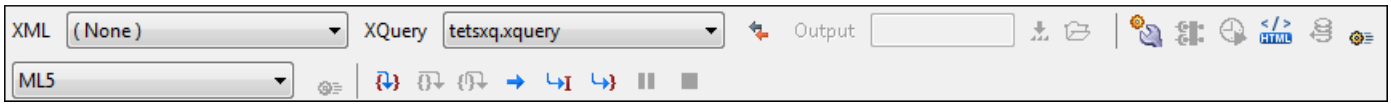


Figure 330: Control Toolbar

XML source selector

The current selection represents the source document used as input by the transformation engine. The selection list contains all opened files (XML files being emphasized). This option allows you to use other file types also as source documents. In an XQuery debugging session this selection field can be set to the default value `NONE`, because usually XQuery documents do not require an input source.

XSL / XQuery selector

The current selection represents the stylesheet or XQuery document to be used by the transformation engine. The selection list contains all opened files (XSLT / XQuery files being emphasized).

Link with editor

When enabled, the XML and XSLT/XQuery selectors display the names of the files opened in the central editor panels. This button is disabled by default.

Output selector

The selection represents the output file specified in the associated transformation scenario.

XSLT / XQuery parameters

XSLT / XQuery parameters to be used by the transformation.

Libraries

Add and remove the Java classes and jars used as XSLT extensions.

Turn on profiling

Enables / Disables current transformation profiling.

Enable XHTML output

Enables the rendering of the output in the XHTML output view during the transformation process. For performance issues, disable XHTML output when working with very large files. Note that only XHTML conformant documents can be rendered by this view. In order to view the output result of other formats, such as HTML, save the **Text output** area to a file and use an external browser for viewing.

When starting a debug session from the editor perspective using the **Debug Scenario** action, the state of this toolbar button reflects the state of the **Show as XHTML** output option from the scenario.

Turn on/off output to source mapping

Enables or disables the output to source mapping between every line of output and the instruction element / source context that generated it.

Debugger preferences

Quick link to *Debugger preferences page*.

XSLT / XQuery engine selector

Lists the *processors available for debugging XSLT and XQuery transformations*.

XSLT / XQuery engine advanced options

Advanced options available for Saxon 9.6.0.7.

Step into F7

Starts the debugging process and runs until the next instruction is encountered.

Step over F8 (Alt F8 on OS X)

Run until the current instruction and its sub-instructions are over. Usually this will advance to the next sibling instruction.

Step out Shift F7 (Meta F8 on OS X)

Run until the parent of the current instruction is over. Usually this will advance to the next sibling of the parent instruction.

Run

Starts the debugging process. The execution of the process is paused when a *breakpoint* is encountered or the transformation ends.

Run to cursor Ctrl F5

Starts the debugging process and runs until one of the following conditions occur: the line of cursor is reached, a valid breakpoint is reached or the execution ends.

Run to end Alt F5

Runs the transformation until the end, without taking into account enabled breakpoints, if any.

Pause Shift F6

Request to pause the current transformation as soon as possible.

Stop F6

Request to stop the current transformation without completing its execution.

Show current execution nodes

Reveals the current debugger context showing both the current instruction and the current node in the XML source. Possible displayed states:

- Entering () or leaving () an XML execution node.
- Entering () or leaving () an XSL execution node.
- Entering () or leaving () an XPath execution node.

Note: When you set a MarkLogic server as a processor, the **Show current execution nodes** button is named **Refresh current session context from server**. Click this button to refresh the information in all the views.

Note: For some of the XSLT processors (Saxon-HE/PE/EE) the debugger could be configured to step into the XPath expressions affecting the behavior of the following debugger actions: **Step into**, **Step over** or **Step Out**.

Debugging Information Views

The information views at the bottom of the editor is comprised of two panes that are used to display various types of information used to understand the transformation process. For each information type there is a corresponding tab. While running a transformation, relevant events are displayed in the various information views. This enables the developer to obtain a clear view of the transformation progress. Using the debug controls developers can easily isolate parts of stylesheet therefore they may be understood and modified. The information types include the following:

Left side information views

- ***Context** node view*
- ***XWatch** view*
- ***Breakpoints** view*
- ***Messages** view* (XSLT only)
- ***Variables** view*
- ***Invocation Tree** view*

Right side information views

- ***Stack** view*
- ***Output Mapping Stack** view*
- ***Trace** view*
- ***Templates** view* (XSLT only)
- ***Nodes/Values Set** view*
- ***Hotspots** view*

Context Node View

The context node is valid only for XSLT debugging sessions and is a source node corresponding to the XSL expression that is evaluated. It is also called the context of execution. The context node implicitly changes as the processor hits various steps (at the point where XPath expressions are evaluated). This node has the same value as evaluating '.' (dot) XPath expression in ***XWatch** view*. The value of the context node is presented as a tree in the view.

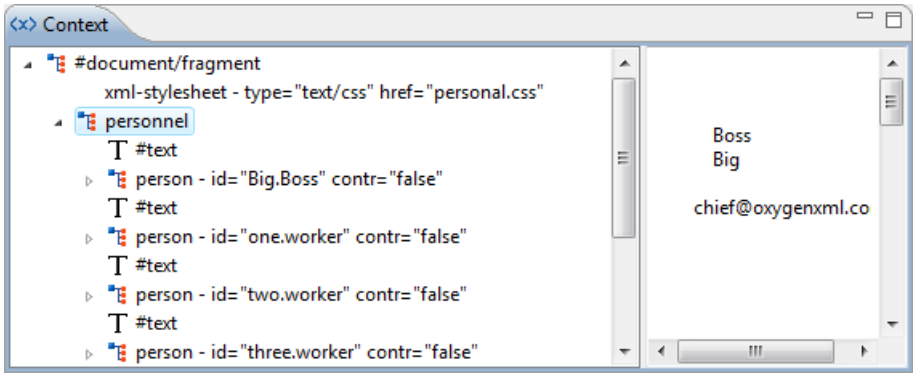


Figure 331: The Context node view

The context node is presented in a tree-like fashion. Nodes from a defined namespace bound to a prefix are displayed using the qualified name. If the namespace is not bound to a prefix, the namespace URI is presented before the node name. The value of the selected attribute or node is displayed in the right side panel. The **Context** view also presents the current mode of the XSLT processor if this mode differs from the default one.

XPath Watch (XWatch) View

The **XWatch** view shows XPath expressions evaluated during the debugging process. Expressions are evaluated dynamically as the processor changes its source context.

When you type an XPath expression in the **Expression** column, Oxygen XML Editor plugin supports you with syntax highlight and *content completion assistance*.

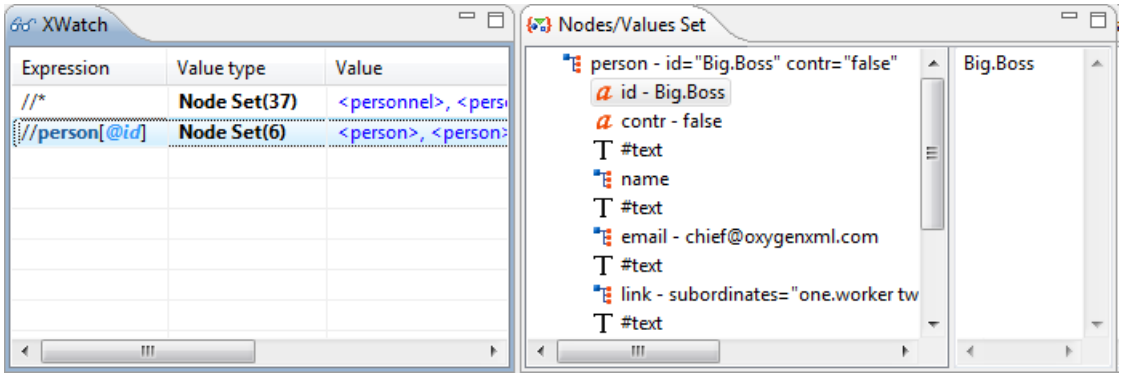


Figure 332: The XPath watch view

Table 9: XWatch columns

Column	Description
Expression	XPath expression to be evaluated (XPath 1.0 or 2.0 / 3.0 compliant).
Value	Result of XPath expression evaluation. Value has a type (see *the possible values* in the section *Variables View* on page 814). For *Node Set* results, the number of nodes in the set is shown in parenthesis.

Important: Remarks about working with the **XWatch** view:

- Expressions that reference variable names are not evaluated.
- The expression list is not deleted at the end of the transformation (it is preserved between debugging sessions).
- To insert a new expression, click the first empty line of the **Expression** column and start typing.

- To delete an expression, click its **Expression** column and delete its content.
- If the expression result type is a *Node Set*, click it (**Value** column) and its value is displayed in the ***Nodes/Values Set** view*.
-

Breakpoints View

This view lists all breakpoints that are set on opened documents. Once you *insert a breakpoint* it is automatically added in this list. Breakpoints can be set in XSLT/XQuery documents and XML documents in XSLT/XQuery debugging sessions. A breakpoint can have an associated break condition that represents an XPath expression evaluated in the current debugger context. In order to be processed, their evaluation result should be a boolean value. A breakpoint with an associated condition only stops the execution of the Debugger if the breakpoint condition is evaluated as **true**.

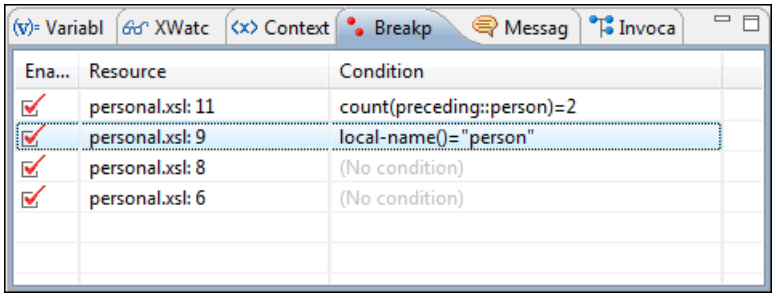


Figure 333: The Breakpoints View

The **Breakpoints** view contains the following columns:

- **Enabled** - If checked, the current condition is evaluated and taken into account.
- **Resource** - Resource file and number of the line where the breakpoint is set.
- **Condition** - XSLT/XQuery expression to be evaluated during debugging. The expression will be evaluated at every debug step.

Clicking a record highlights the breakpoint line in the document.

Note: The breakpoints list is not deleted at the end of a transformation (it is preserved between debugging sessions).

The following actions are available in the contextual menu of the table:

Go to

Moves the cursor to the source of the breakpoint.

Enable

Enables the breakpoint.

Disable

Disables the breakpoint. A disabled breakpoint will not be evaluated by the Debugger.

Add

Allows you to add a new breakpoint and breakpoint condition.

Edit

Allows you to edit an existing breakpoint.

Remove

Deletes the selected breakpoint.

Enable all

Enables all breakpoints.

Disable all

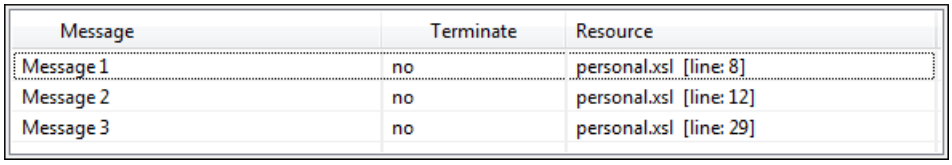
Disables all breakpoints.

Remove all

Removes all breakpoints.

Messages View

`xsl:message` instructions are one way to signal special situations encountered during transformation as well as a raw way of doing the debugging. This view is available only for XSLT debugging sessions and shows all `xsl:message` calls executed by the XSLT processor during transformation.



Message	Terminate	Resource
Message 1	no	personal.xsl [line: 8]
Message 2	no	personal.xsl [line: 12]
Message 3	no	personal.xsl [line: 29]

Figure 334: The Messages View

Table 10: Messages columns

Column	Description
Message	Message content.
Terminate	Signals if processor terminates the transformation or not once it encounters the message (yes/no respectively).
Resource	Resource file where `xsl:message` instruction is defined and the message line number.

The following actions are available in the contextual menu:

Go to

Highlight the XSL fragment that generated the message.

Copy

Copies to clipboard message details (system ID, severity info, description, start location, terminate state).

Important: Remarks

- Clicking a record from the table highlights the `xsl:message` declaration line.
- Message table values can be sorted by clicking the corresponding column header. Clicking the column header switches the sorting order between: ascending, descending, no sort.

Stack View

This view shows the current execution stack of both source and XSLT/XQuery nodes. During transformation two stacks are managed: one of source nodes being processed and the other for XSLT/XQuery nodes being processed. Oxygen XML Editor plugin shows both node types into one common stack. The source (XML) nodes are preceded by a red color icon while XSLT/XQuery nodes are preceded by a green color icon. The advantage of this approach is that you can always see the source scope on which a XSLT/XQuery instruction is executed (the last red color node on the stack). The stack is oriented upside down.

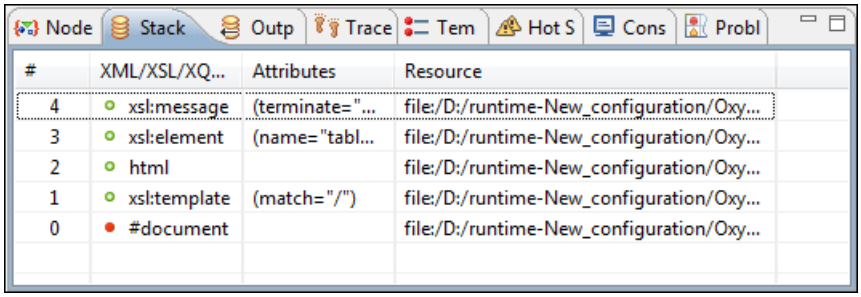


Figure 335: The Stack View

The contextual menu contains one action: **Go to**, which moves the selection in the editor panel to the line containing the XSLT element that is displayed on the selected line from the view.

Table 11: Stack columns

Column	Description
#	Order number, represents the depth of the node (0 is the stack base).
XML/XSLT/XQuery Node	Node from source or stylesheet document currently being processed. One particular stack node is the document root, noted as **#document**.
Attributes	Attributes of the node (a list of `id="value"` pairs).
Resource	Resource file where the node is located.

Important: Remarks:

- Clicking a record from the stack highlights that node's location inside resource.
- Using Saxon, the stylesheet elements are qualified with XSL proxy, while using Xalan you only see their names. (example: `xsl:template` using Saxon and `template` using Xalan).
- Only the Saxon processor shows element attributes.
- The Xalan processor shows also the built-in rules.

Output Mapping Stack View

The **Output Mapping Stack** view displays *context data* and presents the XSLT templates/XQuery elements that generated specific areas of the output.

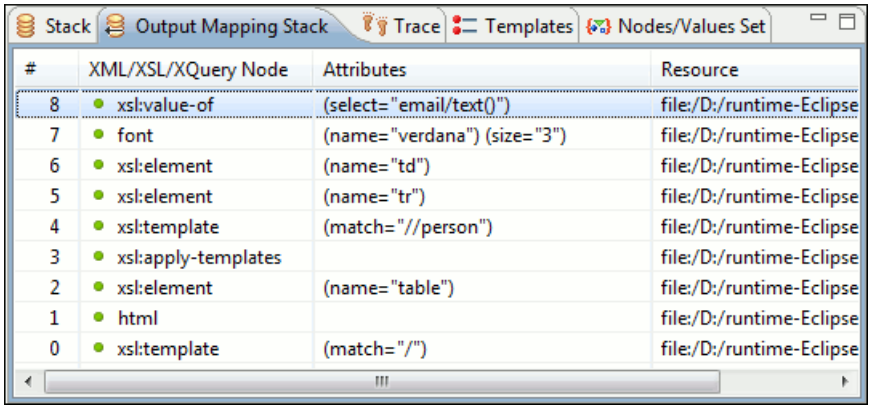


Figure 336: The Output Mapping Stack view

The **Go to** action of the contextual menu takes you in the editor panel at the line containing the XSLT element displayed in the **Output Mapping Stack** view.

Table 12: Output Mapping Stack columns

Column	Description
#	The order number in the stack of XSLT templates/XQuery elements. Number 0 corresponds to the bottom of the stack in the status of the XSLT/XQuery processor. The highest number corresponds to the top of the stack.
XSL/XQuery Node	The name of an XSLT template/XQuery element that participated in the generation of the selected output area.
Attributes	The attributes of the XSLT template/XQuery node.
Resource	The name of the file containing the XSLT template/XQuery element.

Important: Remarks:

- Clicking a record highlights that XSLT template definition/XQuery element inside the resource (XSLT stylesheet file/XQuery file).
- Saxon only shows the applied XSLT templates having at least one hit from the processor. Xalan shows all defined XSLT templates, with or without hits.
- The table can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in XSLT rules.

Trace History View

Usually the XSLT/XQuery processors signal the following events during transformation:

- - Entering a source (XML) node.
- - Leaving a source (XML) node.
- - Entering a XSLT/XQuery node.
- - Leaving a XSLT/XQuery node.

The trace history catches all these events, so you can see how the process evolved. The red icon lines denote source nodes while the green icon lines denote XSLT/XQuery nodes.

It is possible to save the element trace in a structured XML document. The action is available on the contextual menu of the view. In this way you have the possibility to compare the trace results from different debug sessions.

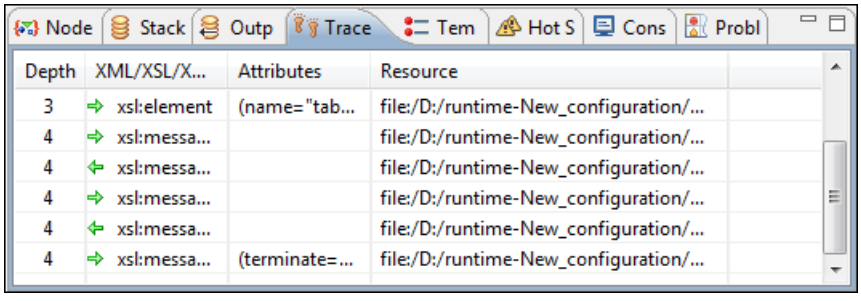


Figure 337: The Trace History View

The contextual menu contains the following actions:

Go to

Moves the selection in the editor panel to the line containing the XSLT element or XML element that is displayed on the selected line from the view;

Export to XML

Saves the entire trace list into XML format.

Table 13: Trace History columns

Column	Description
Depth	Shows you how deep the node is nested in the XML or stylesheet structure. The bigger the number, the more nested the node is. A depth 0 node is the document root.
XML/XSLT/XQuery Node	Represents the node from the processed source or stylesheet document. One particular node is the document root, noted as `#document`. Every node is preceded by an arrow that represents what action was performed on it (entering or leaving the node).
Attributes	Attributes of the node (a list of `id="value"` pairs).
Resource	Resource file where the node is located.

Important: Remarks:

- Clicking a record highlights that node's location inside the resource.
- Only the Saxon processor shows the element attributes.
- The Xalan processor shows also the built-in rules.

Templates View

The `xsl:template` is the basic element for stylesheets transformation. This view is only available during XSLT debugging sessions and shows all `xsl:template` instructions used by the transformation. Being able to see the number of *hits* for each of the templates allows you to get an idea of the stylesheet coverage by template rules with respect to the input source.

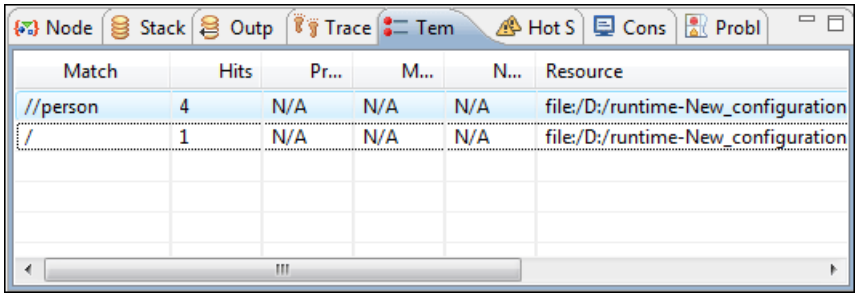


Figure 338: The Templates view

The contextual menu contains one action: **Go to**, which moves the selection in the editor panel to the line containing the XSLT template that is displayed on the selected line from the view.

Table 14: Templates columns

Column	Description
Match	The `match` attribute of the `xsl:template`.

Column	Description
Hits	The number of hits for the `xsl:template`. Shows how many times the XSLT processor used this particular template.
Priority	The template priority as established by XSLT processor.
Mode	The `mode` attribute of the `xsl:template`.
Name	The `name` attribute of the `xsl:template`.
Resource	The resource file where the template is located.

Important: Remarks:

- Clicking a record highlights that template definition inside the resource.
- Saxon only shows the applied templates having at least one hit from the processor. Xalan shows all defined templates, with or without hits.
- Template table values can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in rules.

Nodes/Values Set View

This view is always used in relation with *The **Variables** view* and *the **XWatch** view*l. It shows an XSLT node set value in a tree form. The node set view is updated as response to the following events:

- You click a variable having a node set value in one of the above 2 views.
- You click a tree fragment in one of the above 2 views.
- You click an XPath expression evaluated to a node set in one of the above 2 views.

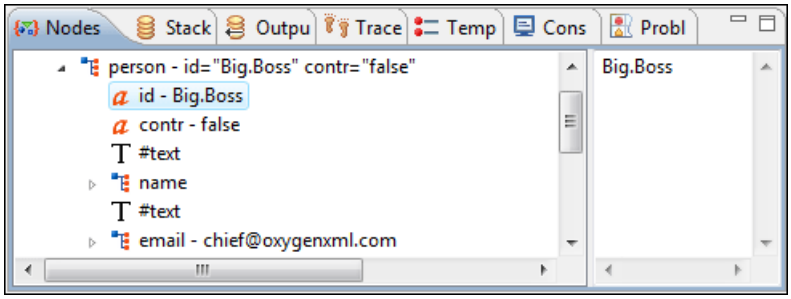


Figure 339: The Node Set view

The nodes / values set is presented in a tree-like fashion. Nodes from a defined namespace bound to a prefix are displayed using the qualified name. If the namespace is not bound to a prefix the namespace URI is presented before the node name. The value of the selected attribute or node is displayed in the right side panel.

Important: Remarks:

- For longer values in the right side panel, the interface displays it with an ellipsis (...) at the end. A more detailed value is available as a tooltip when hovering over it.
- Clicking a record highlights the location of that node in the source or stylesheet view.

Variables View

Variables and parameters play an important role during an XSLT/XQuery transformation. Oxygen XML Editor plugin uses the following icons to differentiate variables and parameters:

- V - Global variable.
- {V} - Local variable.

- P - Global parameter.
- {P} - Local parameter.

The following value types are available:

- **Boolean**
- **String**
- **Date** - XSLT 2.0 / 3.0 only.
- **Number**
- **Set**
- **Object**
- **Fragment** - Tree fragment.
- **Any**
- **Undefined** - The value was not yet set, or it is not accessible.

  Note:

  When Saxon 6.5 is used, if the value is unavailable, then the following message is displayed in the **Value** field: "The variable value is unavailable".

  When Saxon 9 is used:

  - If the variable is not used, the **Value** field displays "The variable is declared but never used".
  - If the variable value cannot be evaluated, the **Value** field displays "The variable value is unavailable".

- **Document**
- **Element**
- **Attribute**
- **ProcessingInstruction**
- **Comment**
- **Text**
- **Namespace**
- **Evaluating** - Value under evaluation.
- **Not Known** - Unknown types.

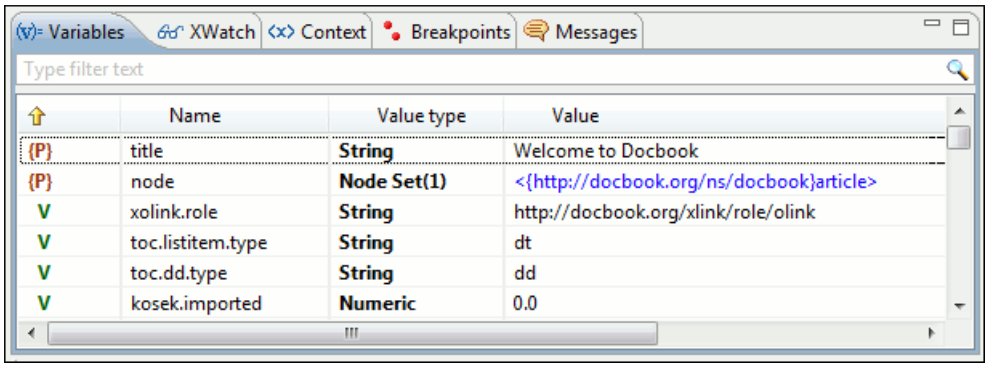


Figure 340: The Variables View

Table 15: Variables columns

Column	Description
Name	Name of variable / parameter.
Value type	Type of variable/parameter.
Value	Current value of variable / parameter.

The value of a variable (the **Value** column) can be copied to the clipboard for pasting it to other editor area with the action **Copy value** from the contextual menu of the table from the view. This is useful in case of long and complex values which are not easy to remember by looking at them once.

Important: Remarks:

- Local variables and parameters are the first entries presented in the table.
- Clicking a record highlights the variable definition line.
- Variable values could differ depending on the transformation engine used or stylesheet version set.
- If the value of the variable is a node set or a tree fragment, clicking it causes the ***Node Set** view* to be shown with the corresponding set of values.
- Variable table values can be sorted by clicking the corresponding column header. Clicking the column header switches between the orders: ascending, descending, no sort.

Multiple Output Documents in XSLT 2.0 and XSLT 3.0

For XSLT 2.0 and XSLT 3.0 stylesheets that store the output in more than one file by using the `xsl:result-document` instruction the content of the file created in this way is displayed dynamically while the transformation is running in an output view. There is one view for each `xsl:result-document` instruction so that the output of different instructions is not mixed but is presented in different views.

Working with the XSLT / XQuery Debugger

This section describes how to work with the debugger in the most common use cases.

To watch our video demonstration about how you can use the **XSLT Debugger**, go to *http://oxygenxml.com/demo/XSLT_Debugger.html*.

Steps in a Typical Debug Process

To debug a stylesheet or XQuery document follow the procedure:

1. Open the source XML document and the XSLT/XQuery document.
2. If you are in the Oxygen XML Editor plugin XML perspective switch to the Oxygen XML Editor plugin XSLT Debugger perspective or the Oxygen XML Editor plugin XQuery Debugger perspective with one of the actions (here explained for XSLT):
   - Menu **Window** > **Open Perspective** > **Other** > **Oxygen XSLT Debugger**
   - The toolbar button **Debug scenario** - This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer engine, the XSLT parameters, the transformer extensions, etc.) will be saved back in the scenario when exiting from the Debugger perspective.
3. Select the source XML document in the XML source selector of *the Control toolbar*. In the case of XQuery debugging, if your XQuery document has no implicit source, set the source selector value to **NONE**.
4. Select the XSLT/XQuery document in the XSLT/XQuery selector of *the Control toolbar*.
5. Set XSLT/XQuery parameters from the button available on *the Control toolbar*.
6. *Set one or more breakpoints*.
7. Step through the stylesheet using the buttons available on *the Control toolbar*:
   - **Step into**
   - **Step over**
   - **Step out**
   - **Run**
   - **Run to cursor**

- **Run to end**
- **Pause**
- **Stop**

8. Examine the information in the Information views to find the bug in the transformation process.

   You may find *the procedure for determining the XSLT template/XQuery element that generated an output section* useful for fixing bugs in the transformation.

Using Breakpoints

The Oxygen XML Editor plugin XSLT/XQuery Debugger allows you to interrupt XSLT/XQuery processing to gather information about variables and processor execution at particular points. To ensure breakpoints are persistent between work sessions, they are saved at project level. You can set a maximum of 100 breakpoints per project.

Inserting Breakpoints

To insert a breakpoint, follow these steps:

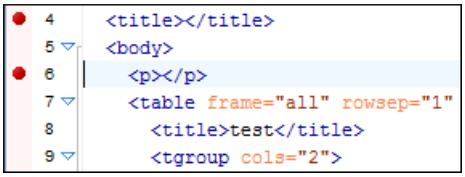
1. Click the line where you want to insert the breakpoint in the XML source document or the XSLT/XQuery document.

   You can only set breakpoints on the XML source in XSLT or XQuery debugging sessions.

   Breakpoints are automatically created on the ending line of a start tag, even if you click a different line.

2. Right-click the vertical stripe on the left side of the editor panel and select **Add breakpoint**.

Breakpoints



Removing Breakpoints

Only one action is required to remove a breakpoint:

Right-click the breakpoint icon in the vertical stripe on the left side of the editor panel and select **Remove breakpoint**.

Determining What XSLT / XQuery Expression Generated Particular Output

In order to quickly spot the XSLT templates or XQuery expressions with problems it is important to know what XSLT template in the XSLT stylesheet or XQuery expression in the XQuery document and what element in the source XML document generated a specified area in the output.

Some of the debugging capabilities, for example *Step in* can be used for this purpose. Using *Step in* you can see how output is generated and link it with the XSLT/XQuery element being executed in the current source context. However, this can become difficult on complex XSLT stylesheets or XQuery documents that generate a large output.

You can click the text from the **Text** output view or **XHTML** output view and the editor will select the XML source context and the XSLT template/XQuery element that generated the text. Also inspecting the whole stack of XSLT templates/XQuery elements that determined the state of the XSLT/XQuery processor at the moment of generating the specified output area speeds up the debugging process.

1. Switch to the Oxygen XML Editor plugin XSLT Debugger perspective or Oxygen XML Editor plugin XQuery Debugger perspective with one of the actions (here explained for XSLT):

   - Go to **Window** > **Open Perspective** > **Other** > **Oxygen XSLT Debugger**
   - Go to . The toolbar button **Debug scenario** . This action initializes the Debugger perspective with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer

engine, XSLT parameters, transformer extensions, etc.) will be saved back in the scenario when exiting from the Debugger perspective.

2. Select the source XML document in the XML source selector of *the Control toolbar*. In the case of XQuery debugging without an implicit source choose the NONE value.
3. Select the XSLT / XQuery document in the XSLT / XQuery selector of *the Control toolbar*.
4. Select the XSLT / XQuery engine in the XSLT / XQuery engine selector of *the Control toolbar*.
5. Set XSLT / XQuery parameters from the button available on *the Control toolbar*.
6. Apply the XSLT stylesheet or XQuery transformation using the **Run to end** button that is available on *the Control toolbar*.
7. Inspect the mapping by clicking a section of the output from the **Output** view of the Oxygen XML Editor plugin XSLT Debugger or Oxygen XML Editor plugin XQuery Debugger perspectives.

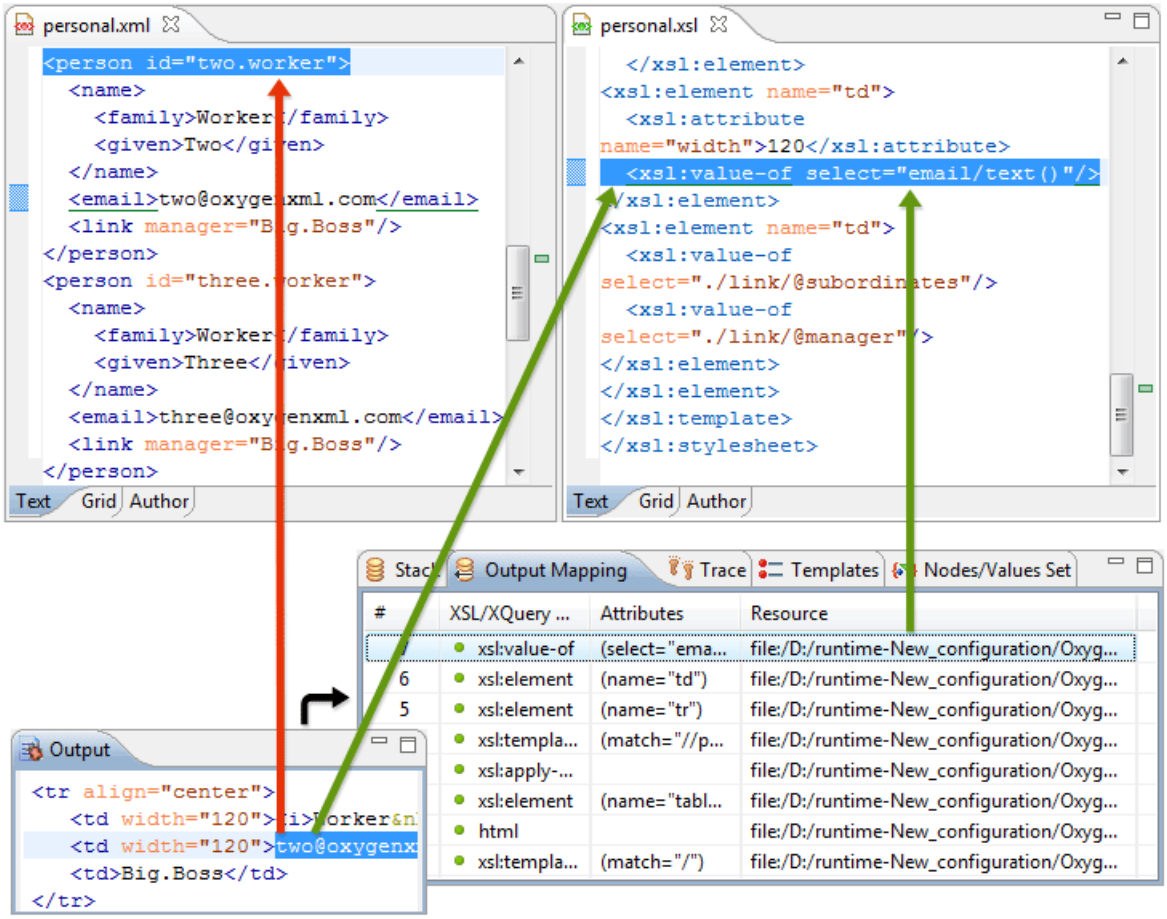


Figure 341: Text Output to Source Mapping

This action will highlight the XSLT / XQuery element and the XML source context. This XSLT template/XQuery element that is highlighted in the XSLT/XQuery editor represents only the top of the stack of XSLT templates/XQuery elements that determined the state of the XSLT/XQuery processor at the moment of generating the clicked output section. In the case of complex transformations inspecting the whole stack of XSLT templates/XQuery elements speeds up the debugging process. This stack is available in *the **Output Mapping Stack** view*.

Debugging Java Extensions

The XSLT/XQuery debugger does not step into Java classes that are configured as XSLT/XQuery extensions of the transformation. To step into Java classes, inspect variable values and set breakpoints in Java methods, set up a Java debug configuration in an IDE like the Eclipse SDK as described in the following steps:

1. Create a debug configuration.
   a) Set at least 256 MB as heap memory for the Java virtual machine (recommended 512 MB) by setting the `-Xmx` parameter in the debug configuration, for example "-Xmx512m".
   b) Make sure the `[OXYGEN_DIR]/lib/oxygen.jar` file and your Java extension classes are on the Java classpath.

      The Java extension classes should be the same classes that were *set as an extension* of the XSLT/XQuery transformation in the debugging perspective.
   c) Set the class `ro.sync.exml.Oxygen` as the main Java class of the configuration.

      The main Java class `ro.sync.exml.Oxygen` is located in the `oxygen.jar` file.
2. Start the debug configuration.

   Now you can set breakpoints and inspect Java variables as in any Java debugging process executed in the selected IDE (Eclipse SDK, and so on.).

Supported Processors for XSLT / XQuery Debugging

The following built-in XSLT processors are integrated in the debugger and can be selected in the *Control Toolbar*:

- Saxon 9.6.0.7 HE (Home Edition) - a limited version of the Saxon 9 processor, capable of running XSLT 1.0, XSLT 2.0 / 3.0 basic and XQuery 1.0 transformations, available in both the XSLT debugger and the XQuery one,
- Saxon 9.6.0.7 PE (Professional Edition) - capable of running XSLT 1.0 transformations, XSLT 2.0 basic ones and XQuery 1.0 ones, available in both the XSLT debugger and the XQuery one,
- Saxon 9.6.0.7 EE (Enterprise Edition) - a schema aware processor, capable of running XSLT 1.0 transformations, XSLT 2.0 /3.0 basic ones, XSLT 2.0 / 3.0 schema aware ones and XQuery 1.0 / 3.0 ones, available in both the XSLT debugger and the XQuery debugger,
- Saxon 6.5.5 - capable of running only XSLT 1.0 transformations, available only in the XSLT debugger,
- Xalan 2.7.1 - capable of running only XSLT 1.0 transformations, available only in the XSLT debugger.

Chapter 13

Performance Profiling of XSLT Stylesheets and XQuery Documents

Topics:

This chapter explains the user interface and how to use the profiler for finding performance problems in XSLT transformations and XQuery ones.

XSLT/XQuery Performance Profiling Overview

Whether you are trying to identify a performance issue that is causing your production XSLT/XQuery transformation to not meet customer expectations or you are trying to proactively identify issues prior to deploying your XSLT/XQuery transformation, using the XSLT/XQuery profiler feature is essential to helping you save time and ultimately ensure a better performing, more scalable XSLT/XQuery transformation.

The XSLT/XQuery profiling feature can use any available XSLT/XQuery processors that could be used for debugging and it is available from the debugging perspective.

Enabling and disabling the profiler is controlled by the *Profiler button* from the *debugger control toolbar*. The XSLT/XQuery profiler is off by default. This option is not available during a debugger session so you should set it before starting the transformation.

Viewing Profiling Information

This section explains the views that display the profiling data collected by the profiles during the transformation.

Invocation Tree View

The **Invocation Tree** view shows a top-down call tree that represents how XSLT instructions or XQuery expressions are processed.

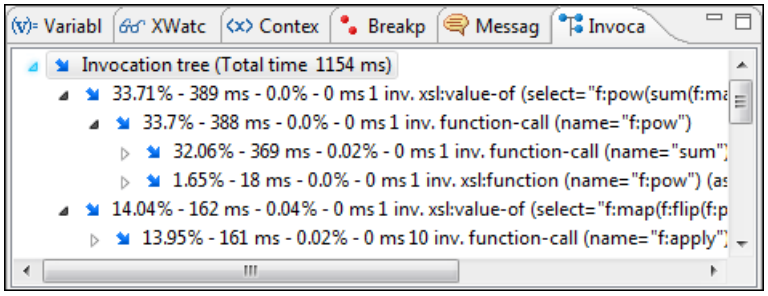


Figure 342: Invocation Tree View

The entries in the invocation tree include a few possible icons that indicate the following:

- - Points to a call whose inherent time is insignificant compared to its total time.
- - Points to a call whose inherent time is significant compared to its total time (greater than 1/3rd of its total time).

Every entry in the invocation tree includes textual information that depends on the ***XSLT/XQuery profiler settings***:

- A percentage number of the total time that is calculated with respect to either the root of the tree or the calling instruction.
- A total time measurement in milliseconds or microseconds. This is the total execution time that includes calls into other instructions.
- A percentage number of the inherent time that is calculated with respect to either the root of the tree or the calling instruction.
- An inherent time measurement in milliseconds or microseconds. This is the inherent execution time of the instruction.
- An invocation count that shows how often the instruction has been invoked on this call-path.
- An instruction name that contains also the attributes description.

Hotspots View

The **Hotspots** view displays a list of all instruction calls that lie above the threshold defined in the ***XSLT/XQuery profiler settings***.

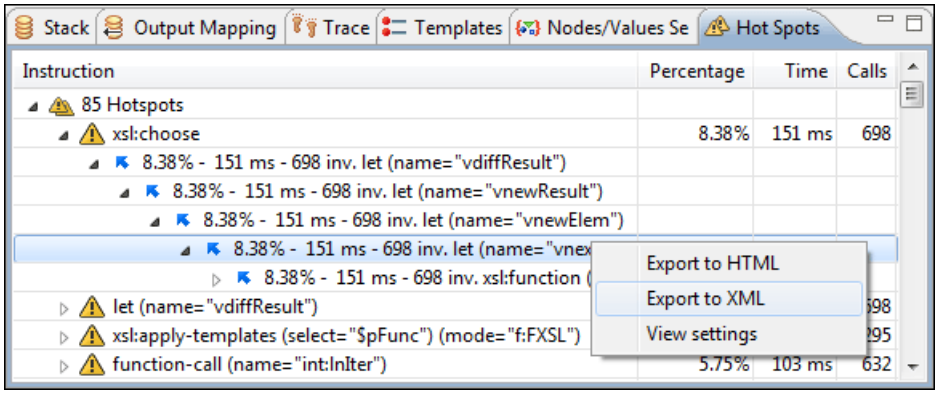


Figure 343: Hotspots View

By opening a hotspot instruction entry, the tree of back-traces leading to that instruction call are calculated and shown.

Every hotspot is described by the values from the following columns:

- **Instruction** - The name of the instruction.
- **Percentage** - The percentage number for this hotspot entry with respect to the total time.
- **Time** - The inherent time in milliseconds or microseconds of how much time has been spent in the hotspot. All calls into this instruction are summed up regardless of the particular call sequence.
- **Calls** - The invocation count of the hotspot entry.

If you click the ⚠ handle on the left side of a hotspot, a tree of back-traces will be shown.

Every entry in the backtrace tree has textual information attached to it that depends on the ***XSLT/XQuery profiler settings***:

- A percentage number that is calculated with respect to either the total time or the called instruction.
- A time measured in milliseconds or microseconds of how much time has been contributed to the parent hotspot on this call-path.
- An invocation count that shows how often the hotspot has been invoked on this call-path.

  Note: This is not the number of invocations of this instruction.

- An instruction name that also contains its attributes.

Working with XSLT/XQuery Profiler

Profiling activity is linked with debugging activity, so the first step in order to profile is to switch to debugging perspective and follow the corresponding procedure for debugging (see *Working with XSLT Debugger*).

Immediately after turning the profiler on two new information views are added to the current debugger *information views*:

- ***Invocation tree** view* on left side
- ***Hotspots** view* on right side

Profiling data is available only after the transformation ends successfully.

Looking to the right side (***Hotspots** view*), you can immediately spot the time the processor spent in each instruction. As an instruction usually calls other instructions the used time of the called instruction is extracted from the duration time of the caller (the hotspot only presents the inherent time of the instruction).

Looking to the left side (***Invocation tree** view*), you can examine how style instructions are processed. This result view is also named call-tree, as it represents the order of style processing. The profiling result shows the duration time for each of the style-instruction including the time needed for its called children.

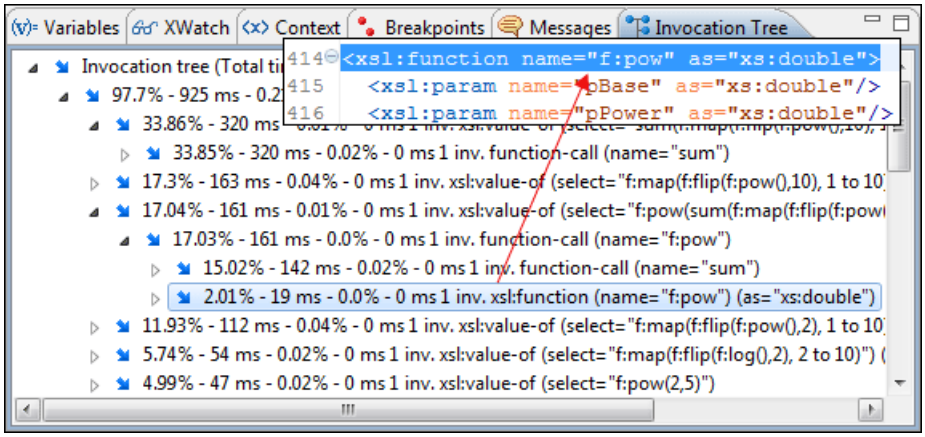


Figure 344: Source backmapping

In any of the above views you can use the backmapping feature in order to find the XSLT stylesheet or XQuery expression definition. Clicking the selected item cause Oxygen XML Editor plugin to highlight the XSLT stylesheet or XQuery expression source line where the instruction is defined.

When navigating through the trees by opening instruction calls, Oxygen XML Editor plugin automatically expands instructions which are only called by one other instruction themselves.

The profiling data can be saved into XML and HTML format. On any of the above views, use the contextual menu and select the corresponding choice. Basically saving HTML means saving XML and applying an XSLT stylesheet to render the report as XML. These stylesheets are included in the Oxygen XML Editor plugin distribution (see the subfolder `[OXYGEN_DIR]/frameworks/profiler/`) so you can make your own report based on the profiling raw data.

If you want to change the *XSLT/XQuery profiler settings*, use the contextual menu and choose the corresponding **View settings** entry.

Caution: Profiling exhaustive transformation may run into an OutOfMemory error due to the large amount of information being collected. If this is the case you can close unused projects when running the profiling or use high values for Java VM options `-Xms` and `-Xmx`. If this does not help you can shorten your source XML file and try again.

To watch our video demonstration about the XSLT/XQuery Profiler, go to *http://oxygenxml.com/demo/XSLT_Profiling.html*.

Chapter 14

Working with Archives

Topics:

Oxygen XML Editor plugin offers the means to manipulate files directly from ZIP type archives. By manipulation one should understand opening and saving files directly in archives, browsing and modifying archive structures. The archive support is available for all ZIP-type archives, which includes:

- ZIP archives
- EPUB books
- JAR archives
- Office Open XML (OOXML) files
- Open Document Format (ODF) files
- IDML files

This means that you can modify, transform, validate files directly from OOXML or ODF packages. The structure and content of an EPUB book, OOXML file or ODF file *can be opened, edited and saved* as for any other ZIP archive.

You can transform, validate and perform many other operations on files directly from an archive. When selecting an URL for a specific operation like transformation or validation you can click the  **Browse for archived file** button to navigate and choose the file from a certain archive.

Browsing and Modifying Archives

You can navigate archives either by opening them from the **Navigator** or by using the integration with the Eclipse File System. For the EFS (Eclipse File System) integration you must right click the archive in the **Navigator** and choose **Expand Zip Archive**. All the standard Eclipse **Navigator** actions are available on the mounted archive. If you decide to close the archive you can use the **Collapse ZIP Archive** action located in the contextual menu for the expanded archive. Any file opened from the archive expanded in the EFS will be closed when the archive in unmounted.

If you open an archive as an Eclipse editor, the archive will be unmounted when the editor is closed.

Important: If a file is not recognized by Oxygen XML Editor plugin as a supported archive type, you can add it from the *Archive preferences page*.

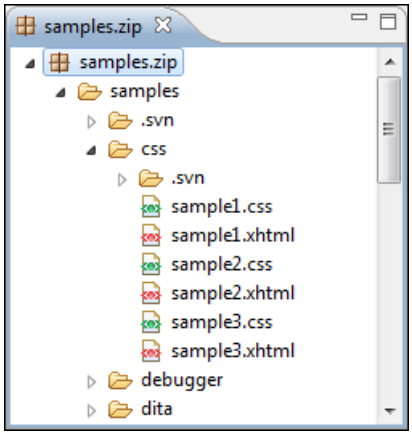


Figure 345: Browsing an Archive

Working with EPUB

EPUB is a free and open electronic book standard by the International Digital Publishing Forum (IDPF). It was designed for *reflowable content*, meaning that the text display can be optimized for the particular display device used by the reader of the EPUB-formatted book. Oxygen XML Editor plugin supports both EPUB 2.0 and EPUB 3.0.

Opening an EPUB file exposes all its internal components:

- Document content (XHTML and image files).
- Packaging files.
- Container files.

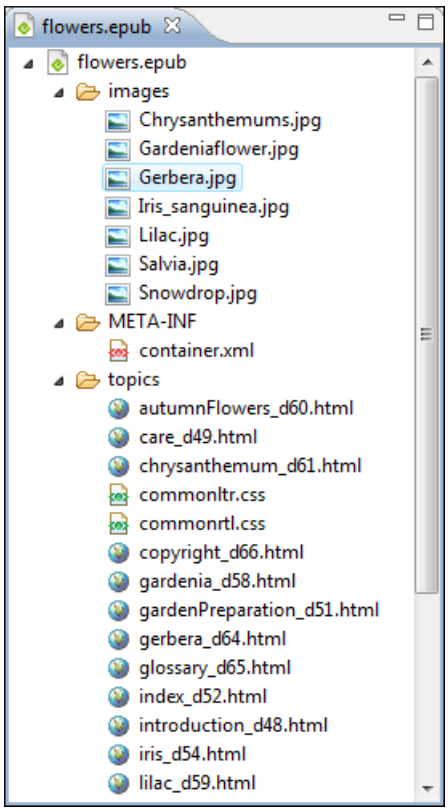


Figure 346: EPUB file displayed in Eclipse

Here you can edit, delete and add files that compose the EPUB structure. To check that the EPUB file you are editing is valid, invoke the **Validate and Check for Completeness** action. Oxygen XML Editor plugin uses the open-source EpubCheck validator to perform the validation. This validator detects many types of errors, including OCF container structure, OPF and OPS mark-up, as well as internal reference consistency. All errors found during validation are displayed in a separate tab in the **Errors** view.

Note: Invoke the **Open in System Application** action to see how the EPUB is rendered in your system default EPUB reader application.

Note: All changes made to the structure of an EPUB, or to the contents of the files inside an EPUB are immediately saved.

To watch our video demonstration about the EPUB support in Oxygen XML Editor plugin, go to *http://oxygenxml.com/demo/Epub.html*.

Create an EPUB

To begin writing an EPUB file from scratch, do the following:

1. Go to **File** > **New** , press **Ctrl N (Meta N on OS X)** on your keyboard. or click **New** on the main toolbar.
2. Choose **EPUB Book** template. Click **Create**. Choose the name and location of the file. Click **Save**.
   A skeleton EPUB file is saved on disk and open in the **Archive Browser** view.
3. Use the **Archive Browser** view specific actions to edit, add and remove resources from the archive.
4. Use the **Validate and Check for Completeness** action to verify the integrity of the EPUB archive.

Publish to EPUB

Oxygen XML Editor plugin comes with built-in support for publishing DocBook and DITA XML documents directly to EPUB.

1. Open the **Configure Transformation Scenario(s)** dialog box and select a predefined transformation scenario. To publish from DITA, select the **DITA Map EPUB** transformation scenario. To publish from DocBook select the **DocBook EPUB** transformation scenario.
2. Click **Apply associated** to run the transformation scenario.

Editing Files From Archives

You can open and edit files directly from an archive using the **Archive Browser** view. When saving the file back to archive, you are prompted to choose if you want the application to make a backup copy of the archive before saving the new content. If you choose **Never ask me again**, you will not be asked again to make backup copies. You can re-enable the pop-up message from the *Archive preferences page*.

Note: All changes made to the structure of an archive, or to the contents of the files inside an archive are immediately saved.

Chapter 15

Working with Databases

Topics:

XML is a storage and interchange format for structured data and is supported by all major database systems. Oxygen XML Editor plugin offers the means for managing the interaction with some of the most commonly used databases (both relational and Native XML Databases). Through this interaction, Oxygen XML Editor plugin helps users to understand browsing, querying, SQL execution support, content editing, importing from databases, and generating XML Schema from database structure.

Relational Database Support

Relational databases use a relational model and are based on tables linked by a common key. Oxygen XML Editor plugin offers support for the following relational databases: IBM DB2, MySQL, Microsoft SQL Server, and Oracle 11g.

The following actions are allowed:

- Browsing the tables of these types of databases in the **Data Source Explorer** view
- Executing SQL queries against them
- Calling stored procedures with input and output parameters

Oxygen XML Editor plugin offers generic support (table browsing and execution of SQL queries) for any JDBC-compliant database (for example, *MariaDB*).

To watch our video demonstration about the integration between the relational databases and Oxygen XML Editor plugin, go to *http://www.oxygenxml.com/demo/Author_Database_Integration.html*.

Configuring Relational Database Data Sources

This section describes the procedures for configuring the data sources for relational databases:

- *IBM DB2*
- *Microsoft SQL Server*
- *Generic JDBC*
- *MySQL*
- *Oracle 11g*
- *PostgreSQL 8.3*

Configuring Database Connections

This section describes the procedures for configuring the connections for relational databases:

- *IBM DB2*
- *Microsoft SQL Server*
- *JDBC-ODBC*
- *MySQL*
- *Generic ODBC*
- *Oracle 11g*
- *PostgreSQL 8.3*

How to Configure Support For Relational Databases

This section contains procedures about configuring the support for various relational databases.

How to Configure IBM DB2 Support

To configure the support for the IBM DB2 database follow this procedure:

1. Go to the *IBM website* and in the *DB2 Clients and Development Tools* category select the *DB2 Driver for JDBC and SQLJ* download link. Fill out the download form and download the zip file. Unzip the zip file and use the `db2jcc.jar` and `db2jcc_license_cu.jar` files in Oxygen XML Editor plugin for *configuring a DB2 data source*.
2. *Configure a IBM DB2 Data Source driver.*
3. *Configure a IBM DB2 Server Connection.*
4. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure an IBM DB2 Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to an IBM DB2 server are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

   The dialog box for configuring a data source is opened.

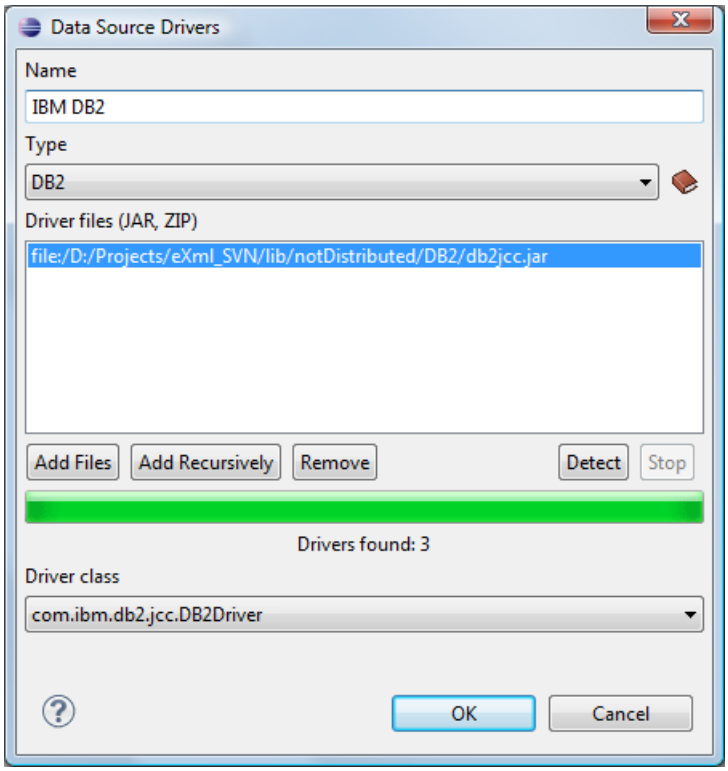


Figure 347: Data Source Drivers Configuration Dialog Box

3. Enter a unique name for the data source.
4. Select *DB2* in the driver **Type** drop-down menu.
5. Add the driver files for IBM DB2 using the **Add Files** button.

   The IBM DB2 driver files are:

   - `db2jcc.jar`
   - `db2jcc_license_cisuz.jar`
   - `db2jcc_license_cu.jar`

   The **Driver files** section lists *download links for database drivers* that are necessary for accessing IBM DB2 databases in Oxygen XML Editor plugin.

6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

To watch our video demonstration about running XQuery against an IBM DB2 Pure XML database, go to *http://www.oxygenxml.com/demo/DB2.html*.

How to Configure an IBM DB2 Connection

The support to create an IBM DB2 connection is available in the Enterprise edition only.

To configure a connection to an IBM DB2 server, follow these steps:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

   The dialog box for configuring a database connection is displayed.

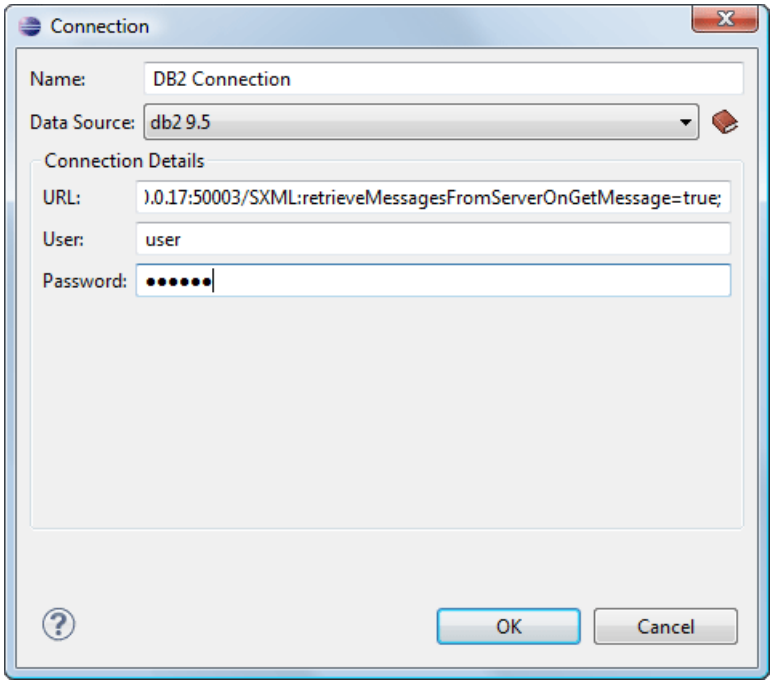


Figure 348: The Connection Configuration Dialog Box

3. Enter a unique name for the connection.
4. Select an *IBM DB2* data source in the **Data Source** drop-down menu.
5. Enter the connection details.
   a) Enter the URL to the installed IBM DB2 engine.
   b) Enter the user name to access the IBM DB2 engine.
   c) Enter the password to access the IBM DB2 engine.
6. Click the **OK** button to finish the configuration of the database connection.

To watch our video demonstration about running XQuery against an IBM DB2 Pure XML database, go to *http://www.oxygenxml.com/demo/DB2.html*.

How to Configure Microsoft SQL Server Support

To configure the support for Microsoft SQL Server database follow this procedure:

1. Download the appropriate MS SQL JDBC driver from the Microsoft website. For SQL Server 2008 R2 and older go to *http://www.microsoft.com/en-us/download/details.aspx?id=21599*. For SQL Server 2012 and 2014 go to *http://www.microsoft.com/en-us/download/details.aspx?id=11774*.
2. *Configure a MS SQL Server Data Source driver.*
3. *Configure a MS SQL Server Connection.*
4. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure a Microsoft SQL Server Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to a Microsoft SQL server are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the + **New** button in the **Data Sources** panel.

   The dialog box for configuring a data source is opened.

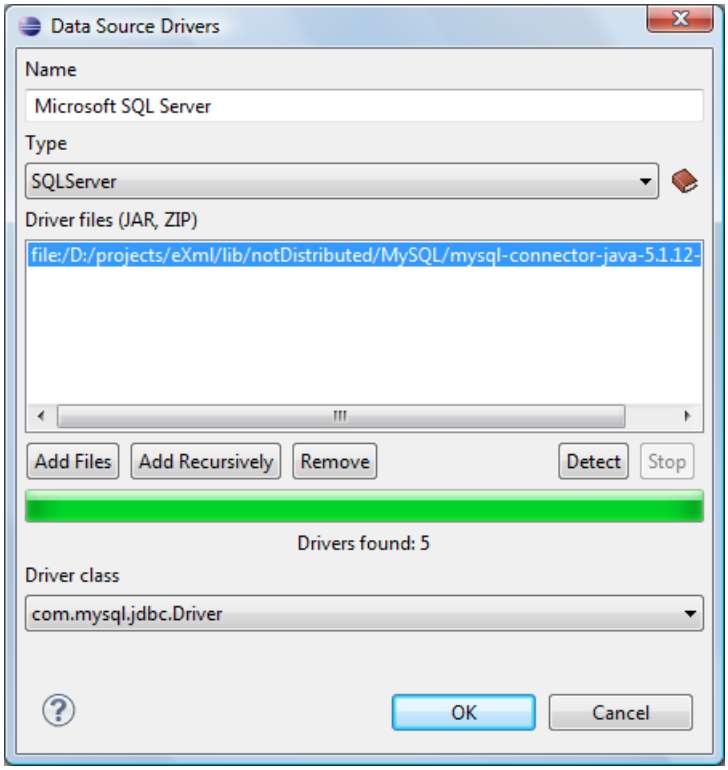


Figure 349: Data Source Drivers Configuration Dialog Box

3. Enter a unique name for the data source.
4. Select *SQLServer* in the driver **Type** drop-down menu.
5. Add the Microsoft SQL Server driver file using the **Add Files** button.

   The SQL Server driver file is called `sqljdbc.jar`. In the The **Driver files** section lists *download links for database drivers* that are necessary for accessing Microsoft SQL Server databases in Oxygen XML Editor plugin.
6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a Microsoft SQL Server Connection

The support to configure a Microsoft SQL Server connection is available in the Enterprise edition only.

To configure a connection to a Microsoft SQL Server, follow these steps:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the + **New** button.

   The dialog box for configuring a database connection is displayed.

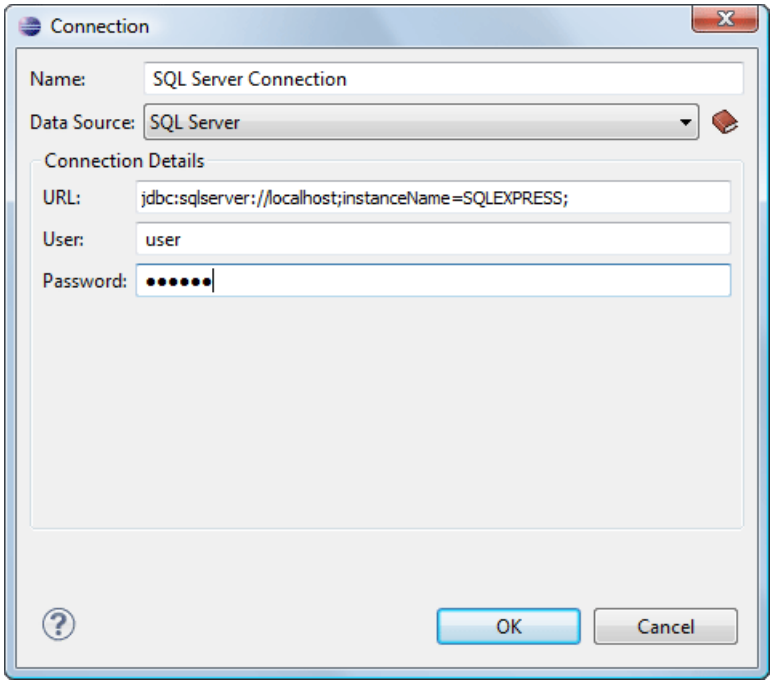


Figure 350: The Connection Configuration Dialog Box

3. Enter a unique name for the connection.
4. Select the *SQL Server* data source in the **Data Source** drop-down menu.
5. Enter the connection details.
   a) Enter the URL of the SQL Server server.

      If you want to connect to the server using Windows integrated authentication, you must add `;integratedSecurity=true` to the end of the URL. The URL will look like this:

      ```
      jdbc:sqlserver://localhost;instanceName=SQLEXPRESS;integratedSecurity=true;
      ```

      Note: For integrated authentication, leave the **User** and **Password** fields empty.

   b) Enter the user name for the connection to the SQL Server.
   c) Enter the password for the connection to the SQL Server.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure Generic JDBC Support

To configure the support for a generic JDBC database follow this procedure:

1. *Configure a Generic JDBC Data Source driver.*
2. *Configure a Generic JDBC Connection.*
3. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure a Generic JDBC Data Source

Starting with version 17, Oxygen XML Editor plugin comes bundled with Java 8, which does not provide built-in access to JDBC-ODBC data sources. To access such sources, you need to find an alternative JDBC-ODBC bridge or use a platform-independent distribution of Oxygen XML Editor plugin along with a Java VM version 7 or 6. The following procedure shows you how to configure a generic JDBC data source:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

3. Enter a unique name for the data source.
4. Select *Generic JDBC* in the driver **Type** drop-down list.
5. Add the driver file(s) using the **Add Files** button.
6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a Generic JDBC Connection

To configure a connection to a generic JDBC database, follow these steps:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.
3. Enter a unique name for the connection.
4. Select the *Generic JDBC* data source in the **Data Source** drop-down menu.
5. Enter the connection details.
   a) Enter the URL of the generic JDBC database, with the following format:`jdbc: <subprotocol>: <subname>`.
   b) Enter the user name for the connection to the generic JDBC database.
   c) Enter the password for the connection to the generic JDBC database.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure MySQL Support

To configure the support for a MySQL database follow this procedure:

1. *Configure a MySQL Data Source driver.*
2. *Configure a MySQL Connection.*
3. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure a MySQL Data Source

To connect to a MySQL server, create a data source of a generic JDBC type, based on *the MySQL JDBC driver available on the MySQL website*. The following steps describe how you can configure such a data source:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.

   The dialog box for configuring a data source is opened.

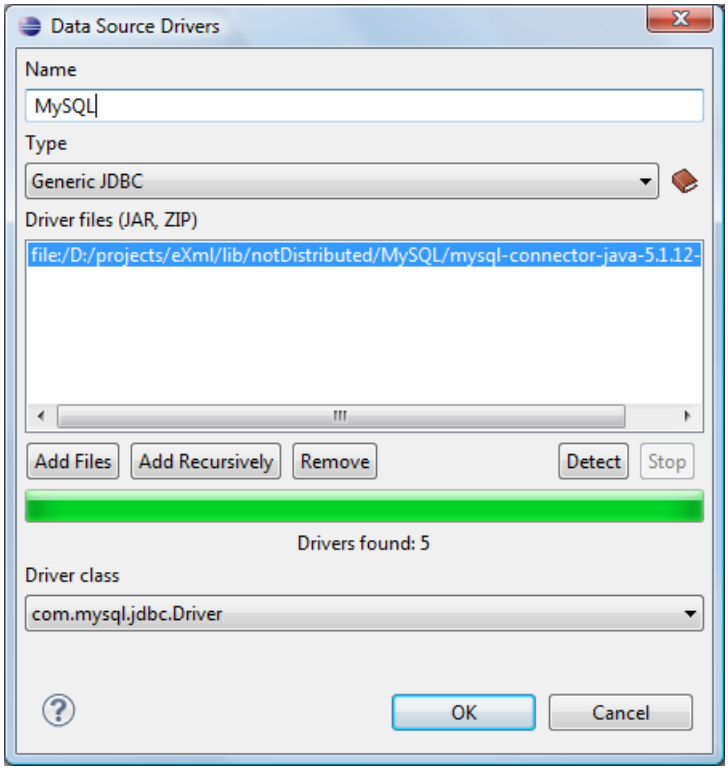


Figure 351: Data Source Drivers Configuration Dialog Box

3. Enter a unique name for the data source.
4. Select *Generic JDBC* in the driver **Type** drop-down list.
5. Add the MySQL driver files using the **Add Files** button.

   The driver file for the MySQL server is called `mysql-com.jar`. The **Driver files** section lists *download links for database drivers* that are necessary for accessing MySQL databases in Oxygen XML Editor plugin.
6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a MySQL Connection

To configure a connection to a MySQL server, follow these steps:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

   The dialog box for configuring a database connection is displayed.

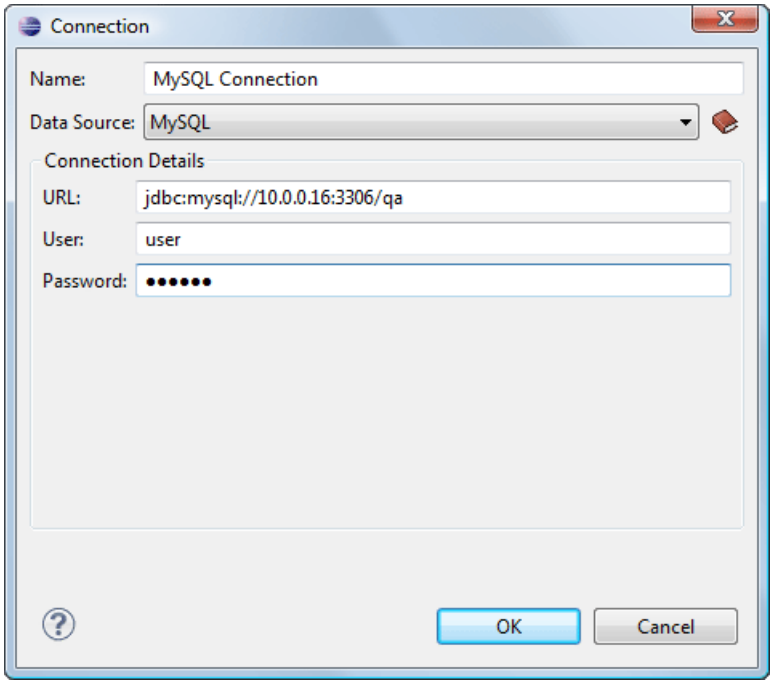


Figure 352: The Connection Configuration Dialog Box

3. Enter a unique name for the connection.
4. Select the *MySQL* data source in the **Data Source** drop-down list.
5. Enter the connection details.
   a) Enter the URL of the MySQL server.
   b) Enter the user name for the connection to the MySQL server.
   c) Enter the password for the connection to the MySQL server.
6. Click the **OK** button to finish the configuration of the database connection.

How to Oracle 11g Support

To configure the support for a Oracle 11g database follow this procedure:

1. Go to the *Oracle website* and download the Oracle 11g JDBC driver called `ojdbc6.jar`.
2. *Configure a Oracle 11g Data Source driver.*
3. *Configure a Oracle 11g Connection.*
4. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure an Oracle 11g Data Source

Available in the Enterprise edition only.

The steps for configuring a data source for connecting to an Oracle 11g server are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the + **New** button in the **Data Sources** panel.

   The dialog box for configuring a data source is opened.

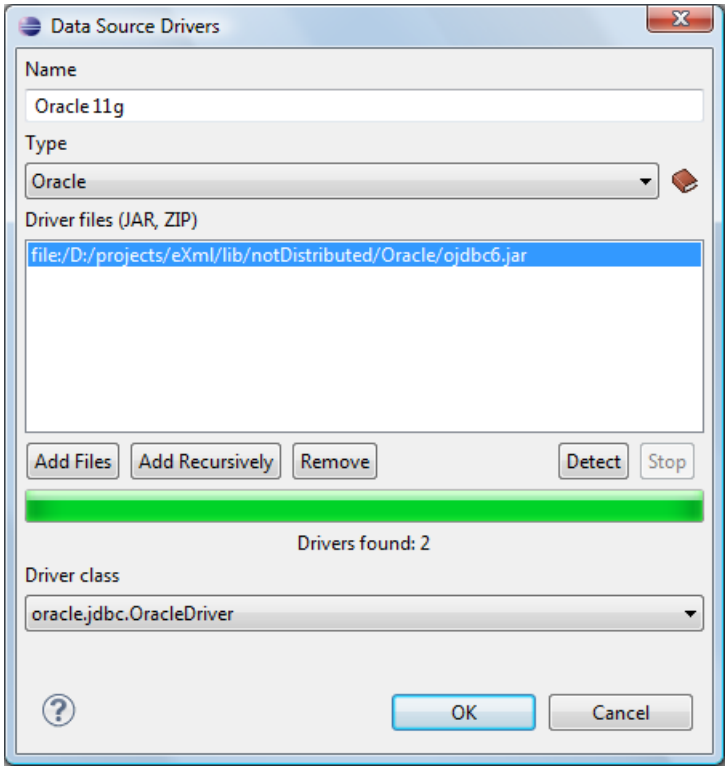


Figure 353: Data Source Drivers Configuration Dialog Box

3. Enter a unique name for the data source.
4. Select *Oracle* in the driver **Type** drop-down menu.
5. Add the Oracle driver file using the **Add Files** button.

   The Oracle driver file is called `ojdbc5.jar`. The **Driver files** section lists *download links for database drivers* that are necessary for accessing Oracle databases in Oxygen XML Editor plugin.
6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure an Oracle 11g Connection

Available in the Enterprise edition only.

The steps for configuring a connection to an Oracle 11g server are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

   The dialog box for configuring a database connection is displayed.

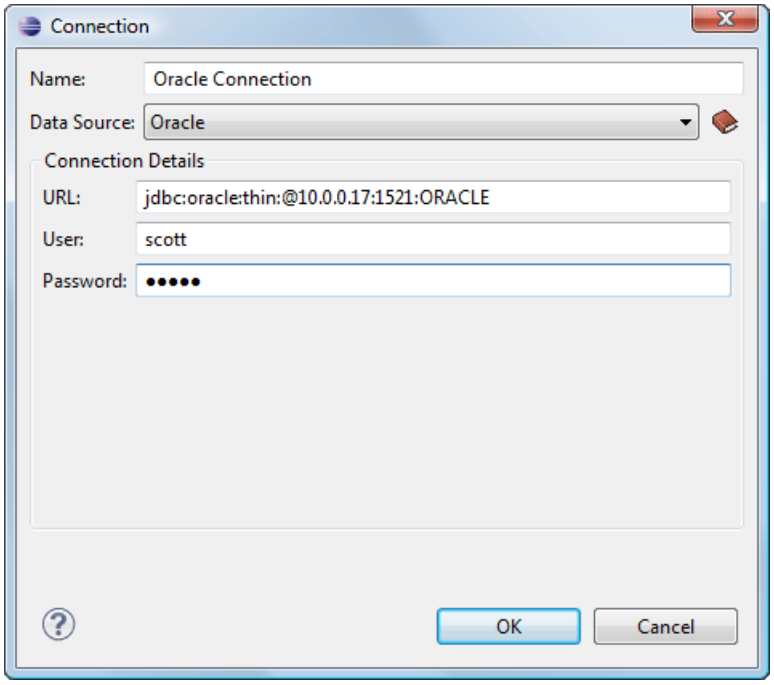


Figure 354: The Connection Configuration Dialog Box

3. Enter a unique name for the connection.
4. Select the *Oracle 11g* data source in the **Data Source** drop-down menu.
5. Enter the connection details.
   a) Enter the URL of the Oracle server.
   b) Enter the user name for the connection to the Oracle server.
   c) Enter the password for the connection to the Oracle server.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure PostgreSQL Support

To configure the support for a PostgreSQL database follow this procedure:

1. Go to the *PostgreSQL website* and download the PostgreSQL 8.3 JDBC driver called `postgresql-8.3-603.jdbc3.jar`.
2. *Configure a PostgreSQL Data Source driver.*
3. *Configure a PostgreSQL Connection.*
4. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure a PostgreSQL 8.3 Data Source

The steps for configuring a data source for connecting to a PostgreSQL server are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the + **New** button in the **Data Sources** panel.

   The dialog box for configuring a data source is opened.

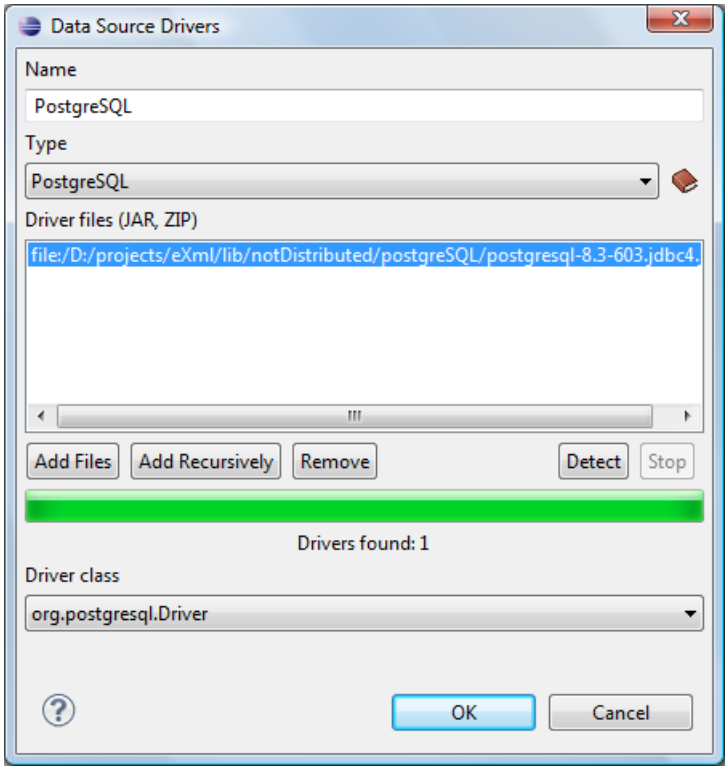


Figure 355: Data Source Drivers Configuration Dialog Box

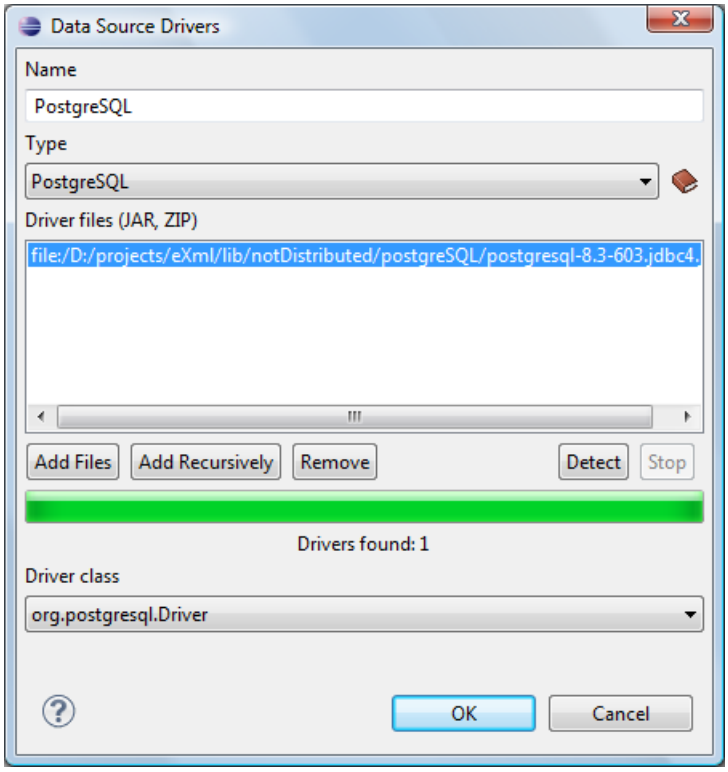


Figure 356: Data Source Drivers Configuration Dialog Box

3. Enter a unique name for the data source.
4. Select *PostgreSQL* in the driver **Type** drop-down list.

5. Add the PostgreSQL driver file using the **Add Files** button.

   The PostgreSQL driver file is called `postgresql-8.3-603.jdbc3.jar`. The **Driver files** section lists *download links for database drivers* that are necessary for accessing PostgreSQL databases in Oxygen XML Editor plugin.
6. Select the most appropriate **Driver class**.
7. Click the **OK** button to finish the data source configuration.

How to Configure a PostgreSQL 8.3 Connection

The steps for configuring a connection to a PostgreSQL 8.3 server are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

   The dialog box for configuring a database connection is displayed.

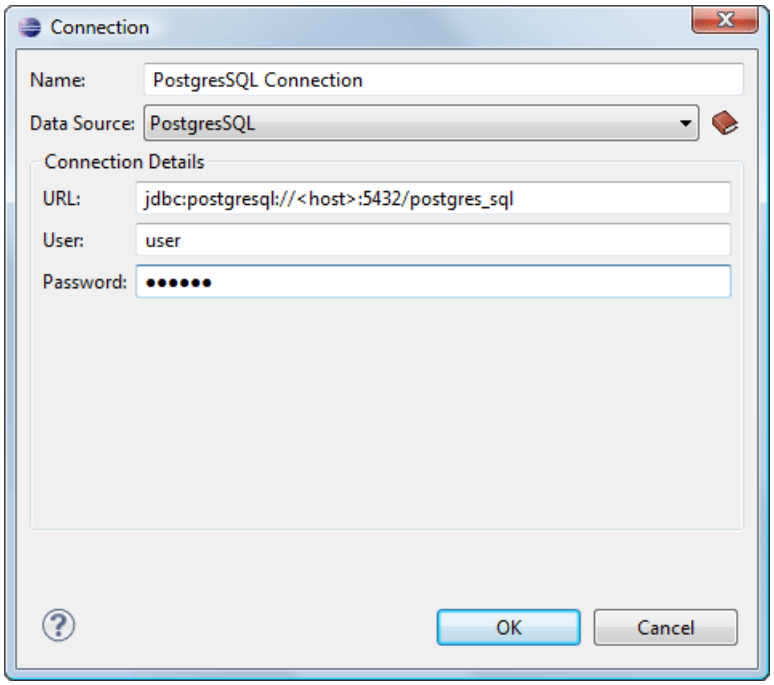


Figure 357: The Connection Configuration Dialog Box

3. Enter a unique name for the connection.
4. Select the *PostgreSQL 8.3* data source in the **Data Source** drop-down menu.
5. Enter the connection details.
   a) Enter the URL of the PostgreSQL 8.3 server.
   b) Enter the user name for the connection to the PostgreSQL 8.3 server.
   c) Enter the password for the connection to the PostgreSQL 8.3 server.
6. Click the **OK** button to finish the configuration of the database connection.

How to Configure JDBC-ODBC Support

To configure the support for a JDBC-ODBC database follow this procedure:

1. Configure a JDBC-ODBC *Data Source* driver.
2. *Configure a JDBC-ODBC Connection.*
3. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure a JDBC-ODBC Connection

Starting with version 17, Oxygen XML Editor plugin comes bundled with Java 8, which does not provide built-in access to JDBC-ODBC data sources. To access such sources, you need to find an alternative JDBC-ODBC bridge or use a platform-independent distribution of Oxygen XML Editor plugin along with a Java VM version 7 or 6. To configure a connection to an ODBC data source, follow these steps:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.

   The dialog box for configuring a database connection is displayed.

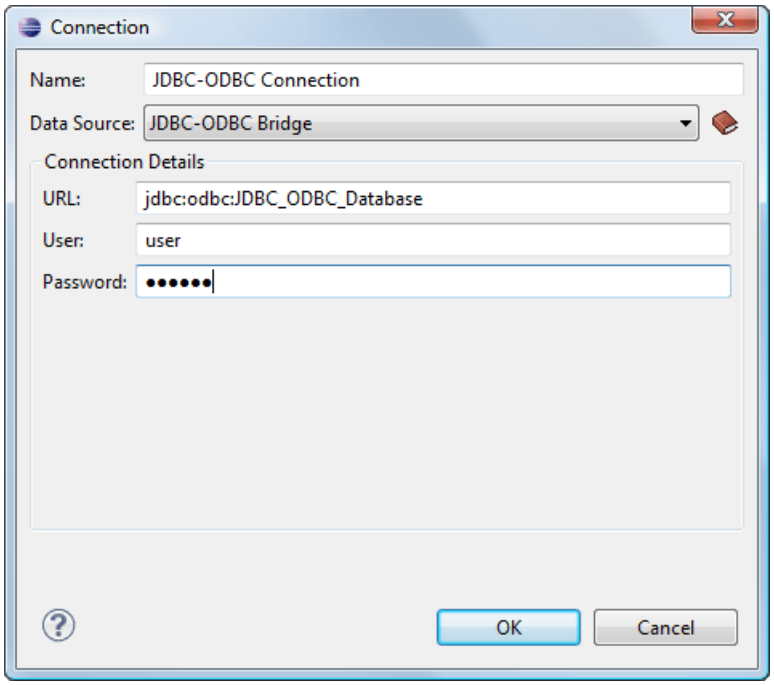


Figure 358: The Connection Configuration Dialog Box

3. Enter a unique name for the connection.
4. Select *JDBC-ODBC Bridge* in the **Data Source** drop-down list.
5. Enter the connection details.
   a) Enter the URL of the ODBC source.
   b) Enter the user name of the ODBC source.
   c) Enter the password of the ODBC source.
6. Click the **OK** button to finish the configuration of the database connection.

Resource Management

This section explains resource management actions for relational databases.

Data Source Explorer View

The **Data Source Explorer** view displays your database connections. You can connect to a database simply by expanding the connection node. The database structure can be expanded to the column level. Oxygen XML Editor plugin supports multiple simultaneous database connections and the connection tree provides an easy method for browsing them.

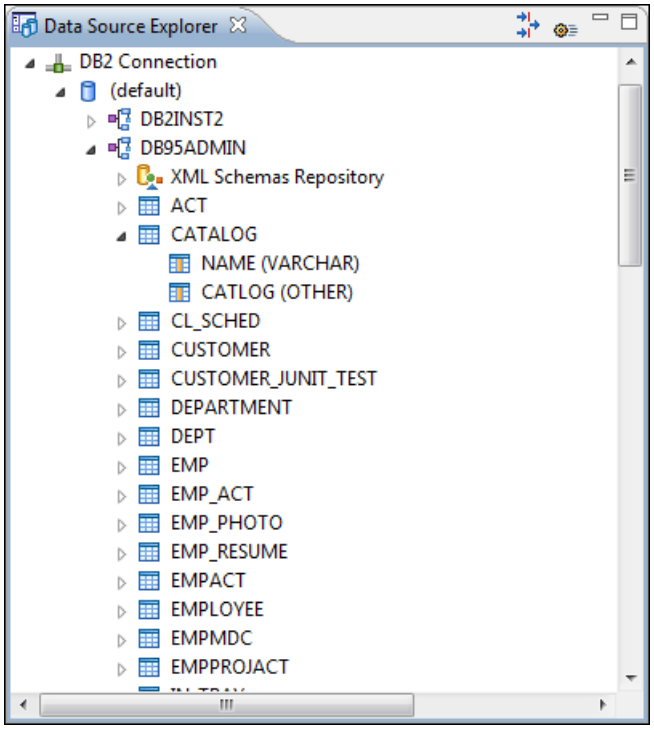


Figure 359: Data Source Explorer View

The following objects are displayed in the **Data Source Explorer** view:

- **Connection**
- **Collection (Catalog)**
- **XML Schema Repository**
- **XML Schema Component**
- **Schema**
- **Table**
- **System Table**
- **Table Column**

A **Collection** (called *catalog* in some databases) is a hierarchical container for resources and sub-collections. There are two types of resources:

- **XML resource** - An XML document or document fragment, selected by a previously executed XPath query.
- **non-XML resource** -Any resource that is not recognized as XML.

Note: For some connections you can add or move resources into a container by dragging them from:

- The **Project view**.
- The default file system application (for example, Windows Explorer in Windows or Finder in Mac OS X).
- Another database container.

The following actions are available in the toolbar of this view:

Filters

Opens the **Data Sources / Table Filters** *Preferences page*, allowing you to decide which table types are displayed in the **Data Source Explorer** view.

Configure Database Sources

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

Actions Available at Connection Level in Data Source Explorer View

The contextual menu of a **Connection** node in the tree from the **Data Source Explorer** view contains the following actions:

Refresh

Performs a refresh for the sub-tree of the selected node.

Disconnect

Closes the current database connection. If a table is already open, you are warned to close it before proceeding.

Configure Database Sources

Opens the ***Data Sources*** *preferences page* where you can configure both data sources and connections.

Actions Available at Catalog Level in Data Source Explorer View

The contextual menu of a **Catalog** node in the tree from the **Data Source Explorer** view contains the following actions:

Refresh

Performs a refresh for the sub-tree of the selected node.

Actions Available at Schema Level in Data Source Explorer View

The contextual menu of a **Schema** node in the tree from the **Data Source Explorer** view contains the following actions:

Refresh

Performs a refresh for the sub-tree of the selected node.

Actions Available at Table Level in Data Source Explorer View

The contextual menu of a **Table** node in the tree from the **Data Source Explorer** view contains the following actions:

Refresh

Performs a refresh for the sub-tree of the selected node.

Edit

Opens the selected table in the **Table Explorer** view.

Export to XML

Opens the **Export Criteria** dialog box (a thorough description of this dialog box can be found in the *Import from Database* chapter) .

XML Schema Repository Level

This section explains the actions available at the XML Schema Repository level.

Oracle XML Schema Repository Level

The Oracle database supports XML schema repository (XSR) in the database catalogs. The contextual menu of a **XML Schema Repository** node in the tree from the **Data Source Explorer** view contains the following actions:

Refresh

Performs a refresh for the sub-tree of the selected node.

Register

Opens a dialog box for adding a new schema file in the XML repository. To add an XML Schema, enter the schema URI and location on your file system. *Local* scope means that the schema is visible only to the user who registers it. *Global* scope means that the schema is public.

Note: Registering a schema may involve dropping/creating types. Hence you need type-related privileges such as DROP TYPE, CREATE TYPE, and ALTER TYPE. You need privileges to delete and register the XML schemas involved in the registering process. You need all privileges on XMLType tables that conform to the registered schemas. For XMLType columns, the ALTER TABLE privilege is needed on corresponding tables. If there are schema-based XMLType tables or columns in other database schemas, you need privileges such as the following:

- `CREATE ANY TABLE`
- `CREATE ANY INDEX`
- `SELECT ANY TABLE`
- `UPDATE ANY TABLE`
- `INSERT ANY TABLE`
- `DELETE ANY TABLE`
- `DROP ANY TABLE`
- `ALTER ANY TABLE`
- `DROP ANY INDEX`

To avoid having to grant all these privileges to the schema owner, Oracle recommends that the registration be performed by a DBA if there are XML schema-based XMLType table or columns in other user database schemas.

IBM DB2 XML Schema Repository Level

The contextual menu of a **XML Schema Repository** node in the tree from the **Data Source Explorer** view contains the following actions:

Refresh

Performs a refresh for the sub-tree of the selected node.

Register

Opens a dialog box for adding a new schema file in the XML Schema repository. In this dialog box, the following fields can be set:

- **XML schema file** - Location on your file system.
- **XSR name** - Schema name.
- **Comment** - Short comment (optional).
- **Schema location** - Primary schema name (optional).

Decomposition means that parts of the XML documents are stored in relational tables. Which parts map to which tables and columns are specified in the schema annotations. Schema dependencies management is done by using the **Add** and **Remove** buttons.

The actions available at **Schema** level are as follows:

Refresh

Performs a refresh of the selected node (and its sub-tree).

Unregister

Removes the selected schema from the XML Schema Repository.

View

Opens the selected schema in Oxygen XML Editor plugin.

Microsoft SQL Server XML Schema Repository Level

The contextual menu of a **XML Schema Repository** node in the tree from the **Data Source Explorer** view contains the following actions:

Refresh

Performs a refresh for the sub-tree of the selected node.

Register

Opens a dialog box for adding a new schema file in the DB XML repository. In this dialog box, you enter a collection name and the necessary schema files. XML Schema files management is done by using the **Add** and **Remove** buttons.

The actions available at **Schema** level are as follows:

Refresh

Performs a refresh of the selected node (and its sub-tree).

Add

Adds a new schema to the XML Schema files.

Unregister

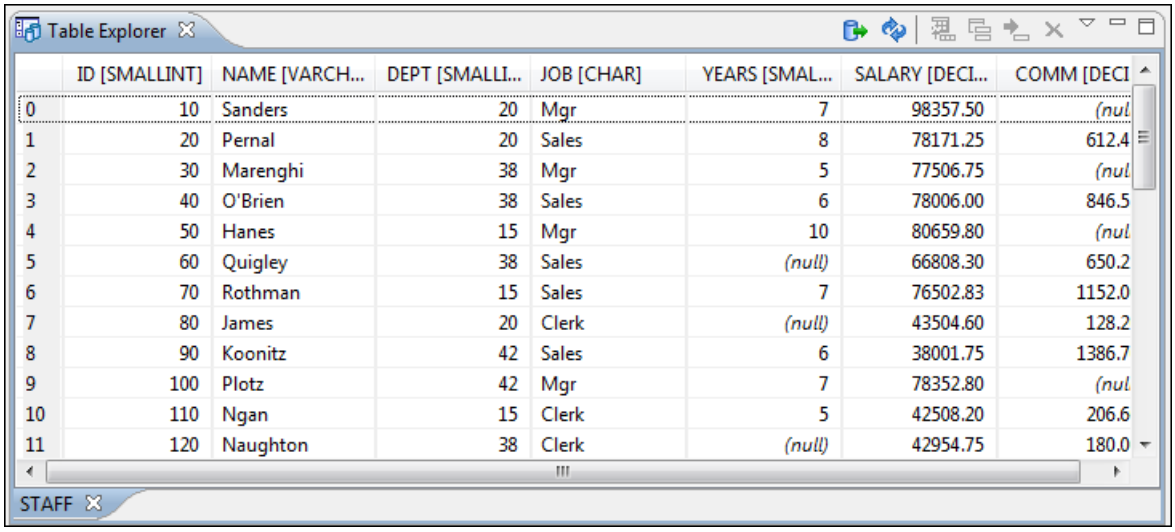
Removes the selected schema from the XML Schema Repository.

View

Opens the selected schema in Oxygen XML Editor plugin.

Table Explorer View

Every table from the **Data Source Explorer** view can be displayed and edited in the **Table Explorer** view by pressing the **Edit** button from the contextual menu or by double-clicking one of its fields. To modify the content of a cell, double-click it and start typing. When editing is complete, Oxygen XML Editor plugin attempts to update the database with the new cell content.



	ID [SMALLINT]	NAME [VARCH...	DEPT [SMALLI...	JOB [CHAR]	YEARS [SMAL...	SALARY [DECI...	COMM [DECI
0	10	Sanders	20	Mgr	7	98357.50	*(nul*
1	20	Pernal	20	Sales	8	78171.25	612.4
2	30	Marenghi	38	Mgr	5	77506.75	*(nul*
3	40	O'Brien	38	Sales	6	78006.00	846.5
4	50	Hanes	15	Mgr	10	80659.80	*(nul*
5	60	Quigley	38	Sales	*(null)*	66808.30	650.2
6	70	Rothman	15	Sales	7	76502.83	1152.0
7	80	James	20	Clerk	*(null)*	43504.60	128.2
8	90	Koonitz	42	Sales	6	38001.75	1386.7
9	100	Plotz	42	Mgr	7	78352.80	*(nul*
10	110	Ngan	15	Clerk	5	42508.20	206.6
11	120	Naughton	38	Clerk	*(null)*	42954.75	180.0

Figure 360: The Table Explorer View

You can sort the content of a table by one of its columns by clicking its column header.

Note the following:

- The first column is an index (not part of the table structure)
- Every column header contains the field name and its data type
- The primary key columns are marked with this symbol:

- Multiple tables are presented in a tabbed manner

For performance issues, you can set the maximum number of cells that are displayed in the **Table Explorer** view (using the **Limit the number of cells** field from the *Data Sources* Preferences page). If a table that has more cells than the value set in the options is displayed in the **Table Explorer** view, a warning dialog box informs you that the table is only partially shown.

You are notified if the value you have entered in a cell is not valid (and thus cannot be updated).

- If the content of the edited cell does not belong to the data type of the column, an information dialog box appears, notifying you that the value you have inserted cannot be converted to the SQL type of that field. For example, if you have a column that contains `LONG` (numerical) values, and a character or string is inserted into one of its cells, you would get the error message that a string value cannot be converted to the requested SQL type (NUMBER).
- If the constraints of the database are not met (for instance, primary key constraints), an information dialog box will appear, notifying you of the reason the database has not been updated. For example, in the table below, trying to set the second record in the primary key `propID` column to 8, results in a duplicate entry error since that value has already been used in the first record:

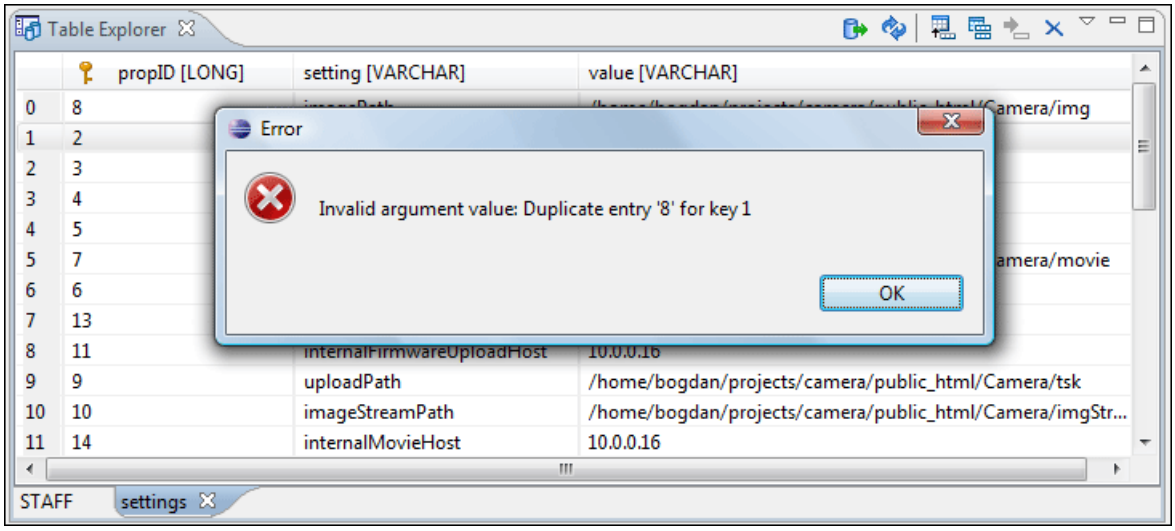


Figure 361: Duplicate Entry for Primary Key

Common edit actions (**Cut**, **Copy**, **Paste**, **Select All**, **Undo**, **Redo**) are available in the contextual menu of an edited cell.

The contextual menu, available on every cell, also has the following actions:

Set NULL

Sets the content of the cell to *null*. This action is disabled for columns that cannot have a value of *null*.

Insert row

Inserts an empty row in the table.

Duplicate row

Makes a copy of the selected row and adds it in the **Table Explorer** view. Note that the new row will not be inserted in the database table until all conflicts are resolved.

Commit row

Commits the selected row.

Delete row

Deletes the selected row.

 **Copy**

Copies the content of the cell.

 **Paste**

Pastes copied content into the selected cell.

The **Table Explorer** toolbar also includes the following actions:

 **Export to XML**

Opens the **Export Criteria** dialog box (a thorough description of this dialog box can be found in the *Import from database* chapter) .

 **Refresh**

Performs a refresh for the sub-tree of the selected node.

 **Insert row**

Inserts an empty row in the table.

 **Duplicate row**

Makes a copy of the selected row and adds it in the **Table Explorer** view. Note that the new row will not be inserted in the database table until all conflicts are resolved.

 **Commit row**

Commits the selected row.

 **Delete row**

Deletes the selected row.

SQL Execution Support

Oxygen XML Editor plugin's support for writing SQL statements includes syntax highlighting, folding, and dragging and dropping from the **Data Source Explorer** view. It also includes transformation scenarios for executing the statements, and the results are displayed in the **Table Explorer** view.

Drag and Drop from Data Source Explorer View

Drag and drop operations from the **Data Source Explorer** view to the SQL Editor allows you to create SQL statements quickly by inserting the names of tables and columns in the SQL statements.

1. Configure a database connection (*see the specific procedure for your database server*).
2. Browse to the table you will use in your statement.
3. Drag the table or a column of the table into the editor where a SQL file is open.

   Drag and drop actions are available both on the table and on its fields. A pop-up menu is displayed in the SQL editor.

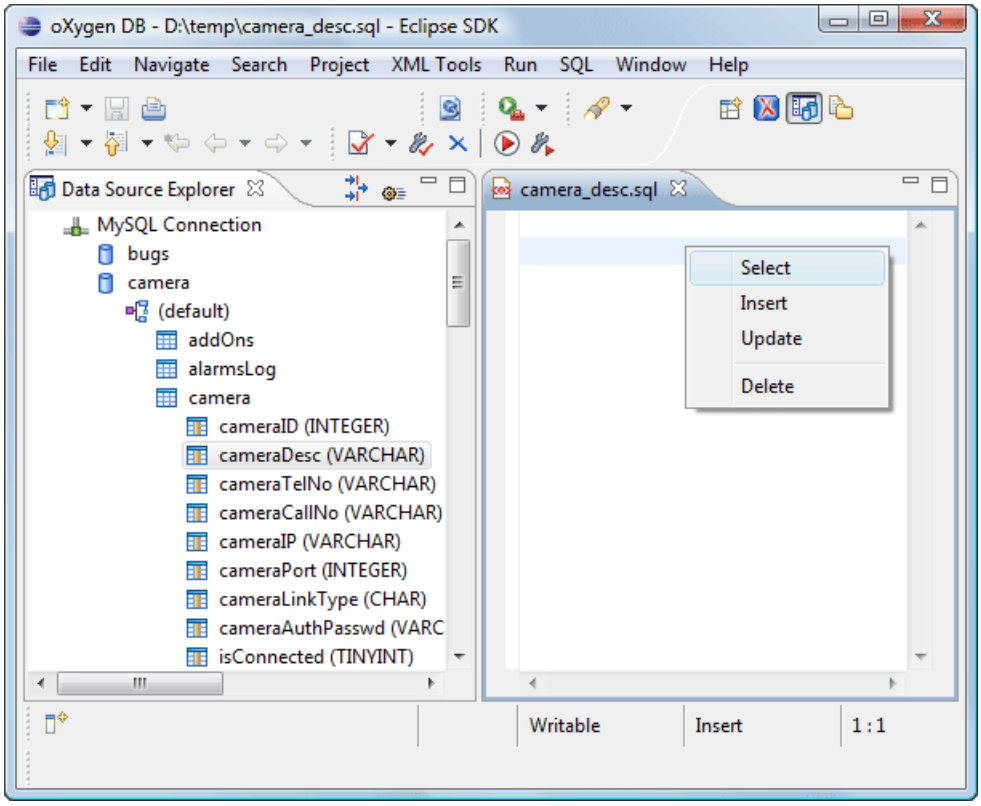


Figure 362: SQL Statement Editing with Drag and Drop

4. Select the type of statement from the pop-up menu.

   Depending on your choice, dragging a table results in one of the following statements being inserted into the document:

   - SELECT `*field1*`,`*field2*`, ... FROM `*catalog*`.`*table*` (for example: `SELECT \`DEPT\`,\`DEPTNAME\`,\`LOCATION\` FROM \`camera\`.\`cameraDesc\``)
   - UPDATE `*catalog*`.`*table*` SET `*field1*`=, `*field2*`=,.... (for example: `UPDATE \`camera\`.\`cameraDesc\` SET \`DEPT\`=, \`DEPTNAME\`=, \`LOCATION\`=`)
   - INSERT INTO`*catalog*`.`*table*` (`*field1*`,`*field2*`, ...) VALUES (, ,) (for example: `INSERT INTO \`camera\`.\`cameraDesc\` (\`DEPT\`,\`DEPTNAME\`,\`LOCATION\`) VALUES (, , ))`
   - DELETE FROM `*catalog*`.`*table*` (for example: `DELETE FROM \`camera\`.\`cameraDesc\``)

   Depending on your choice, dragging a column results in one of the following statements being inserted into the document:

   - SELECT `*field*` FROM `*catalog*`.`*table*` (for example: `SELECT \`DEPT\` FROM \`camera\`.\`cameraDesc\``)
   - UPDATE `*catalog*`.`*table*` SET `*field*`= (for example: `UPDATE \`camera\`.\`cameraDesc\` SET \`DEPT\`=`)
   - INSERT INTO`*catalog*`.`*table*` (`*field1*`) VALUES () (for example: `INSERT INTO \`camera\`.\`cameraDesc\` (\`DEPT\`) VALUES ()`)
   - DELETE FROM `*catalog*`.`*table*` (for example: `DELETE FROM \`camera\`.\`cameraDesc\` WHERE \`DEPT\`=`)

SQL Validation

SQL validation support is offered for IBM DB2. Note that if you choose a connection that does not support SQL validation, you will receive a warning when trying to validate. The SQL document is validated using the connection from the associated transformation scenario.

Executing SQL Statements

The steps for executing an SQL statement on a relational database are as follows:

1. Configure a *transformation scenario* using the **Configure Transformation Scenario(s)** action from the **Transformation** toolbar or the **XML** menu.

   A SQL transformation scenario needs a database connection. You can configure a connection using the **Preferences** button from the SQL transformation dialog box.

   The dialog box contains the list of existing scenarios that apply to SQL documents.
2. Set parameter values for SQL placeholders using the **Parameters** button from the SQL transformation dialog box. For example, in `SELECT * FROM `test`.`department` where DEPT = ? or DEPTNAME = ?` the two parameters can be configured for the place holders (?) in the transformation scenario.

   When the SQL statement is executed, the first placeholder is replaced with the value set for the first parameter in the scenario, the second placeholder is replaced by the second parameter value, and so on.

   Restriction: When a stored procedure is called in an SQL statement executed on an SQL Server database, mixing in-line parameter values with values specified using the **Parameters** button of the scenario dialog box is not recommended. This is due to a limitation of the SQL Server driver for Java applications. An example of stored procedure that is not recommended: `call dbo.Test(22, ?)`.
3. Execute the SQL scenario by clicking the **OK** or **Apply associated** button.

   The result of a SQL transformation is *displayed in a view* at the bottom of the Oxygen XML Editor plugin window.
4. View more complex return values of the SQL transformation in a separate editor panel.

   A more complex value returned by the SQL query (for example, an *XMLTYPE* or *CLOB* value) cannot be displayed entirely in the result table.

   a) Right-click the cell containing the complex value.
   b) Select the action **Copy cell** from the contextual menu.
   The action copies the value in the clipboard.
   c) Paste the value into an appropriate editor.
   For example, you can paste the value in an opened XQuery editor panel of Oxygen XML Editor plugin.

Native XML Database (NXD) Support

Native XML databases have an XML-based internal model and their fundamental unit of storage is XML. Oxygen XML Editor plugin offers support for the following native XML databases:

- Berkeley DB XML
- eXist
- MarkLogic
- 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 Editor plugin, go to *http://www.oxygenxml.com/demo/Author_Database_XML_Native.html*.

Configuring Native XML Database Data Sources

This section describes the procedures for configuring the following native database data sources:

- *Berkeley DB XML*
- *eXist*
- *MarkLogic*
- *Documentul xDB (X-Hive/DB) 10*

Configuring Database Connections

This section describes the procedures for configuring the connections for the following native databases:

- *Berkeley DB XML*
- *eXist*
- *MarkLogic*
- *Documentum xDb (X-Hive/DB) 10*

How to Configure Support for Native XML Databases

This section contains procedures about configuring the support for various native XML databases.

How to Configure Berkeley DB XML Support

Follow this procedure to configure the support for a Berkeley DB XML database:

1. *Configure a Berkeley DB XML Data Source driver.*
2. *Configure a Berkeley DB XML Connection.*
3. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure a Berkeley DB XML Data Source

Oxygen XML Editor plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The steps for configuring a data source for a Berkeley DB XML database are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *Berkeley DBXML* from the driver **Type** drop-down menu.
5. Click 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 in `[DBXML_DIR]/lib` or `[DBXML_DIR]/jar`)
   - `dbxml.jar` (check for it in `[DBXML_DIR]/lib` or `[DBXML_DIR]/jar`)

   Where `[DBXML_DIR]` is the Berkeley DB XML database root directory. For example, in Windows it is: `C:\Program Files\Oracle\Berkeley DB XML <version>`.
6. Click the **OK** button to finish the data source configuration.

How to Configure a Berkeley DB XML Connection

Oxygen XML Editor plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. The steps for configuring a connection to a Berkeley DB XML database are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down menu.
5. Enter the connection details.
   a) Set the path to the Berkeley DB XML `database` directory in the **Environment home directory field**. Use a directory with write access. DO NOT use the installation directory where Berkeley DB XML is installed if you do not have write access to that directory.
   b) Select the **Verbosity** level: *DEBUG*, *INFO*, *WARNING*, or *ERROR*.
   c) Optionally, you can select the check-box **Join existing environment**.

      If checked, an attempt is made to join an existing environment in the specified home directory and all the original environment settings are preserved. If that fails, try reconfiguring the connection with this option unchecked.

6. Click the **OK** button to finish the connection configuration.

How to Configure eXist Support

Follow this procedure to configure the support for an eXist database:

1. *Configure a eXist Data Source driver.*
2. *Configure a eXist Connection.*
3. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure an eXist Data Source

Oxygen XML Editor plugin supports eXist database server versions up to and including version 2.2. The steps for configuring a data source for an eXist database are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *eXist* from the driver **Type** drop-down menu.
5. Click the **Add** button to add the eXist driver files.

   The following driver files should be added in the dialog box for setting up the eXist datasource. They are found in the installation directory of the eXist database server. Make sure you copy the files from the installation of the eXist server where you want to connect from Oxygen XML Editor plugin.

   - `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`
   - `lib/core/slf4j-api-1.x.x.jar` (if available)
   - `lib/core/slf4j-log4j12-1.x.x.jar` (if available)

   The version number from the driver file names may be different for your eXist server installation.

6. Click the **OK** button to finish the connection configuration.

   To watch our video demonstration about running XQuery against an eXist XML database, go to *http://www.oxygenxml.com/demo/eXist_Database.html*.

How to Configure an eXist Connection

The steps for configuring a connection to an eXist database are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down menu.
5. Enter the connection details.
   a) Set the URI to the installed eXist engine in the **XML DB URI** field.
   b) Set the user name in the **User** field.
   c) Set the password in the **Password** field.
   d) Enter the start collection in the **Collection** field.

      eXist organizes all documents in hierarchical collections. Collections are like directories. They are used to group related documents together. This text field allows the user to set the default collection name.

6. Click the **OK** button to finish the connection configuration.

To watch our video demonstration about running XQuery against an eXist XML database, go to *http://www.oxygenxml.com/demo/eXist_Database.html*.

The Create eXist-db XML Connection Dialog Box

A quick way to create an eXist connection is to use the dedicated **Create eXist-db XML connection** dialog box. *Open the **Preferences** dialog box*, go to **Data Sources** and click the **Create eXist-db XML connection** link. After you fill in the fields, click **OK** and go to **Window** > **Show View** > **Data Source Explorer** to view your connection.

To create an eXist connection using this dialog box, Oxygen XML Editor plugin expects the `exist/webstart/exist.jnlp` path to be accessible at the provided **Host** and **Port**.

How to Configure MarkLogic Support

Follow this procedure to configure the support for a MarkLogic database:

1. Download the MarkLogic driver from *MarkLogic Community site*.
2. *Configure a MarkLogic Data Source driver.*
3. *Configure a MarkLogic Connection.*
4. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure a MarkLogic Data Source

Available in the Enterprise edition only.

Note: Oxygen XML Editor plugin supports MarkLogic version 4.0 or later.

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *MarkLogic* from the driver **Type** drop-down list.
5. Click the **Add** button to add the MarkLogic driver file (`marklogic-xcc-{server_version}`, where *{server_version}* is the MarkLogic server version.)

   You can download the driver file from: *http://community.marklogic.com/download*.

6. Click the **OK** button to finish the data source configuration.

How to Configure a MarkLogic Connection

Available in the Enterprise edition only.

Note: Oxygen XML Editor plugin supports MarkLogic version 4.0 or later.

The steps for configuring a connection to a MarkLogic database are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down menu.
5. Enter the connection details.
   a) The host name or IP address of the installed MarkLogic engine in the **XDBC Host** field.

      Oxygen XML Editor plugin uses XCC connector to interact with MarkLogic XDBC server and requires the basic authentication schema to be set. Starting with version MarkLogic 4.0 the default authentication method when you create a HTTP or WebDAV Server is *digest*, so make sure to change it to *basic*.

   b) Set the port number of the MarkLogic engine 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 process XQuery expressions against the

server. Later, if you want to change the XDBC server, instead of editing the configuration just use the ***Use it to execute queries*** action from Data Source Explorer.

c) Set the user name to access the MarkLogic engine in the **User** field.

d) Set the password to access the MarkLogic engine in the **Password** field.

e) Optionally set the URL used for browsing the MarkLogic database in the **Data Source Explorer** view in the **WebDAV URL** field.

The **Database** field specifies the database over which the XQuery expressions are executed. If you set this option to default, the database associated to the application server of the configured port is used.

6. Click the **OK** button to finish the connection configuration.

How to Configure Documentum xDb (X-Hive/DB) 10 Support

Follow this procedure to configure the support for a Documentum xDb (X-Hive/DB) 10 database:

1. *Configure a Documentum xDb Data Source driver.*
2. *Configure a Documentum xDb Connection.*
3. Use the **Data Source Explorer** view from the **Window** > **Show View** > **Data Source Explorer** menu or switch to the **Database Perspective** (available from **Window** > **Open Perspective** > **Database**).

How to Configure a Documentum xDb (X-Hive/DB) 10 Data Source

Available in the Enterprise edition only.

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Data Sources** panel.
3. Enter a unique name for the data source.
4. Select *XHive* from the driver **Type** drop-down menu.
5. Click the **Add** button to add the XHive driver files.

   The driver files for the Documentum xDb (X-Hive/DB) 10 database are found in the Documentum xDb (X-Hive/DB) 10 lib directory from the server installation folder:

   - `antlr-runtime.jar`
   - `aspectjrt.jar`
   - `icu4j.jar`
   - `xhive.jar`
   - `google-collect.jar`

6. Click the **OK** button to finish the data source configuration.

How to Configure an Documentum xDb (X-Hive/DB) 10 Connection

The steps for configuring a connection to a Documentum xDb (X-Hive/DB) 10 database are as follows:

Note: The bootstrap type of X-Hive/DB connections is not supported in Oxygen XML Editor plugin. The following procedure explains the `xhive://` protocol connection type.

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down menu.
5. Enter the connection details.

   a) Set the URL property of the connection in the **URL** field.

   If the property is a URL of the form *xhive://host:port*, the Documentum xDb (X-Hive/DB) 10 connection will attempt to connect to a Documentum xDb (X-Hive/DB) 10 server running behind the specified TCP/IP port.

   b) Set the user name to access the Documentum xDb (X-Hive/DB) 10 engine in the **User** field.

c) Set the password to access the Documentum xDb (X-Hive/DB) 10 engine in the **Password** field.
d) Set the name of the database to access from the Documentum xDb (X-Hive/DB) 10 engine in the **Database** field.
e) Check the **Run XQuery in read / write session (with committing)** checkbox if you want to end the session with a commit. Otherwise, the session ends with a rollback.

6. Click the **OK** button to finish the connection configuration.

Data Source Explorer View

The **Data Source Explorer** view displays your database connections. You can connect to a database simply by expanding the connection node. The database structure can be expanded to the column level. supports multiple simultaneous database connections and the connection tree provides an easy method for browsing them.

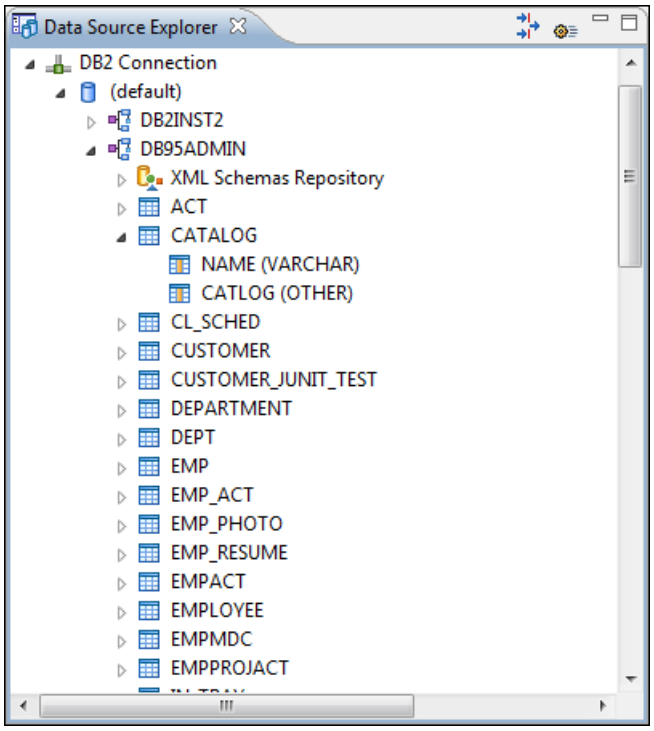


Figure 363: Data Source Explorer View

The following objects are displayed in the **Data Source Explorer** view:

- **Connection**
- **Collection (Catalog)**
- **XML Schema Repository**
- **XML Schema Component**
- **Schema**
- **Table**
- **System Table**
- **Table Column**

A **Collection** (called *catalog* in some databases) is a hierarchical container for resources and sub-collections. There are two types of resources:

- **XML resource** - An XML document or document fragment, selected by a previously executed XPath query.
- **non-XML resource** -Any resource that is not recognized as XML.

Note: For some connections you can add or move resources into a container by dragging them from:

- The **Project view**.
- The default file system application (for example, Windows Explorer in Windows or Finder in Mac OS X).
- Another database container.

The following actions are available in the toolbar of this view:

Filters

Opens the **Data Sources / Table Filters** *Preferences page*, allowing you to decide which table types are displayed in the **Data Source Explorer** view.

Configure Database Sources

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

Oracle XML DB Browser

Oracle XML DB is a feature of the Oracle database. It provides a high-performance, native XML storage and retrieval technology. Oxygen XML Editor plugin allows you to browse the native Oracle XML Repository and perform various operations on the resources in the repository.

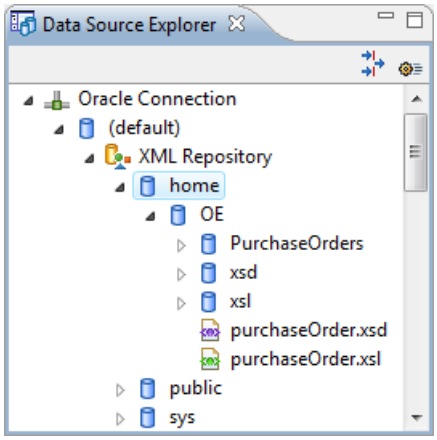


Figure 364: Browsing the Oracle XML DB Repository

The actions available at XML Repository level are as follows:

Refresh

Performs a refresh of the XML Repository.

Add container

Adds a new child container to the XML Repository.

Add resource

Adds a new resource to the XML Repository.

The actions available at container level are as follows:

Refresh

Performs a refresh of the selected container.

Add container

Adds a new child container to the current one.

Add resource

Adds a new resource to the folder.

Delete

Deletes the current container.

Properties

Shows various properties of the current container.

The actions available at resource level are as follows:

 **Refresh**

Performs a refresh of the selected resource.

Open

Opens the selected resource in the editor.

Rename

Renames the current resource

Move

Moves the current resource to a new container (also available through drag and drop).

Delete

Deletes the current resource.

Copy location

Allows you to copy (to the clipboard) an application-specific URL for the resource that can then be used for various actions, such as opening or transforming the resources.

Properties

Shows various properties of the current resource.

Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

For running an XQuery transformation on collections from the XML Repository, see *a tutorial from Oracle*.

PostgreSQL Connection

Oxygen XML Editor plugin allows you to browse the structure of the PostgreSQL database in the **Data Source Explorer** view and open the tables in the **Table Explorer** view.

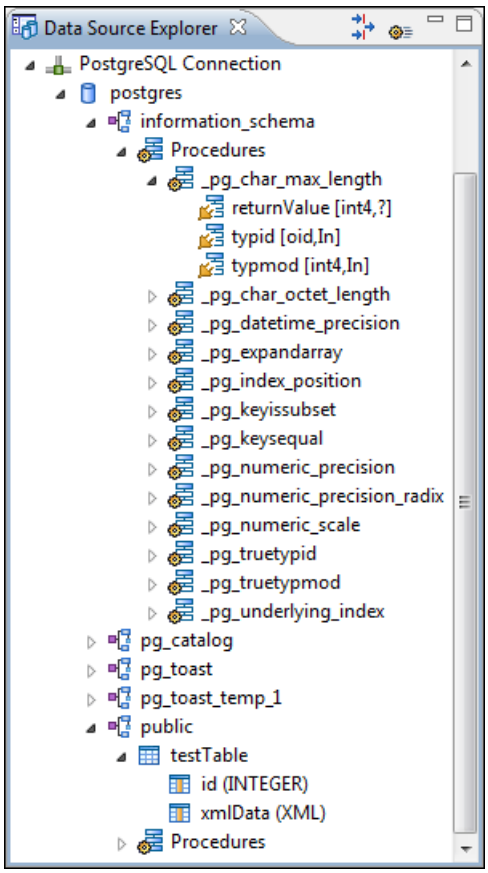


Figure 365: Browsing a PostgreSQL repository

The actions available at container level are as follows:

Refresh

Performs a refresh of the selected container.

The actions available at resource level are as follows:

Refresh

Performs a refresh of the selected database table.

Edit

Opens the selected database table in the **Table Explorer** view.

Export to XML

Exports the content of the selected database table as an XML file using *the dialog box from importing data from a database*.

Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

Berkeley DB XML Connection

This section explains the actions that are available on a Berkeley DB XML connection.

Actions Available at Connection Level

In a Berkeley DB XML repository, the actions available at connection level in the **Data Source Explorer** view are as follows:

Refresh

Performs a refresh for the sub-tree of the selected node.

Disconnect

Closes the current database connection.

Configure Database Sources

Opens *the **Data Sources** preferences page* where you can configure both data sources and connections.

Add container

Adds a new container in the repository with the following attributes:

- **Name** - The name of the new container.
- **Container type** - At creation time, every container must have a type defined for it. This container type identifies how XML documents are stored in the container. As such, the container type can only be determined at container creation time. You cannot change it when subsequent container opens. Containers can have one of the following types specified for them:
  - **Node container** - XML documents are stored as individual nodes in the container. Each record in the underlying database contains a single leaf node, its attributes and attribute values (if any), and its text nodes (if any). Berkeley DB XML also keeps the information it requires to reassemble the document from the individual nodes stored in the underlying databases. This is the default, and preferred, container type.
  - **Whole document container** - The container contains entire documents. The documents are stored without any manipulation of line breaks or whitespace.
- **Allow validation** - If checked, it causes documents to be validated when they are loaded into the container. The default behavior is to not validate documents.
- **Index nodes** - If checked, it causes indices for the container to return nodes rather than documents. The default is to index at the document level. This property has no meaning if the container type is **Whole document container**.

Properties

Displays a dialog box that contains a list of the Berkeley connection properties (*version*, *home location*, *default container type*, *compression algorithm*, etc.)

Actions Available at Container Level

In a Berkeley DB XML repository, the actions available at container level in the **Data Source Explorer** view are as follows:

Add Resource

Adds a new XML resource to the selected container.

Rename

Allows you to specify a new name for the selected container.

Delete

Removes the selected container from the database tree.

Edit indices

Allows you to edit the indices for the selected container.

Refresh

Performs a refresh for the sub-tree of the selected node.

Properties

Displays a dialog box with a list of properties of the Berkeley container (such as *container type*, *auto indexing*, *page size*, *validate on load*, *compression algorithm*, *number of documents*, etc.)

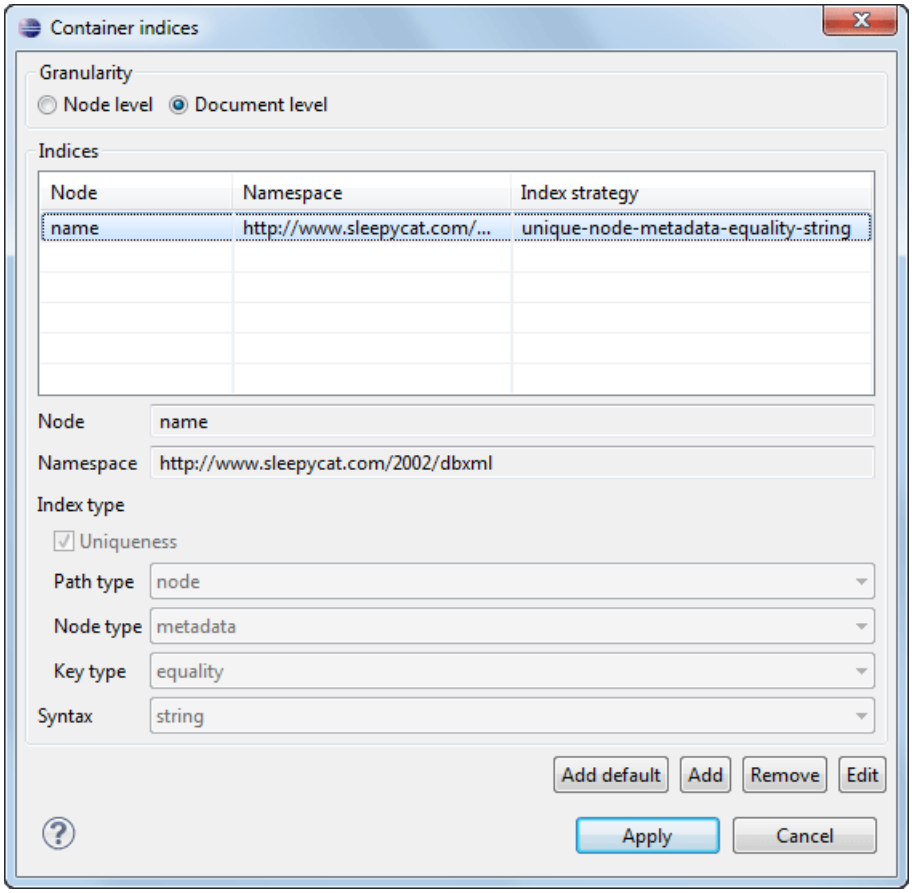


Figure 366: Container indices

The fields of the dialog box are as follows:

- Granularity:
  - **Document level** - Good option for retrieving large documents.
  - **Node level** - Good option for retrieving nodes from within documents.
- Add / Edit indices:
  - **Node** - The node name.
  - **Namespace** - The index namespace.
  - Index strategy:
    - **Index type**:
      - **Uniqueness** - Indicates whether the indexed value must be unique within the container.
      - **Path type**:
        - **node** - Indicates that you want to index a single node in the path.
        - **edge** - Indicates that you want to index the portion of the path where two nodes meet.
    - **Node type**:
      - **element** - An element node in the document content.
      - **attribute** - An attribute node in the document content.
      - **metadata** - A node found only in the metadata content of a document.
    - **Key type**:
      - **equality** - Improves the performances of tests that look for nodes with a specific value.

- **presence** - Improves the performances of tests that look for the existence of a node regardless of its value.
- **substring** - Improves the performance of tests that look for a node whose value contains a given sub-string.

- **Syntax types** - The syntax describes the type of data the index contains and is mostly used to determine how indexed values are compared.

Actions Available at Resource Level

In a Berkeley DB XML repository, the actions available at resource level in the **Data Source Explorer** view are as follows:

 **Refresh**

Performs a refresh of the selected resource.

Open

Opens the selected resource in the editor.

Rename

Allows you to change the name of the selected resource.

Move

Allows you to move the selected resource in a different container in the database tree (also available through drag and drop).

 **Delete**

Removes the selected resource from the container.

Copy location

Allows you to copy (to the clipboard) an application-specific URL for the resource that can then be used for various actions, such as opening or transforming the resources.

Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

eXist Connection

This section explains the actions that are available on an eXist connection.

Actions Available at Connection Level

For an eXist database, the actions available at connection level in the **Data Source Explorer** view are as follows:

 **Configure Database Sources**

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

Disconnect

Closes the current database connection.

 **Refresh**

Performs a refresh for the sub-tree of the selected node.

Actions Available at Container Level

For an eXist database, the actions available at container level in the **Data Source Explorer** view are as follows:

New File

Creates a file in the selected container.

New Collection

Creates a collection.

Import Folders

Adds the content of specified folders from the local file system.

Import Files

Adds a set of XML resources from the local file system.

Cut

Cuts the selected containers.

Copy

Copies the selected containers.

Note: You can add or move resources into the container by dragging them from the **Project** view, the default file management application (for example, Windows Explorer on Windows or Finder on OS X), or from another database container.

Paste

Paste resources into the selected container.

Rename

Allows you to change the name of the selected collection.

Delete

Removes the selected collection.

Refresh

Performs a refresh of the selected container.

Properties

Allows you to view various useful properties associated with the container, such as *name*, *creation date*, *owner*, *group*, or *permissions*.

Actions Available at Resource Level

For an eXist database, the actions available at resource level in the **Data Source Explorer** view are as follows:

Refresh

Performs a refresh of the selected resource.

Open

Opens the selected resource in the editor.

Rename

Allows you to change the name of the selected resource.

Cut

Cuts the selected resources.

Copy

Copies the selected resources.

Note: You can add or move resources into the container by dragging them from the **Project** view, the default file management application (for example, Windows Explorer on Windows or Finder on OS X), or from another database container.

Paste

Pastes the copied resources.

Delete

Removes the selected resource from the collection.

Copy location

Allows you to copy (to the clipboard) an application-specific URL for the resource that can then be used for various actions, such as opening or transforming the resources.

Properties

Allows you to view various useful properties associated with the resource.

Save As

Allows you to save the name of the selected binary resource as a file on disk.

Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

MarkLogic Connection

Once you configure a MarkLogic connection, you can use the **Data Source Explorer** view to display all the application servers that are configured on the server. You can expand each application server and view all the modules that it is configured to use. The **Data Source Explorer** view allows you to open and edit these modules.

Note: To browse modules located in a database, directory properties must be associated with them. These directory properties are generated automatically if the *directory creation* property of the database is set to automatic. If this property is set to *manual* or *manual-enforced*, add the directory properties of the modules manually, using the XQuery function `xdmp:directory-create()`. For example, 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 Editor plugin displays both the queries that are stopped for debugging purposes and the queries that are still running. To clean up the entire **Requests** container at the end of your session, right-click it and use the **Cancel all running requests** action.

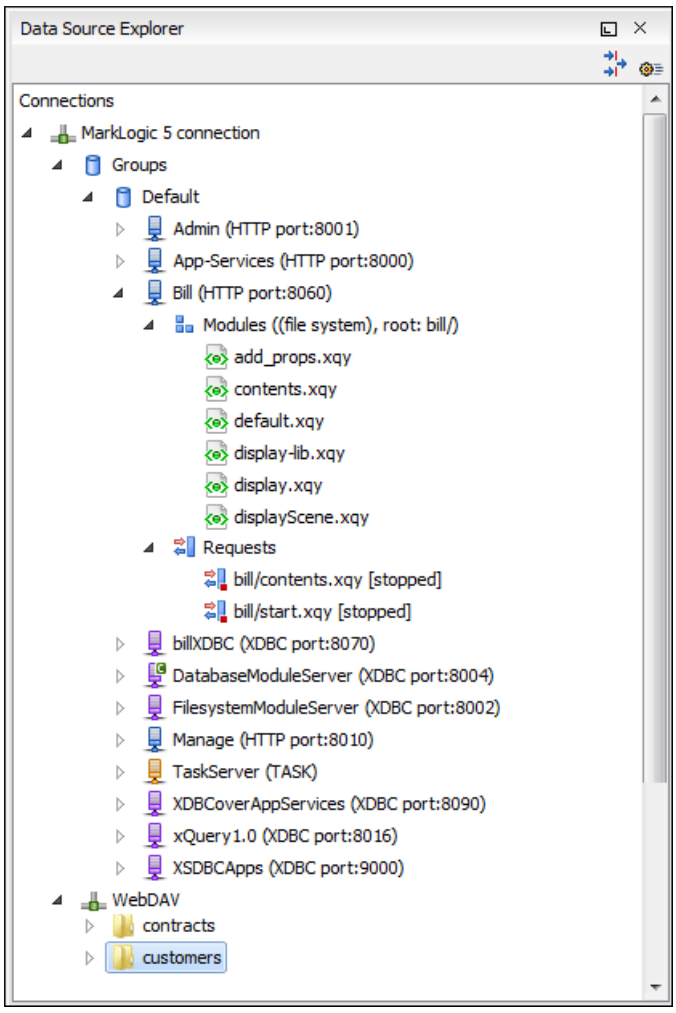


Figure 367: MarkLogic Connection in Data Source Explorer

The **Data Source Explorer** view displays all the application servers available on the MarkLogic server. To change the XDBC application server that Oxygen XML Editor plugin uses to process XQuery expressions, select the **Use it to execute queries** option from its contextual menu.

To manage resources for a MarkLogic database through WebDAV, configure a WebDAV URL in *the MarkLogic connection*.

The following actions are available in the contextual menu of the WebDAV connection:

- Connection level actions:

  Configure Database Sources

  Opens the **Data Sources** *preferences page*. Here you can configure both data sources and connections.

  New Folder

  Creates a new folder on the server.

  Import Files

  Allows you to add a new file on the server.

  Refresh

  Performs a refresh of the connection.

  Find/Replace in Files

  Allows you to find and replace text in multiple files from the server.

- Folder level actions:

  New File

  Creates a new file on the server in the current folder.

  New Folder

  Creates a new folder on the server.

  Import Folders

  Imports folders on the server.

   **Import Files**

  Allows you to add a new file on the server in the current folder.

   **Cut**

  Removes the current selection and places it in the clipboard.

   **Copy**

  Copies the current selection into the clipboard.

  Rename

  Allows you to change the name of the selected folder.

   **Delete**

  Removes the selected folder.

   **Refresh**

  Refreshes the sub-tree of the selected node.

   **Find/Replace in Files**

  Allows you to find and replace text in multiple files from the server.

- File level actions:

   **Open**

  Allows you to open the selected file in the editor.

   **Cut**

  Removes the current selection and places it in the clipboard.

   **Copy**

  Copies the current selection into the clipboard.

  Copy Location

  Copies an application-specific URL for the selected resource into the clipboard. You can use this URL for various actions, such as opening or transforming the resources.

  Rename

  Allows you to change the name of the selected file.

   **Delete**

  Removes the selected file.

   **Refresh**

  Performs a refresh of the selected node.

   **Properties**

  Displays the properties of the current file in a **Properties** dialog box.

   **Find/Replace in Files**

  Allows you to find and replace text in multiple files from the server.

Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

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 as follows:

 **Refresh**

Performs a refresh for the sub-tree of the selected node.

Disconnect

Closes the current database connection.

 **Configure Database Sources**

Opens the **Data Sources** *preferences page* where you can configure both data sources and connections.

Add library

Allows you to add a new library.

 **Insert XML Instance**

Allows you to add a new XML resource directly into the database root. See *Documentum xDb (X-Hive/DB) 10 Parser Configuration* for more details.

 **Insert non-XML Instance**

Allows you to add a new non-XML resource directly in the database root.

Properties

Displays the connection properties.

Actions Available at Catalog Level

For a Documentum xDb (X-Hive/DB) 10 database, the actions available at catalog level in the **Data Source Explorer** view are as follows:

 **Refresh**

Performs a refresh of the selected catalog.

Add as models

Allows you to add a new abstract schema model to the selected catalog.

Set default schema

Allows you to set a default DTD to be used for parsing. It is not possible to set a default XML Schema.

Clear default schema

Allows you to clear the default DTD. The action is available only if there is a DTD set as default.

Properties

Displays the catalog properties.

Actions Available at Schema Resource Level

For a Documentum xDb (X-Hive/DB) 10 database, the actions available at schema resource level in the **Data Source Explorer** view are as follows:

 **Refresh**

Performs a refresh of the selected schema resource.

 **Open**

Opens the selected schema resource in the editor.

Rename

Allows you to change the name of the selected schema resource.

Save As

Allows you to save the selected schema resource as a file on disk.

Delete

Removes the selected schema resource from the catalog.

Copy location

Allows you to copy the URL of the selected schema resource to the clipboard.

Set default schema

Allows you to set the selected DTD to be used as default for parsing. The action is available only for DTD.

Clear default schema

Allows you to unset the selected DTD. The action is available only if the selected DTD is the current default to be used for parsing.

Actions Available at Library Level

For a Documentum xDb (X-Hive/DB) 10 database, the actions available at library level in the **Data Source Explorer** view are as follows:

Refresh

Performs a refresh of the selected library.

Add library

Adds a new library as a child of the selected library.

Add local catalog

Adds a catalog to the selected library. By default, only the root-library has a catalog, and all models are stored there.

Insert XML Instance

Allows you to add a new XML resource to the selected library. See *Documentum xDb (X-Hive/DB) 10 Parser Configuration* for more details.

Insert non-XML Instance

Allows you to add a new non-XML resource to the selected library.

Rename

Allows you to specify a new name for the selected library.

Move

Allows you to move the selected library to a different one (also available through drag and drop).

Delete

Removes the selected library.

Properties

Displays the library properties.

Actions Available at Resource Level

When an XML instance document is added for a Documentum xDb (X-Hive/DB) 10 database, the actions available at resource level in the **Data Source Explorer** view are as follows:

Refresh

Performs a refresh of the selected resource.

Open

Opens the selected resource in the editor.

Rename

Allows you to change the name of the selected resource.

Move

Allows you to move the selected resource into a different library in the database tree (also available through drag and drop).

Note: You can copy or move resources by dragging them from another database catalog.

Save As

Allows you to save the selected binary resource as a file on disk.

Delete

Removes the selected resource from the library.

Copy location

Allows you to copy the URL of the selected resource to the clipboard.

Add AS model

Allows you to add an XML schema to the selected XML resource.

Set AS model

Allows you to set an active AS model for the selected XML resource.

Clear AS model

Allows you to clear the active AS model of the selected XML resource.

Properties

Displays the resource properties. Available only for XML resources.

Compare

Compares the resources using Diff Files (this action is available in the contextual menu of two selected resources).

Validation of an XML resource stored in 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, consult the Documentum xDb (X-Hive/DB) 10 manual):
  - **xhive-store-schema** - If checked, the corresponding DTD or XML schemas are stored in the catalog during validated parsing.
  - **xhive-store-schema-only-internal-subset** - Stores only the internal sub-set of the document (not an external sub-set). This option modifies the **xhive-store-schema** (only has a function when that parameter is set to true, and when a DTD is involved). Select this option if you only want to store the internal sub-set of the document (not the external sub-set).
  - **xhive-ignore-catalog** - Ignores the corresponding DTD and XML schemas in the catalog during validated parsing.
  - **xhive-psvi** - Stores **psvi** information about elements and attributes. Documents parsed with this feature turned on give access to **psvi** information and enable support of data types by XQuery queries.
  - **xhive-sync-features** - With this convenience setting turned on, parameter settings of `XhiveDocumentIf` are synchronized with the parameter settings of `LSParser`. Note that parameter settings **xhive-psvi** and **schema-location** are always synchronized.

Troubleshooting Documentum xDB

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
```

Since the driver was configured to use `xhive.jar` directly from the *xDB* installation (where many other jars are located), `core/xercesImpl.jar` from the *xDB* installation directory is loaded even though it is not specified in the list of jars from the data source driver configuration (it is in the classpath from `xhive.jar`'s *MANIFEST.MF*). A simple workaround for this issue is to copy **ONLY** the jar files used in the driver configuration to a separate folder and configure the data source driver to use them from there.

XQuery and Databases

XQuery is a native XML query language that is useful for querying XML views of relational data to create XML results. It also provides the mechanism to efficiently and easily extract information from Native XML Databases (NXD) and relational data. The following database systems supported in Oxygen XML Editor plugin offer XQuery support:

- *Native XML Databases:*
  - Berkeley DB XML
  - eXist
  - MarkLogic (validation support available starting with version 5)
  - Documentum xDb (X-Hive/DB) 10
- *Relational Databases:*
  - IBM DB2
  - Microsoft SQL Server (validation support not available)
  - Oracle (validation support not available)

Build Queries with Drag and Drop from the Data Source Explorer View

When a query is edited in the XQuery editor, the XPath expressions can be composed quickly by dragging them from the **Data Source Explorer** view and dropping them into the editor panel.

1. *Configure the data source* to the relational database.
2. *Configure the connection* to the relational database.
3. Browse the connection in the **Data Source Explorer** view, expanded to the table or column that you want to insert in the query.
4. Drag the table or column name to the XQuery editor panel.
5. Drop the table or column name where the XPath expression is needed.

An XPath expression that selects the dragged name is inserted in the XQuery document at the cursor position.

XQuery Transformation

XQuery is designed to retrieve and interpret XML data from any source, whether it is a database or document. Data is stored in relational databases but it is often required that the data be extracted and transformed as XML when interfacing to other components and services. Also, it is an XPath-based querying language supported by most NXD vendors. To perform a query, you need an XQuery transformation scenario.

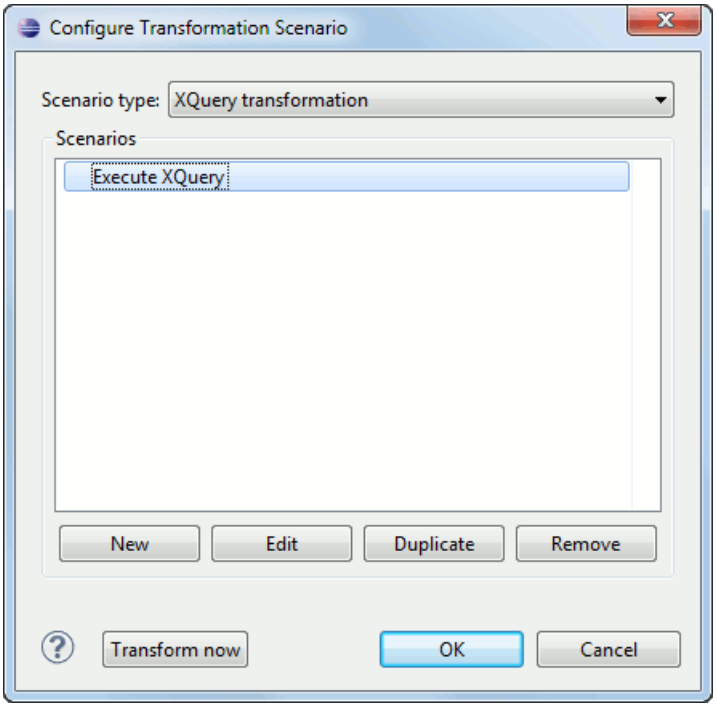
1. Configure a data source for the database.

   The data source can be *relational* or *XML native*.

2. Configure an XQuery transformation scenario.

   a) Click the **Configure Transformation Scenario** toolbar button or go to menu **Document** > **Transformation** > **Configure Transformation Scenario**.

   The **Configure Transformation Scenario** dialog box is opened.



   b) Click the **New** button in the dialog box.

   The dialog box for editing an XQuery scenario is opened.

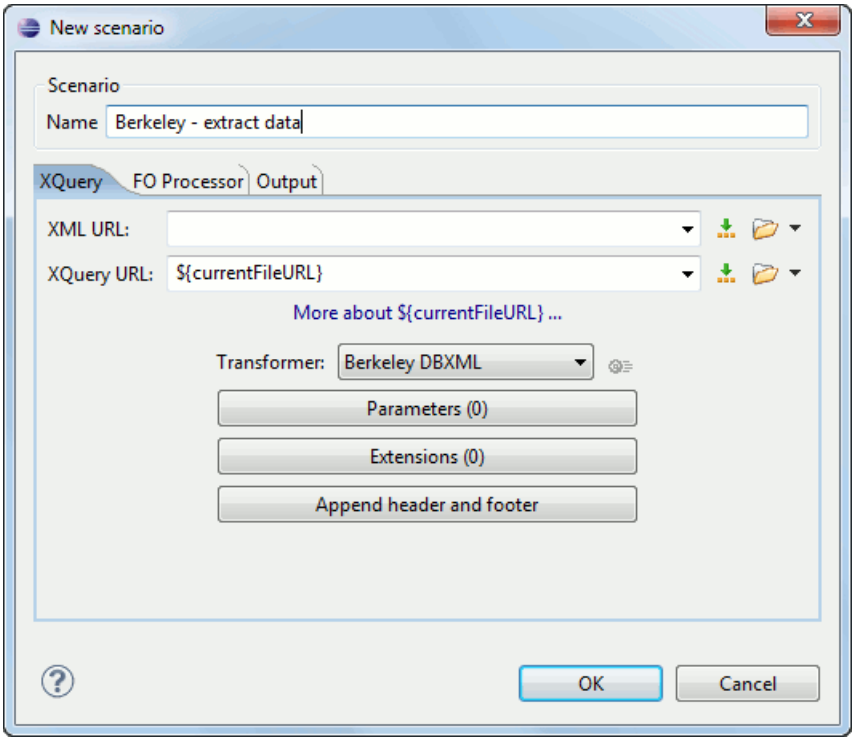


Figure 368: Edit Scenario Dialog Box

c) Insert the scenario name in the dialog box for editing the scenario.

d) Choose the database connection in the **Transformer** drop-down list.

e) Configure any other parameters as needed.

For an XQuery transformation, the output tab has an option called **Sequence** that allows you to run an XQuery in lazy mode. The amount of data extracted from the database is controlled from the option *Size limit on Sequence view*. If you choose **Perform FO Processing** in the **FO Processor** tab, the **Sequence** option is ignored.

f) Click the **OK** button to finish editing the scenario.

Once the scenario is associated with the XQuery file, the query can include calls to specific XQuery functions that are implemented by that engine. The available functions depend on the target database engine selected in the scenario. For example, for eXist and Berkeley DB XML, *the **Content Completion Assistant*** lists the functions supported by that database engine. This is useful for only inserting calls to the supported functions (standard XQuery functions or extension ones) into the query .

Note: An XQuery transformation is executed against a Berkeley DB XML server as a transaction using the query transaction support of the server.

3. Run the scenario.

To view a more complex value returned by the query that cannot be entirely displayed in the XQuery query result table at the bottom of the Oxygen XML Editor plugin window (for example, an XMLTYPE or CLOB value), do the following:

- Right-click that table cell.
- Select the **Copy cell** action from the contextual menu to copy the value into the clipboard.
- Paste the value wherever you need it (for example, in an opened XQuery editor panel of Oxygen XML Editor plugin).

XQuery Database Debugging

This section describes the procedures for debugging XQuery transformations that are executed against MarkLogic and Berkeley DB XML databases.

Debugging with MarkLogic

To start a debug session against the MarkLogic engine, configure a *MarkLogic data source* and a *MarkLogic connection*. Make sure that the debugging support is enabled in the MarkLogic server that Oxygen XML Editor plugin accesses. On the server side, debugging must be activated in the XDBC server and in the *Task Server* section of the server control console (the switch *debug allow*). If the debugging is not activated, the MarkLogic server reports the `DBG-TASKDEBUGALLOW` error.

The MarkLogic XQuery debugger integrates seamlessly into the *XQuery Debugger perspective*. If you have a MarkLogic scenario configured for the XQuery file, you can choose to *debug the scenario* directly. If not, switch to the XQuery Debugger perspective, open the XQuery file in the editor, and select the MarkLogic connection in the XQuery engine selector from the *debug control toolbar*. For general information about how a debugging session is started and controlled see the *Working with the Debugger* section.

If you want to debug an XQuery file stored on the MarkLogic server, we recommend you to use the **Data Source Explorer** view to open the module and start the debugging session. This improves the resolving of any imported modules.

Before starting a debugging session, we recommend that you link the MarkLogic connection with an Eclipse project. To do this, go to the **Data Source Explorer** view and select **Link to project** in the contextual menu of the MarkLogic connection. The major benefit of linking a debugging session with a project is that you can add breakpoints in the XQuery modules stored on the server. You are also able to access these modules from the Eclipse navigator and run debugging sessions from them.

Oxygen XML Editor plugin supports collaborative debugging. This feature allows multiple users to participate in the same debugging session. You can start a debugging session and at a certain point another user can continue it.

In a MarkLogic debugging session, when you add a breakpoint on a line where the debugger never stops, Oxygen XML Editor plugin displays a warning message. These warnings are displayed for breakpoints you add either in the main XQuery (which you can open locally or from the server), or for breakpoints you add in any XQuery that is opened from the connection that participates in the debugging session.

Remote Debugging with MarkLogic

The HTTP and the XDBC servers involved in the debugging session must have the same modules configuration.

To watch our video demonstration about the XQuery debugger for MarkLogic, go to *http://oxygenxml.com/demo/XQueryDebuggerforMarkLogic.html*.

Peculiarities and Limitations of the MarkLogic Debugger

MarkLogic debugger has the following peculiarities and limitations:

- Debugging support is available only for MarkLogic server versions 4.0 or newer.
- For MarkLogic server versions 4.0 or newer, there are three XQuery syntaxes that are supported: '0.9-ml' (inherited from MarkLogic 3.2), '1.0-ml', and '1.0'.
- The local debugger user interface presents all the debugging steps that the MarkLogic server executes and the results or possible errors of each step.
- All declared variables are presented as strings. The **Value** column of the **Variables** view contains the expression from the variable declaration. It can be evaluated by copying the expression with the **Copy value** action from the contextual menu of *the **Variables** view* and pasting it in *the **XWatch** view*.
- There is no support for *Output to Source Mapping*.
- There is no support for *showing the trace*.
- You can set *Breakpoints* in imported modules in one of the following cases:
  - When you open the module from the context of the application server involved in the debugging, using the **data source explorer**.

- When the debugger automatically opens the modules in the Editor.

- No breakpoints are set in modules from the same server that are not involved in the current debugging session.
- No support for *profiling* when an XQuery transformation is executed in the debugger.

Debugging Queries Which Import Modules

When debugging queries on a MarkLogic database that imports modules stored in the database, the recommended steps for placing a *breakpoint* in a module are as follows:

1. Start the debugging session with the action **Debug Scenario** from the **Transformation** toolbar or the **XQuery Debugger** toolbar button.
2. Click **Step into** repeatedly until reaching the desired module.
3. Add the module to the current *project* for easy access.
4. Set breakpoints in the module as needed.
5. *Continue debugging* the query.

When starting a new debugging session, make sure that the modules that you will debug are already opened in the editor. This is necessary so that the breakpoints in the modules will be considered. Also, make sure that there are no other opened modules that are not involved in the current debugging session.

Debugging with Berkeley DB XML

The Berkeley DB XML database added a debugging interface starting with version 2.5. The current version is supported in the Oxygen XML Editor plugin XQuery Debugger. *The same restrictions and peculiarities* apply for the Berkeley debugger as for the MarkLogic debugger.

WebDAV Connection

This section explains how to work with a WebDAV connection in the **Data Source Explorer** view.

How to Configure a WebDAV Connection

By default, Oxygen XML Editor plugin is configured to contain a WebDAV data source connection called **WebDAV (S)FTP**. Based on this data source, you can create a WebDAV connection for browsing and editing data from a database that provides a WebDAV interface. The connection is available in *the **Data Source Explorer** view*. The steps for configuring a WebDAV connection are as follows:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel, click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the WebDAV data sources in the **Data Source** drop-down menu.
5. Enter the connection details:
   a) Set the URL to the WebDAV repository in the field **WebDAV URL**.
   b) Set the user name that is used to access the WebDAV repository in the **User** field.
   c) Set the password that is used to access the WebDAV repository in the **Password** field.
6. Click the **OK** button.

   To watch our video demonstration about the WebDAV support in Oxygen XML Editor plugin, go to *http://www.oxygenxml.com/demo/WebDAV_Support.html*.

WebDAV Connection Actions

This section explains the actions that are available for a WebDAV connection in the **Data Source Explorer** view.

Actions Available at Connection Level

The contextual menu of a WebDAV connection in the **Data Source Explorer** view contains the following actions:

Configure Database Sources

Opens the **Data Sources** *preferences page*. Here you can configure both data sources and connections.

Disconnect

Stops the connection.

Import Files

Allows you to add a new file on the server.

New Folder

Creates a new folder on the server.

Refresh

Performs a refresh of the connection.

Find/Replace in Files

Allows you to find and replace text in multiple files from the server.

Actions Available at Folder Level

The contextual menu of a folder node in a WebDAV connection in the **Data Source Explorer** view contains the following actions:

New File

Creates a new file on the server in the current folder.

New Folder

Creates a new folder on the server.

Import Folders

Imports folders on the server.

Import Files

Allows you to add a new file on the server in the current folder.

Cut

Removes the current selection and places it in the clipboard.

Copy

Copies the current selection into the clipboard.

Paste

Pastes the copied selection.

Rename

Allows you to change the name of the selected folder.

Delete

Removes the selected folder.

Refresh

Refreshes the sub-tree of the selected node.

Find/Replace in Files

Allows you to find and replace text in multiple files from the server.

Actions Available at File Level

The contextual menu of a file node in a WebDAV connection in the **Data Source Explorer** view contains the following actions:

Open

Allows you to open the selected file in the editor.

Cut

Removes the current selection and places it in the clipboard.

Copy

Copies the current selection into the clipboard.

Copy Location

Copies an application-specific URL for the selected resource into the clipboard. You can use this URL for various actions, such as opening or transforming the resources.

Rename

Allows you to change the name of the selected file.

Delete

Removes the selected file.

Refresh

Performs a refresh of the selected node.

Properties

Displays the properties of the current file in a **Properties** dialog box.

Find/Replace in Files

Allows you to find and replace text in multiple files from the server.

BaseX Support

This section explains how to configure the BaseX XML database support. The BaseX support is composed of two parts:

- Resource management in the **Data Source Explorer** view.
- XQuery execution.

Resource Management

Resource management is available by creating a WebDAV connection to the BaseX server.

First of all, make sure the BaseX HTTP Server is started. For details about starting the BaseX HTTP server, go to *http://docs.basex.org/wiki/Startup#BaseX_HTTP_Server*. The configuration file for the HTTP server is named `.basex` and is located in the BaseX installation directory. This file helps you to find out the port on which the HTTP server is running. The default port for BaseX WebDAV is 8984.

To ensure that everything is functioning, open a WebDAV URL inside a browser and check to see if it works. For example, the following URL retrieves a document from a database named TEST:
`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 Editor plugin as described in *How to Configure a WebDAV Connection* on page 873. 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 798. The BaseX XQJ API-specific files that must be added in the configuration dialog box are `xqj-api-1.0.jar`, `xqj2-0.1.0.jar` and `basex-xqj-1.2.3.jar` (the version names of the JAR file may differ). These libraries can be downloaded from *`xqj.net/basex/basex-xqj-1.2.3.zip`*. As an alternative, you can also find the libraries in the BaseX installation directory, in the **lib** sub-directory.

BaseX XQJ Connection

The next step is to create an XQJ connection as described in *How to Configure an XQJ Connection* on page 799.

For a default BaseX configuration, the following connection details apply (you can modify them when necessary):

- *Port*: 1984
- *serverName*: localhost
- *user*: admin
- *password*: admin

XQuery Execution

Now that the XQJ connection is configured, open the XQuery file you want to execute in Oxygen XML Editor plugin and create a *Transformation Scenario* as described in *XQuery Transformation* on page 742. In the **Transformer** drop-down menu, select the name of the XQJ connection you created. Apply the transformation scenario and the XQuery will be executed.

Chapter 16

Importing Data

Topics:

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.

Oxygen XML Editor plugin offers support for importing text files, MS Excel files, Database Data, and HTML files into XML documents. The XML documents can be further converted into other formats using the *Transform features*.

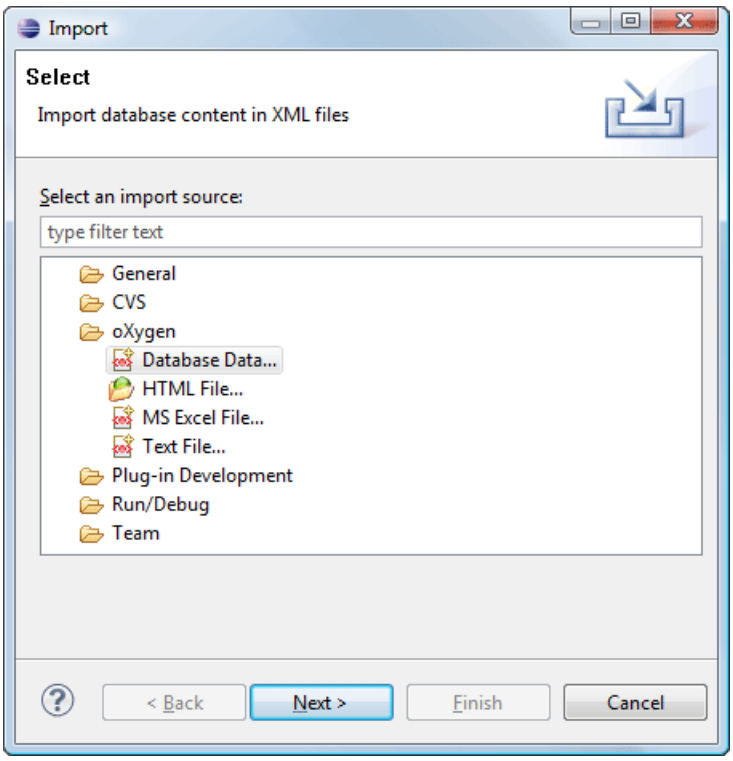


Figure 369: The Import Wizards of the Oxygen XML Editor plugin Plugin

Import from Text Files

To import a text file into an XML file, follow these steps:

1. Go to **File** > **Import** > **Oxygen XML Editor plugin** > **Text File** and click **Next**.
   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 **Next** button.
   The **Import from text file** dialog box is displayed.

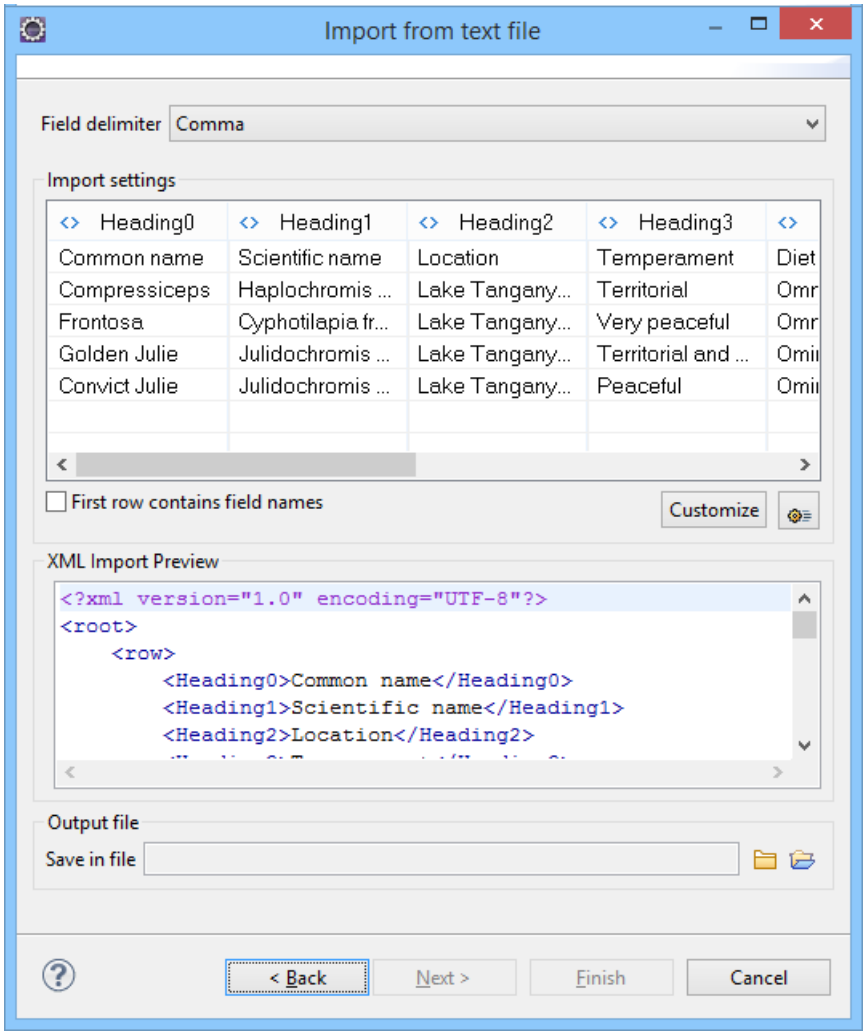


Figure 370: Import from Text File Dialog Box

5. Configure the settings for the conversion.
   a) Select the **Field delimiter** for the import settings. You can choose between the following: `Comma`, `Semicolon`, `Tab`, `Space`, or `Pipe`.
   b) The **Import settings** section presents the input data in a tabular form. By default, all data items are converted to element content (<> symbol), but this can be overridden by clicking the individual column headers. Clicking a column header once causes the data from this column to be converted to attribute values (= symbol). Clicking a second time causes the column data to be ignored (× symbol) when generating the XML file. You can cycle through these three options by continuing to click the column header.

c) **First row contains field names** - If this option is enabled, 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 panel.

d) **Customize** - This button opens a **Presentation Names** dialog box that allows you to edit the name, XML name, and conversion criterion for the root and row elements. The XML names can be edited by double-clicking the desired item and entering the label. The conversion criteria can also be modified by selecting one of the following option in the drop-down menu: `ELEMENT`, `ATTRIBUTE`, or `SKIPPED`.

e) **Import Settings** - Clicking this button opens the *Import Preferences* on page 975 page that allows you to configure more import options.

f) The **XML Import Preview** panel contains an example of what the generated XML document looks like.

g) **Save in file** - If checked, the new XML document is saved in the specified path.

6. Click **Finish** to generate the XML document.

Import from MS Excel Files

By default, importing Excel 97/2000/XP/2003 formats are supported out-of-the-box. To import spreadsheet data from Excel 2007 or newer, additional libraries are needed before using this procedure. See *Import Data from MS Excel 2007 or Newer* on page 881 for instructions on adding the additional libraries.

Oxygen XML Editor plugin offers support for importing MS Excel Files.

To import an Excel file into an XML file, follow these steps:

1. Go to **File** > **Import** > **Oxygen XML Editor plugin** > **MS Excel file**.
2. Select the URL of the Excel file.

   The sheets of the document you are importing are presented in the **Available Sheets** section of this dialog box.
3. Click the **Next** button.
   Opens the **Import from Excel** dialog box.

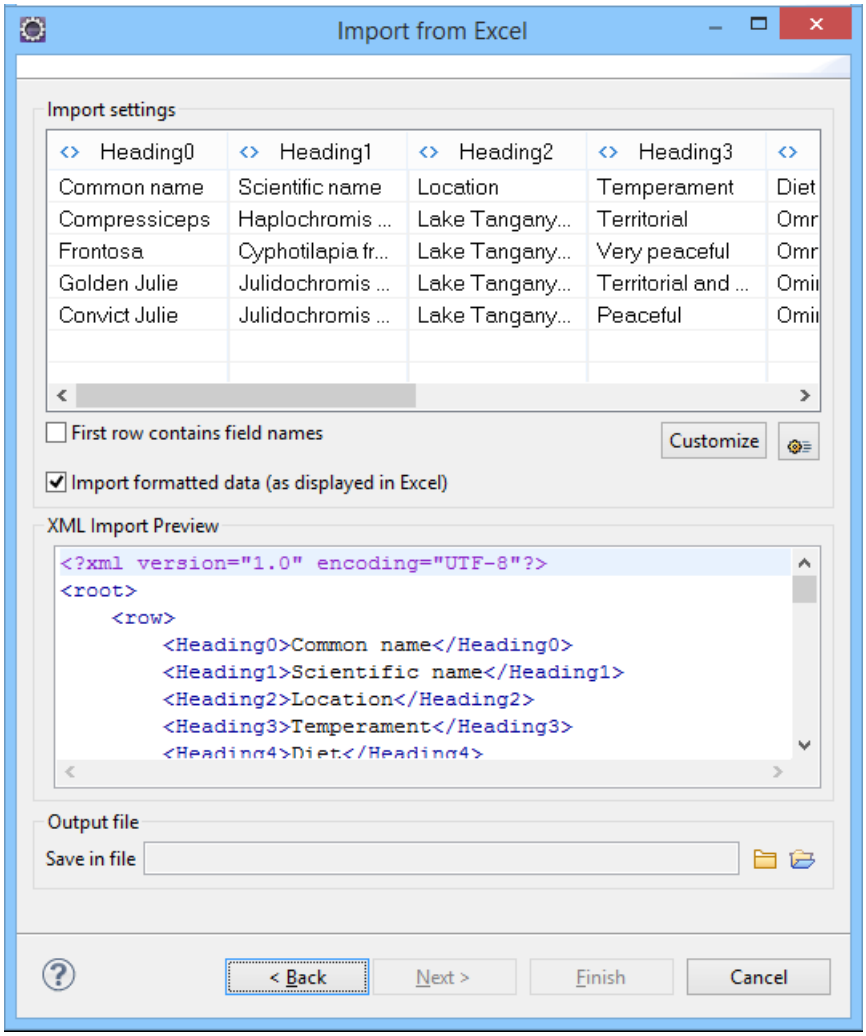


Figure 371: Import from Excel Dialog Box

4. Configure the settings for the conversion.
   a) The **Import settings** section presents the input data in a tabular form. By default, all data items are converted to element content (‹› symbol), but this can be overridden by clicking the individual column headers. Clicking a column header once causes the data from this column to be converted to attribute values (= symbol). Clicking a second time causes the column data to be ignored (× symbol) when generating the XML file. You can cycle through these three options by continuing to click the column header.
   b) **First row contains field names** - If this option is enabled, 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 panel.
   c) **Customize** - This button opens a **Presentation Names** dialog box that allows you to edit the name, XML name, and conversion criterion for the root and row elements. The XML names can be edited by double-clicking the desired item and entering the label. The conversion criteria can also be modified by selecting one of the following option in the drop-down menu: `ELEMENT`, `ATTRIBUTE`, or `SKIPPED`.
   d) **Import Settings** - Clicking this button opens the *Import Preferences* on page 975 page that allows you to configure more import options.
   e) **Import formatted data (as displayed in Excel)** - If this option is selected, the imported data retains the Excel styling. If deselected, the data formatting is not imported.
   f) The **XML Import Preview** panel contains an example of what the generated XML document looks like.
   g) **Save in file** - If checked, the new XML document is saved in the specified path.

5. Click **Finish** to generate the XML document.

Import Data from MS Excel 2007 or Newer

To import spreadsheet data from Excel 2007 or newer (`.xlsx`), Oxygen XML Editor plugin needs additional libraries from the release 3.10 of the Apache POI project.

To add the additional libraries, follow these steps:

1. Download version 3.10 of the Apache POI project from *http://archive.apache.org/dist/poi/release/bin/*. The specific ZIP file that you need is: `poi-bin-3.10-FINAL-20140208.zip`.
2. Unpack `poi-bin-3.10-FINAL-20140208.zip`.
3. Copy the following `.jar` files in the `plugin.xml` file of the Oxygen XML Editor plugin Eclipse plugin (if you installed the plugin via the Eclipse update site, you will find it in the `eclipse/plugins/com.oxygenxml...` folder, and if you installed it via the dropins ZIP distribution, it is located in the `eclipse/dropins/plugins/com.oxygenxml...` folder):
   - `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 Database Data as an XML Document

To import the data from a relational database table as an XML document, follow these steps:

1. Go to **File** > **Import** > **oXygen / Database Data** and click **Next** to start the **Import** wizard.

   This opens a **Select database table** dialog box that lists all the defined database connections:

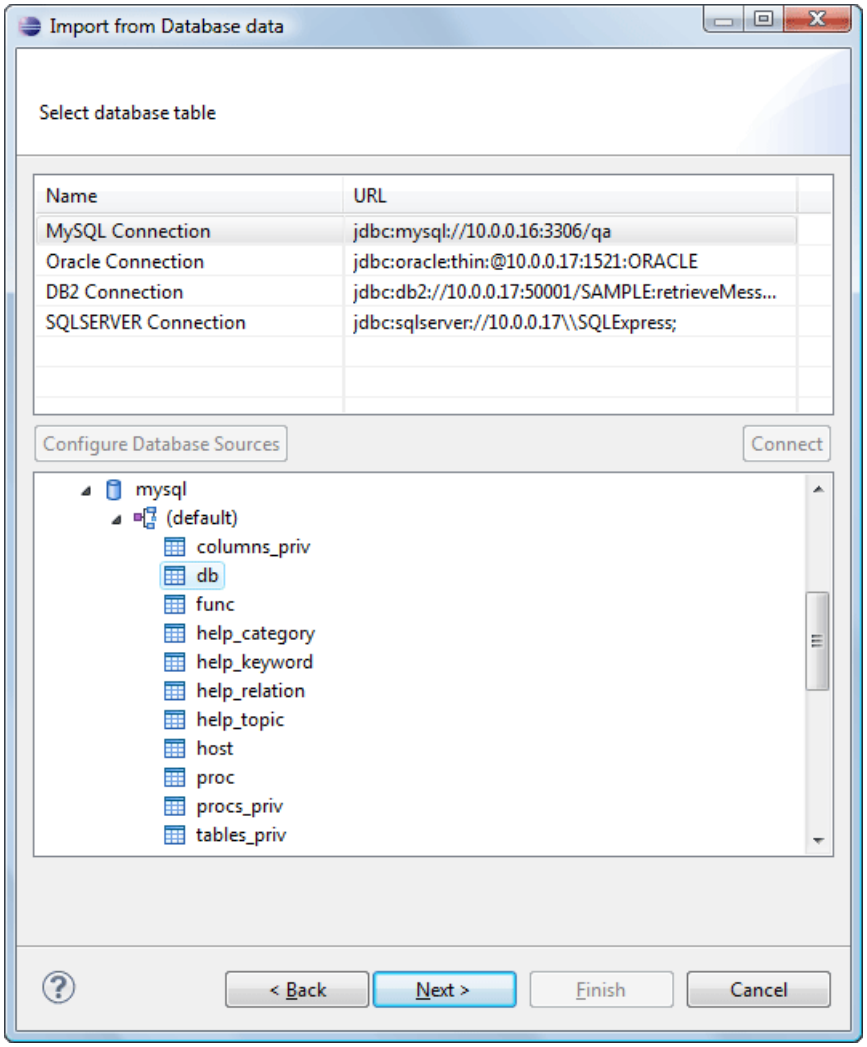


Figure 372: Select Database Table Dialog Box

2. Select the connection to the database that contains the appropriate data.

   Only connections configured in relational data sources can be used to import data.
3. If you want to edit, delete, or add a data source or connection, click the **Configure Database Sources** button. The **Preferences**/**Data Sources** option page is opened.
4. Click **Connect**.
5. In the list of sources, expand a schema and choose the required table.
6. Click the **Next** button.

   The **Import Criteria** dialog box is opened with a default query string in the **SQL Query** pane.

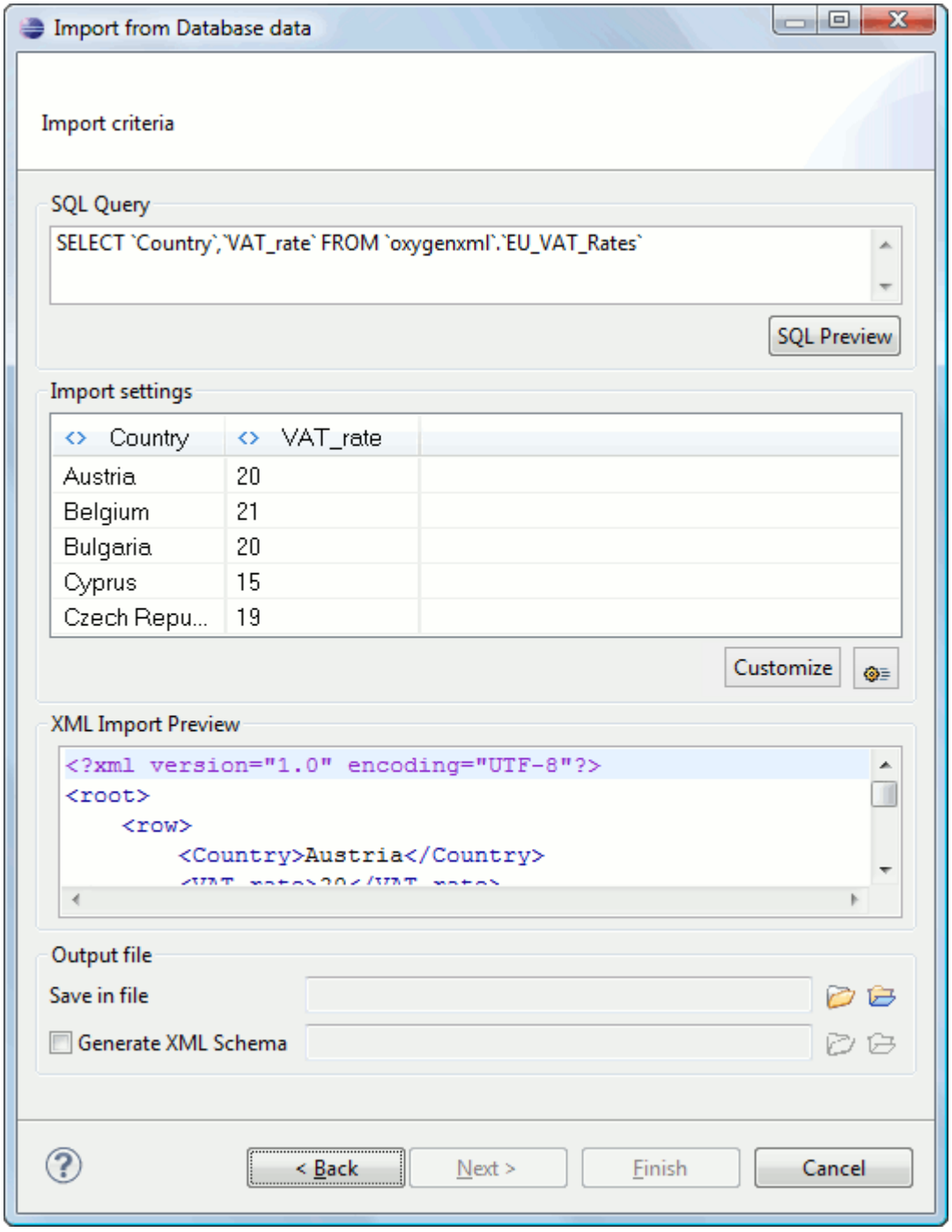


Figure 373: Import from Database Criteria Dialog Box

7. Configure the settings for the conversion.
   a) **SQL Preview** - If this button is pressed, the **Import settings** pane displays the labels that are used in the XML document and the first five lines from the database. By default, all data items are converted to element content (‹› symbol), but this can be overridden by clicking the individual column headers. Clicking a column header once causes the data from this column to be converted to attribute values (= symbol). Clicking a second time causes the column data to be ignored (× symbol) when generating the XML file. You can cycle through these three options by continuing to click the column header.
   b) **Customize** - This button opens a **Presentation Names** dialog box that allows you to edit the name, XML name, and conversion criterion for the root and row elements. The XML names can be edited by double-clicking the

desired item and entering the label. The conversion criteria can also be modified by selecting one of the following option in the drop-down menu: `ELEMENT`, `ATTRIBUTE`, or `SKIPPED`.

c) **Import Settings** - Clicking this button opens the *Import Preferences* on page 975 page that allows you to configure more import options.

d) The **XML Import Preview** panel contains an example of what the generated XML document looks like.

e) **Save in file** - If checked, the new XML document is saved in the specified path.

f) **Generate XML Schema** - Allows you to specify the path of the generated XML Schema file.

8. Click **Finish** to generate the XML document.

Import from HTML Files

Oxygen XML Editor plugin offers support for importing HTML files as an XML document.

To import from HTML files, follow these steps:

1. Go to **File** > **Import** > **Oxygen XML Editor plugin** > **HTML File**.
   The **Import HTML** wizard is displayed.
2. Select a parent folder and file name for the resulting XHTML document.
3. Enter the URL of the HTML document.
4. Select the type of the resulting XHTML document:
   - XHTML 1.0 Transitional
   - XHTML 1.0 Strict
5. Click the **Finish** button.

The resulting document is an XHTML file containing a DOCTYPE declaration that references the XHTML DTD definition on the Web. The parsed content of the imported file is transformed to XHTML Transitional or XHTML Strict depending on the option you chose when performing the import operation.

Import Content Dynamically

Along with the built-in support for various useful URL protocols (such as HTTP or FTP), Oxygen XML Editor plugin also provides special support for a *convert* protocol that can be used to chain predefined processors to import content from various sources dynamically.

A dynamic conversion URL chains various processors that can be applied in sequence on a target resource and has the following general syntax:

```
convert:/processor=xslt;ss=urn:processors:excel2d.xsl/processor=excel!/urn:files:sample.xls
```

The previous example first applies a processor called `excel` on a target identified by the identifier `urn:files:sample.xls` and converts the Excel™ resource to XML. The second applied processor (`xslt`) applies an XSLT stylesheet identified using the identifier `urn:processors:excel2d.xsl` over the content resulting from the first applied processor. These identifiers are all mapped to real resources on disk via an *XML catalog* that is configured in the application, as in the following example:

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
 <rewriteURI uriStartString="urn:files:" rewritePrefix="./resources/"/>
 <rewriteURI uriStartString="urn:processors:" rewritePrefix="./processors/"/>
</catalog>
```

This type of URL can be opened in the application by using the **Open URL** action from the **File** menu. It can also be referenced from existing XML resources via *xi:include* or from *DITA maps* as topic references.

A *GitHub* project that contains various dynamic conversion samples for producing DITA content from various sources (and then publishing it) can be found here: *https://github.com/oxygenxml/dita-glass*.

Conversion Processors

A set of predefined conversion processors is provided in Oxygen XML Editor plugin out-of-the box. Each processor has its own parameters that can be set to control the behavior of the conversion process. All parameters that are resolved to resources are passed through the XML catalog mapping.

The following predefined conversion processors are included:

- **`xslt` Processor** - Converts an XML input using XSLT 2.0 processing. The `ss` parameter indicates the stylesheet resource to be loaded. All other specified parameters will be set as parameters to the XSLT transformation.

```
convert:/processor=xslt;ss=urn:processors:convert.xsl;p1=v1!/urn:files:sample.xml
```

- **`xquery` Processor** - Converts an XML input using XQuery processing. The `ss` parameter indicates the XQuery script to be loaded. All other specified parameters will be set as parameters to the XSLT transformation.

```
convert:/processor=xquery;ss=urn:processors:convert.xquery;p1=v1!/urn:files:sample.xml
```

- **`excel` Processor** - Converts an Excel™ input to an XML format that can be later converted by other piped processors. It has a single parameter `sn`, which indicates the name of the sheet that needs to be converted. If this parameter is missing, the XML will contain the combined content of all sheets included in the Excel™ document.

```
convert:/processor=excel;sn=test!/urn:files:sample.xls
```

- **`java` Processor** - Converts an input to another format by applying a specific Java method. The `jars` parameter is a comma separated list of JAR libraries or folders, from which libraries will be loaded. The `ccn` parameter is the fully qualified name of the conversion class that will be instantiated. The conversion class needs to have a method with the following signature:

```
  public void convert(String systemID, String originalSourceSystemID, InputStream is,
OutputStream os, LinkedHashMap<String, String> properties) throws IOException
```

```
convert:/processor=java;jars=libs;ccn=test.JavaToXML!/
urn:files:java/WSEditorBase.java
```

- **`js` Processor** - Converts an input to another format by applying a JavaScript method. The `js` parameter indicates the script that will be used. The `fn` parameter is the name of the method that will be called from the script. The method must take a string as an argument and return a string. If any of the parameters are missing, an error is thrown and the conversion stops.

```
convert:/processor=js;js=urn:processors:md.js;fn=convertExternal!/urn:files:sample.md
```

- **`json` Processor** - Converts a JSON input to XML. It has no parameters.

```
convert:/processor=json!/urn:files:personal.json
```

- **`xhtml` Processor** - Converts HTML content to well-formed XHTML. It has no parameters.

```
convert:/processor=xhtml!/urn:files:test.html
```

- **`wrap` Processor** - Wraps content in a tag name making it well-formed XML. The `rn` parameter indicates the name of the root tag to use. By default, it is `wrapper`. The `encoding` parameter specifies the encoding that should be used to read the content. By default, it is `UTF8`. As an example, this processor can be used if you want to process a comma-separated values file with an XSLT stylesheet to produce XML content. The CSV file is first wrapped as well-formed XML, which is then processed with an `xslt` processor.

```
convert:/processor=wrap!/urn:files:test.csv
```

Reverse Conversion Processors

All processors defined above can also be used for saving content back to the target resource if they are defined in the URL as reverse processors. Reverse processors are evaluated right to left. These reverse processors allow *round-tripping* content to and from the target resource.

As an example, the following URL converts HTML to DITA when the URL is opened using the `h2d.xsl` stylesheet and converts DITA to HTML when the content is saved in the application using the `d2h.xsl` stylesheet.

```
convert:/processor=xslt;ss=h2d.xsl/rprocessor=xslt;ss=d2h.xsl!/urn:files:sample.html
```

Important: If you are publishing a DITA map that has such conversion URL references inside, you need to edit the transformation scenario and set the value of the parameter *fix.external.refs.com.oxygenxml* to *true*. This will instruct Oxygen XML Editor plugin to resolve such references during a special pre-processing stage. Depending on the conversion, you may also require additional libraries to be added using the **Libaries** button in the **Advanced** tab of the transformation scenario.

Chapter 17

Content Management System (CMS) Integration

Topics:

This chapter explains how you can use Oxygen XML Editor plugin with Documentum CMS and Microsoft SharePoint. .

All the major CMS vendors that are listed in the *CMS Solution Partners section of our website* also provide CMS integration with Oxygen XML Editor plugin.

Integration with Documentum (CMS) (deprecated)

Important: Starting with version 17.0, the support for Documentum (CMS) is deprecated and will no longer be actively maintained.

Oxygen XML Editor plugin provides support for browsing and managing Documentum repositories in the Data Source Explorer. You can easily create new resources on the repository, copy and move them using contextual actions or the drag and drop support, edit and transform the documents in the editor. The operations that can be performed on repository resources are described in the *Documentum (CMS) actions* section.

Oxygen XML Editor plugin supports Documentum (CMS) version 6.5 and 6.6 with *Documentum Foundation Services 6.5 or 6.6* installed.

Attention:

It is recommended to use the latest 1.6.x Java version. It is possible that the Documentum (CMS) support will not work properly if you use other Java versions.

Configure Connection to Documentum Server

This section explains how to configure a connection to a Documentum server.

How to Configure a Documentum (CMS) Data Source

Available in the Enterprise edition only.

To configure a Documentum (CMS) data source you need the Documentum Foundation Services Software Development Kit (*DFS SDK*) corresponding to your server version. The *DFS SDK* can be found in the Documentum (CMS) server installation kit or it can be downloaded from *EMC Community Network*.

Note: The *DFS SDK* can be found in the form of an archive named, for example, *emc-dfs-sdk-6.5.zip* for Documentum (CMS) 6.5.

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Data Sources** panel click the **New** button.
3. Enter a unique name for the data source.
4. Select **Documentum (CMS)** from the driver type combo box.
5. Press the **Choose DFS SDK Folder** button.
6. Select the folder where you have unpacked the *DFS SDK* archive file.

   If you have indicated the correct folder the following Java libraries (jar files) will be added to the list (some variation of the library names is possible in future versions of the *DFS SDK*):

   - `lib/java/emc-bpm-services-remote.jar`
   - `lib/java/emc-ci-services-remote.jar`
   - `lib/java/emc-collaboration-services-remote.jar`
   - `lib/java/emc-dfs-rt-remote.jar`
   - `lib/java/emc-dfs-services-remote.jar`
   - `lib/java/emc-dfs-tools.jar`
   - `lib/java/emc-search-services-remote.jar`
   - `lib/java/ucf/client/ucf-installer.jar`
   - `lib/java/commons/*.jar` (multiple jar files)
   - `lib/java/jaxws/*.jar` (multiple jar files)
   - `lib/java/utils/*.jar` (multiple jar files)

   Note: If for some reason the jar files are not found, you can add them manually by using the **Add Files** and **Add Recursively** buttons and navigating to the `lib/java` folder from the *DFS SDK*.

7. Click the **OK** button to finish the data source configuration.

How to Configure a Documentum (CMS) Connection

Available in the Enterprise edition only.

The steps for configuring a connection to a Documentum (CMS) server are the following:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select one of the previously configured Documentum (CMS) data sources in the **Data Source** combo box.
5. Fill-in the connection details:
   - **URL** - The URL to the Documentum (CMS) server: `http://<hostname>:<port>`
   - **User** - The user name to access the Documentum (CMS) repository.
   - **Password** - The password to access the Documentum (CMS) repository.
   - **Repository** - The name of the repository to log into.
6. Click the **OK** button to finish the configuration of the connection.

Known Issues

The following are known issues with the Documentum (CMS):

1. Note that there is a known problem in the UCF Client implementation for Mac OS X from Documentum 6.5 which prevents you from viewing or editing XML documents from the repository on Mac OS X. The UCF Client is the component responsible for file transfer between the repository and the local machine. This component is deployed automatically from the server. Documentum 6.6 does not exhibit this problem.

   Note: This issue was reproduced with Documentum 6.5 SP1. In Documentum 6.6 this is no longer reproducing.

2. In order for the Documentum driver to work faster on Linux, you need to specify to the JVM to use a weaker random generator, instead of the very slow native implementation. This can be done by modifying in the Oxygen XML Editor plugin startup scripts (or in the `*.vmoptions` file) the system property:

```
-Djava.security.egd=file:/dev/./urandom
```

Documentum (CMS) Actions in the Data Source Explorer View

Oxygen XML Editor plugin allows you to browse the structure of a Documentum repository in the **Data Source Explorer** view and perform various operations on the repository resources.

You can drag and drop folders and resources to other folders to perform move or copy operations with ease. If the drag and drop is between resources (drag the child item to the parent item) you can create a relationship between the respective resources.

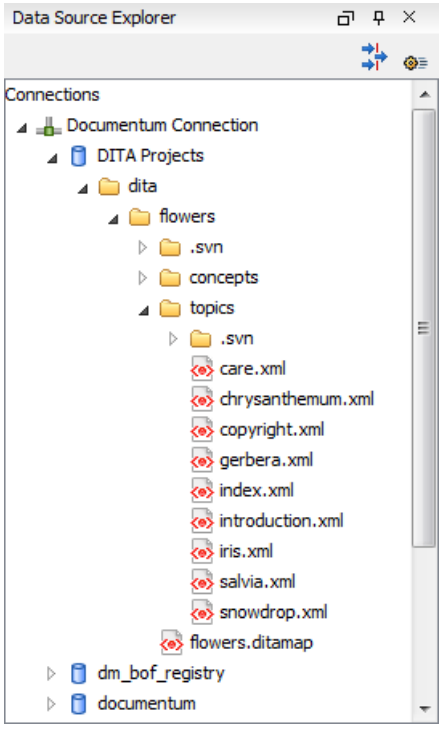


Figure 374: Browsing a Documentum repository

Actions Available on Connection

The contextual menu of a Documentum (CMS) connection in the **Data Source Explorer** view offers the following actions:

Configure Database Sources

Opens the *Data Sources* *preferences page* where you can configure both data sources and connections.

New Cabinet

Creates a new cabinet in the repository. The cabinet properties are:

- **Type** - The type of the new cabinet (default is **dm_cabinet**).
- **Name** - The name of the new cabinet.
- **Title** - The title property of the cabinet.
- **Subject** - The subject property of the cabinet.

Refresh

Refreshes the connection.

Actions Available on Cabinets / Folders

The actions available on a Documentum (CMS) cabinet in the **Data Source Explorer** view are the following:

New Folder

Creates a new folder in the current cabinet / folder. The folder properties are the following:

- **Path** - Shows the path where the new folder will be created.
- **Type** - The type of the new folder (default is **dm_folder**).
- **Name** - The name of the new folder.
- **Title** - The title property of the folder.
- **Subject** - The subject property of the folder.

New Document

Creates a new document in the current cabinet / folder. The document properties are the following:

- **Path** - Shows the path where the new document will be created.
- **Name** - The name of the new document.
- **Type** - The type of the new document (default is **dm_document**).
- **Format** - The document content type format.

Import

Imports local files / folders in the selected cabinet / folder of the repository. Actions available when performing an import:

- **Add Files** - Opens a file browse dialog box and allows you to select files to add to the list.
- **Add Folders** - Opens a folder browse dialog box that allows you to select folders to add to the list. The subfolders will be added recursively.
- **Edit** - Opens a dialog box where you can change the properties of the selected file / folder from the list.
- **Remove** - Removes the selected files / folders from the list.

Rename

Changes the name of the selected cabinet / folder.

Copy

Copies the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop while holding the **(Ctrl (Meta on Mac OS))** key pressed.

Move

Moves the selected folder to a different location in the tree (available only upon folders). This action can also be performed with drag and drop.

Delete

Deletes the selected cabinet / folder from the repository. The following options are available:

- **Folder(s)** - Allows you to delete only the selected folder or to delete recursively the folder and all subfolders and objects.
- **Version(s)** - Allows you to specify what versions of the resources will be deleted.
- **Virtual document(s)** - Here you can specify what happens when virtual documents are encountered. They can be either deleted either by themselves or together with their descendants.

Refresh

Performs a refresh of the selected node's sub-tree.

Properties

Displays the list of properties of the selected cabinet / folder.

Actions Available on Resources

The actions available on a Documentum (CMS) resource in the **Data Source Explorer** view are the following:

Edit

Checks out (if not already checked out) and opens the selected resource in the editor.

Edit with

Checks out (if not already checked out) and opens the selected resource in the specified editor / tool.

Open (Read-only)

Opens the selected resource in the editor.

Open with

Opens the selected resource in the specified editor / tool.

Check Out

Checks out the selected resource from the repository. The action is not available if the resource is already checked out.

Check In

Checks in the selected resource (commits changes) into the repository. The action is only available if the resource is checked out.

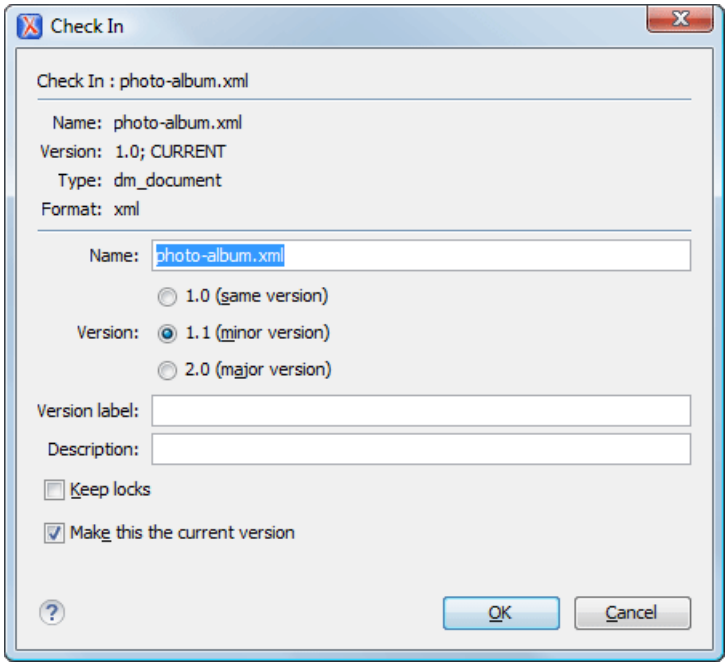


Figure 375: Check In Dialog Box

The following resource properties are available:

- **Name** - The resource name in the repository.
- **Version** - Allows you to choose what version the resource will have after being checked in.
- **Version label** - The label of the updated version.
- **Description** - An optional description of the resource.
- **Keep Locks** - When this option is enabled, the updated resource is checked into the repository but it also keeps it locked.
- **Make this the current version** - Makes the updated resource the current version (will have the *CURRENT* version label).

Cancel Checkout

Cancels the checkout process and loses all modifications since the checkout. Action is only available if the resource is checked out.

Export

Allows you to export the resource and save it locally.

Rename

Changes the name of the selected resource.

Copy

Copies the selected resource in a different location in the tree. Action is not available on virtual document descendants. This action can also be performed with drag and drop while holding the **Ctrl (Meta on OS X)** key pressed.

Move

Moves the selected resource in a different location in the tree. Action is not available on virtual document descendants and on checked out resources. This action can also be performed with drag and drop.

Delete

Deletes the selected resource from the repository. Action is not available on virtual document descendants and on checked out resources.

Add Relationship

Adds a new relationship for the selected resource. This action can also be performed with drag and drop between resources.

Convert to Virtual Document

Allows you to convert a simple document to a virtual document. Action is available only if the resource is a simple document.

Convert to Simple Document

Allows you to convert a virtual document to a simple document. Action is available only if the resource is a virtual document with no descendants.

Copy location

Allows you to copy to clipboard an application-specific URL for the resource which can then be used for various actions like opening or transforming the resources.

Refresh

Performs a refresh of the selected resource.

Properties

Displays the list of properties of the selected resource.

Transformations on DITA Content from Documentum (CMS)

Oxygen XML Editor plugin comes with the DITA Open Toolkit which is able to transform a DITA map to various output formats. However DITA Open Toolkit requires local DITA files so first you need to check out a local version of your DITA content. Once you have a local version of a DITA map just load it in *the **DITA Maps Manager** view* and run one of the DITA transformations that are predefined in Oxygen XML Editor plugin or a customization of such a predefined DITA transformation.

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 sites that use as authentication method either *Cloud identity (default)* or *Federated identity (ADFS)*.

To watch our video demonstration about connecting to a repository located on a SharePoint server and using it, go to *http://www.oxygenxml.com/demo/SharePoint_Support.html*.

How to Configure a SharePoint Connection

By default Oxygen XML Editor plugin contains a predefined **SharePoint** data source. Use this data source to create a connection to a SharePoint server which will be available in *the **Data Source Explorer** view*.

Follow these steps to configure a SharePoint connection:

1. *Open the **Preferences** dialog box* and go to **Data Sources**.
2. In the **Connections** panel click the **New** button.
3. Enter a unique name for the connection.
4. Select SharePoint in the **Data Source** combo box.
5. Fill-in the connection details:

a) Set the URL to the SharePoint repository in the field **SharePoint URL**.
b) Set the server domain in the **Domain** field.
c) Set the user name to access the SharePoint repository in the **User** field.
d) Set the password to access the SharePoint repository in the **Password** field.

To watch our video demonstration about connecting to repository located on a SharePoint server, go to *http://www.oxygenxml.com/demo/SharePoint_Support.html*.

SharePoint Connection Actions

This section explains the actions that are available on a SharePoint connection in the **Data Source Explorer** view.

Actions Available at Connection Level

The contextual menu of a SharePoint connection in the **Data Source Explorer** view contains the following actions:


Configure Database Sources

Opens the **Data Sources** *preferences page*. Here you can configure both data sources and connections.

Disconnect

Stops the connection.

New Folder

Creates a new folder on the server.


Import Files

Allows you to add a new file on the server.


Refresh

Performs a refresh of the connection.


Find/Replace in Files

Allows you to find and replace text in multiple files from the server.

Actions Available at Folder Level

The contextual menu of a folder node in a SharePoint connection in the **Data Source Explorer** view contains the following actions:

New File

Creates a new file on the server in the current folder.

New Folder

Creates a new folder on the server.

Import Folders

Imports folders on the server.


Import Files

Allows you to add a new file on the server in the current folder.


Cut

Removes the current selection and places it in the clipboard.


Copy

Copies the current selection.


Paste

Pastes the copied selection.

Rename

Allows you to change the name of the selected folder.

Delete

Removes the selected folder.

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 into the clipboard.

Copy Location

Copies an application specific URL for the selected resource to the clipboard. You can use this URL for various actions like opening or transforming the resources.

Check Out

Checks out the selected document on the server.

Check In

Checks in the selected document on the server. This action opens the **Check In** dialog box. In this dialog box, the following options are available:

- **Minor Version** - Increments the minor version of the file on the server.
- **Major Version** - Increments the major version of the file on the server.
- **Overwrite** - Overwrites the latest version of the file on the server.
- **Comment** - Allows you to comment on a file that you check in.

Discard Check Out

Discards the previous checkout operation, making the file available for editing to other users.

Rename

Allows you to change the name of the selected file.

Delete

Removes the selected file.

Refresh

Performs a refresh of the selected node.

Properties

Displays the properties of the current file in a **Properties** dialog box.

Find/Replace in Files

Allows you to find and replace text in multiple files from the server.

Note: The **Check In**, **Check Out**, and **Discard Check Out** options are available in the Enterprise edition only.

Chapter 18

Extending Oxygen XML Editor plugin Using the SDK

Topics:

This chapter includes information about the available extension points for Eclipse.

Extension points for Oxygen XML Editor plugin

The Oxygen XML Editor plugin includes a number of extension points, which can be implemented by other Eclipse plugins that depend on it. All of them are listed in the `plugin.xml` file, along with samples of usage code. The following is a list with short descriptions for some of the most useful extension points:

Extension point: *ditaKeyDefinitionManager*

It can be used to provide an external keys manager, responsible of providing DITA keys that are then used for editing and resolving referenced content. Its EXSD schema can be found in:
`OXYGEN_PLUGIN_DIR/exsd-schema/ditaKeyDefinitionManager.exsd`.

Extension point: *actionBarContributorCustomizer*

A very useful extension point that can add or remove actions from various menus, contextual menus, and toolbars that are contributed by the Oxygen XML Editor plugin. Its EXSD schema can be found in:
`OXYGEN_PLUGIN_DIR/exsd-schema/actionBarContributorCustomizer.exsd`.

Extension point: *customEditorInputCreator*

Create your custom editor input for a certain resource that will be opened by the Oxygen XML Editor plugin when clicking links. Its EXSD schema can be found in:
`OXYGEN_PLUGIN_DIR/exsd-schema/customEditorInputCreator.exsd`.

Extension point: *editorAdapterContributor*

When an adapter is requested to the opened XML editor you can provide your custom adapter from your external plugin. Its EXSD schema can be found in: `OXYGEN_PLUGIN_DIR/exsd-schema/editorAdapterContributor.exsd`.

Extension point: *extensionsBundleContributor*

Provide your own `ExtensionsBundle` implementation for a certain opened XML resource. Its EXSD schema can be found in: `OXYGEN_PLUGIN_DIR/exsd-schema/extensionsBundleContributor.exsd`.

Extension point: *stylesFilterContributor*

Provide your own `StylesFilter` implementation for special visual rendering when an XML resource is opened in the **Author** editing mode. Its EXSD schema can be found in:
`OXYGEN_PLUGIN_DIR/exsd-schema/stylesFilterContributor.exsd`.

Extension point: *XMLRefactoringContributor*

Contribute a folder that contains the additional XML Refactoring operation descriptor files and XQuery scripts that can be used by the batch XML refactoring actions. Its EXSD schema can be found in:
`OXYGEN_PLUGIN_DIR/exsd-schema/xmlRefactoringContributor.exsd`.

Extension point: *AuthorStylesheet*

Use this extension point to provide a stylesheet layer that will be used when rendering any XML document in **Author** mode. Its EXSD schema can be found in:
`OXYGEN_PLUGIN_DIR/exsd-schema/authorStylesheetContributor.exsd`.

Chapter 19

Tools

Topics:

Oxygen XML Editor plugin includes a variety of helpful tools to help you accomplish XML-related tasks. This section presents many of those tools.

XML Digital Signatures

This chapter explains how to apply and verify digital signatures on XML documents.

Digital Signatures 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. Since the signature is dependent on the content it is signing, a signature produced from a non-canonicalized document could possibly be different from one produced from a canonicalized document. The canonical form of an XML document is physical representation of the document produced by the method described in this specification. The term canonical XML refers to XML that is in canonical form. The XML canonicalization method is the algorithm defined by this specification that generates the canonical form of a given XML document or document subset. The term XML canonicalization refers to the process of applying the XML canonicalization method to an XML document or document subset. XML canonicalization is designed to be useful to applications that require the ability to test whether the information content of a document or document subset has been changed. This is done by comparing the canonical form of the original document before application processing with the canonical form of the document result of the application processing.

A digital signature over the canonical form of an XML document or document subset would allows the signature digest calculations to be oblivious to changes in the original document's physical representation. During signature generation, the digest is computed over the canonical form of the document. The document is then transferred to the relying party, which validates the signature by reading the document and computing a digest of the canonical form of the received document. The equivalence of the digests computed by the signing and relying parties (and hence the equivalence of the canonical forms over which they were computed) ensures that the information content of the document has not been altered since it was signed.

The following canonicalization algorithms are used in Oxygen XML Editor plugin: Canonical XML (or Inclusive XML Canonicalization)(*XMLC14N*) and Exclusive XML Canonicalization(*EXCC14N*). The first is used for XML where the context doesn't change while the second was designed for canonicalization where the context might change.

Inclusive Canonicalization copies all the declarations, even if they are defined outside of the scope of the signature, and all the declarations you might use will be unambiguously specified. Inclusive Canonicalization is useful when it is less likely that the signed data will be inserted in other XML document and it is 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. A problem may occur if the signed document is moved into another XML document that has other declarations because the Inclusive Canonicalization will copy them and the signature will be invalid.

Exclusive Canonicalization just copies the namespaces you are actually using (the ones that are a part of the XML syntax). It does not look into attribute values or element content, so the namespace declarations required to process these are not copied. This is useful if you have a signed XML document that you want to insert into other XML documents (or you need self-signed structures that support placement within different XML contexts), as it will ensure the signature is verified correctly each time.

The canonicalization method can specify whether or not comments should be included in the canonical form output by the XML canonicalization method. If a canonical form contains comments corresponding to the comment nodes in the input node-set, the result is called canonical XML with comments. In an uncommented canonical form comments are removed, including delimiter for comments outside document element.

The three operations. **Canonicalize**, **Sign**, and **Verify Signature**, are available from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **XML Tools** menu.

Canonicalizing Files

You can select the canonicalization algorithm to be used for a document from the dialog box that is displayed by using the **Canonicalize** action that is available from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **XML Tools** menu.

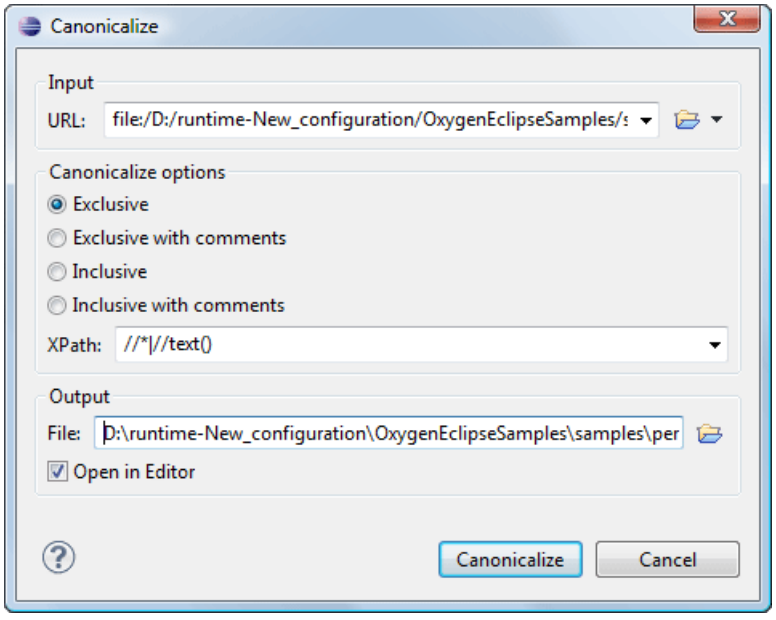


Figure 376: Canonicalization Settings Dialog Box

You can set the following:

- **Input URL** - Available if the **Canonicalize** action was selected from the **XML Tools** menu. It allows you to specify the location of the input file.
- **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.

Note: Exclusive Canonicalization just copies the namespaces you are actually using (the ones that are a part of the XML syntax). It does not look into attribute values or element content, so the namespace declarations required to process these are not copied. This is useful if you have a signed XML document that you want to insert into other XML documents (or you need self-signed structures that support placement within different XML contexts), as it will ensure the signature is verified correctly each time.

- **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
- **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.

Note: Inclusive Canonicalization copies all the declarations, even if they are defined outside of the scope of the signature, and all the declarations you might use will be unambiguously specified. Inclusive Canonicalization is useful when it is less likely that the signed data will be inserted in other XML document and it is 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. A problem may occur if the signed document is moved into another XML document that has other declarations because the Inclusive Canonicalization will copy them and the signature will be invalid.

- **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** - Available if the **Canonicalize** action was selected from the **XML Tools** menu. It allows you to specify the output file path where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in the editor.

Certificates

A certificate is a digitally signed statement from the issuer (an individual, an organization, a website or a firm), saying that the public key (and some other information) of some other entity has a particular value. When data is digitally signed, the signature can be verified to check the data integrity and authenticity. Integrity means that the data has not been modified. Authenticity means the data comes indeed from the entity that claims to have created and signed it. Certificates are kept in special repositories called *keystores*.

A *keystore* is an encrypted file that contains private keys and certificates. All *keystore* entries (key and trusted certificate entries) are accessed via unique aliases. An alias must be assigned for every new entry of either a key or certificate as a reference for that entity. No *keystore* can store an entity if its alias already exists in that *keystore* and cannot store trusted certificates generated with keys in its *keystore*.

In Oxygen XML Editor plugin there are provided two types of *keystores*: Java Key Store (JKS) and Public-Key Cryptography Standards version 12 (PKCS-12). A *keystore* file is protected by a password. In a PKCS 12 *keystore* you should not store a certificate without alias together with other certificates, with or without alias, as in such a case the certificate without alias cannot be extracted from the *keystore*.

To configure the options for a certificate or to validate it, *open the **Preferences** dialog box* and go to **XML** > **XML Signing Certificates**. This opens *the certificates preferences page*.

Signing Files

You can select the type of signature to be used for documents from a signature settings dialog box. To open this dialog box, select the **Sign** action from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **XML Tools** menu.

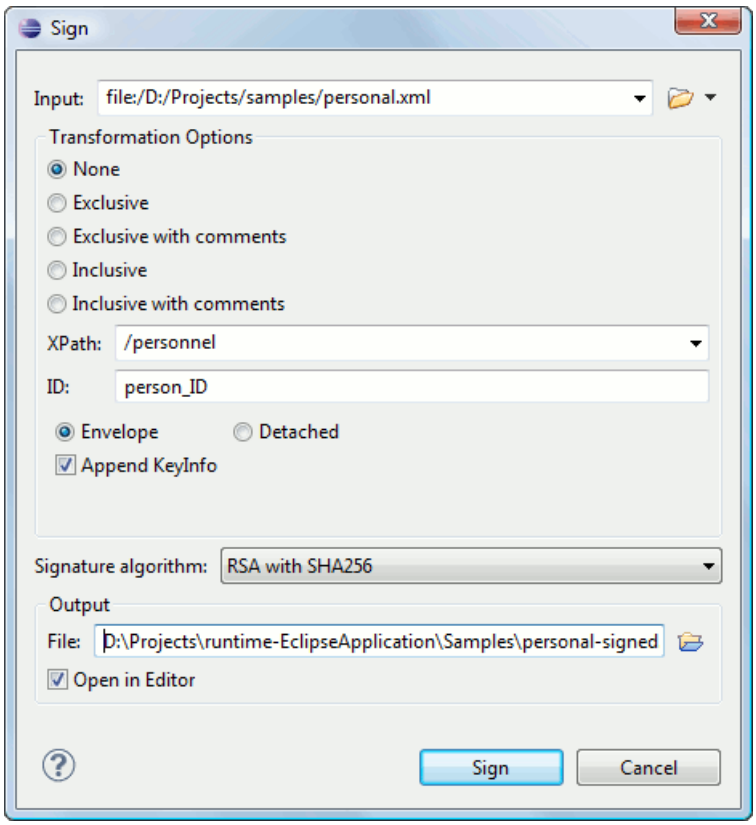


Figure 377: Signature Settings Dialog Box

The following options are available:

Note: If Oxygen XML Editor plugin could not find a valid certificate, a link is provided at the top of the dialog box that opens the *XML Signing Certificates preferences page* where you can configure a valid certificate.

- **Input** - Available if the **Sign** action was selected from the **XML Tools** menu. Specifies the location of the input URL.
- **Transformation Options** - See the *Digital Signature Overview* section for more information about these options.
  - **None** - If selected, no canonicalization algorithm is used.
  - **Exclusive** - If selected, the exclusive (uncommented) canonicalization method is used.

    Note: Exclusive Canonicalization just copies the namespaces you are actually using (the ones that are a part of the XML syntax). It does not look into attribute values or element content, so the namespace declarations required to process these are not copied. This is useful if you have a signed XML document that you want to insert into other XML documents (or you need self-signed structures that support placement within different XML contexts), as it will ensure the signature is verified correctly each time.

  - **Exclusive with comments** - If selected, the exclusive with comments canonicalization method is used.
  - **Inclusive** - If selected, the inclusive (uncommented) canonicalization method is used.

    Note: Inclusive Canonicalization copies all the declarations, even if they are defined outside of the scope of the signature, and all the declarations you might use will be unambiguously specified. Inclusive Canonicalization is useful when it is less likely that the signed data will be inserted in other XML document and it is 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. A problem may occur if the signed document is moved

into another XML document that has other declarations because the Inclusive Canonicalization will copy them and the signature will be invalid.

  - **Inclusive with comments** - If selected, the inclusive with comments canonicalization method is used.

- **XPath** - The XPath expression provides the fragments of the XML document to be signed.
- **ID** - Provides ID of the XML element to be signed.
- **Envelope** - If selected, the *enveloped* signature is used. See the *Digital Signature Overview* for more information.
- **Detached** - If selected, the *detached* signature is used. See the *Digital Signature Overview* for more information.
- **Append KeyInfo** - If this option is checked, the `ds:KeyInfo` element will be added in the signed document.
- **Signature algorithm** - The algorithm used for signing the document. The following options are available: **RSA with SHA1**, **RSA with SHA256**, **RSA with SHA384**, and **RSA with SHA512**.
- **Output** - Available if the **Sign** action was selected from the **XML Tools** menu. Specifies the path of the output file where the signed XML document will be saved.
- **Open in editor** - If checked, the output file will be opened in Oxygen XML Editor plugin.

Verifying the Signature

You can verify the signature of a file by selecting the **Verify Signature** action from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **XML Tools** menu. The **Verify Signature** dialog box then allows you to specify the location of the file whose signature is verified.

If the signature is valid, a dialog box displays the name of the signer. Otherwise, an error shows details about the problem.

Example of How to Digitally Sign XML Files or Content

Suppose you want to digitally sign an XML document, but more specifically, suppose you have multiple instances of the same element in the document and you just want to sign a specific ID. Oxygen XML Editor plugin includes a signature tool that allows you to digitally sign XML documents or specific content.

The Oxygen XML Editor plugin installation directory includes a `samples` folder that contains a file called `personal.xml`. For the purposes of this example, this file will be used to demonstrate how to digitally sign specific content. Notice that this file has multiple `person` elements inside the `personnel` element. Suppose you want to digitally sign the specific `person` element that contains the `id=robert.taylor`. To do this, follow this procedure:

1. Open the `personal.xml` file in Oxygen XML Editor plugin in **Text** editing mode.
2. Right-click anywhere in the editor and select the **Sign** action from the **Source** submenu.
   The **Sign** dialog box is displayed.

   Tip: If you want to sign a file but create a new output file so that the original file remains unchanged, use the **Sign** action from the **XML Tools** menu. Selecting the action from this menu will allow you to choose an input file and output file in the **Sign** dialog box.

3. If Oxygen XML Editor plugin cannot find a valid certificate, click the link at the top of the dialog box to **configure a valid certificate**. This opens the ***XML Signing Certificates*** *preferences page* that allows you to configure and validate a certificate.
4. Once a valid certificate is recognized, continue to configure the **Sign** dialog box.
   a) Select one of *the* ***Transformation Options***. For the purposes of this example, select the **Inclusive with comments** option.
   b) Specify the appropriate **XPath** expression for the specific element that needs to be signed. For this example, type `/personnel/person` in the **XPath** text box.
   c) Enter the specific **ID** that needs to be signed. For this example, type `robert.taylor` in the **ID** field.
   d) Select *the* ***Envelope*** *option* and leave the other options as their default values.

   The digital signature is added at the end of the XML document, just before the end tag. It is always added at the end of the document, even if you only sign specific content within the document.

5. You can verify the signature by choosing the **Verify Signature** action from the **Source** submenu of the contextual menu.

Chapter 20

Configuring Oxygen XML Editor plugin

Topics:

This chapter presents all the user preferences and options that allow you to configure various features, or the application itself, and the editor variables that are available for customizing the user-defined commands.

Preferences

You can configure Oxygen XML Editor plugin options using the **Preferences** dialog box.

To open the preferences dialog box, go to go to **Window** (**Eclipse** on Mac OSX) and choose **Preferences** > **Oxygen XML Editor plugin**.

You can restore options to their default values by pressing the **Restore Defaults** button, available in each preferences page.

Press or **F1** for help on any preferences page.

A limited version of the **Preferences** dialog box is available by selecting **Options** from the contextual menu in the editor.

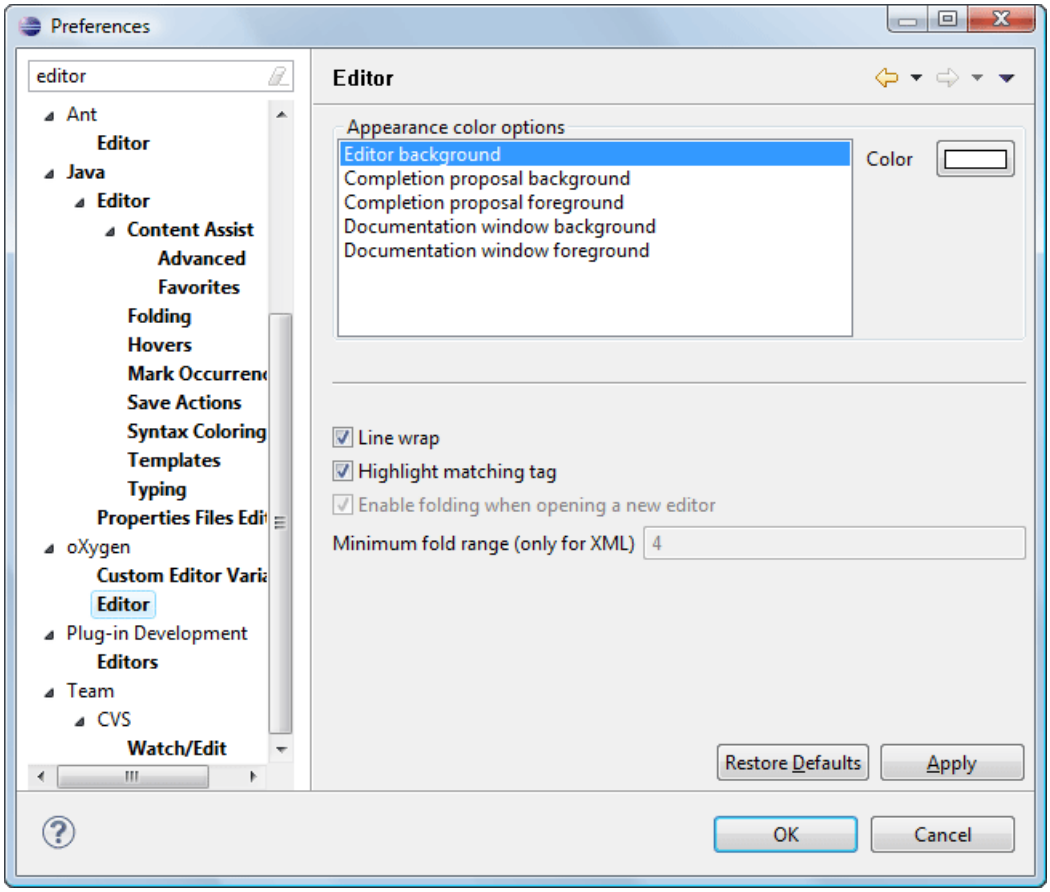


Figure 378: Eclipse Preferences Dialog Box - Restricted Version

Preferences Directory Location

A variety of resources (such as global options, license information, and history files) are stored in a preferences directory (`com.oxygenxml`) that is in the following locations:

- **Windows (Vista, 7, 8, 10)** - `[user_home_directory]\AppData\Roaming\com.oxygenxml`
- **Windows XP** - `[user_home_directory]\Application Data\com.oxygenxml`
- **Mac OS X** - `[user_home_directory]/Library/Preferences/com.oxygenxml`
- **Linux/Unix** - `[user_home_directory]/.com.oxygenxml`

Oxygen XML Editor plugin License

To configure the license options, *open the **Preferences** dialog box*. This preferences page presents the details of the license key that enables the Oxygen XML Editor plugin plugin, such as registration name, category and number of purchased licenses, encrypted signature of the license key. Clicking the **Register** button opens the Oxygen XML Editor plugin **License** dialog box that allows you to insert a new license key.

Archive Preferences

To configure **Archive** preferences, *open the **Preferences** dialog box* and go to **Archive**.

The following options are available in the **Archive** preferences panel:

- **Archive backup options** - Controls if the application makes backup copies of the modified archives. The following options are available:
  - **Always create backup copies of modified archives** - When you modify an archive, its content is backed up.
  - **Never create backup copies of modified archives** - No backup copy is created.
  - **Ask for each archive once per session** - Once per application session for each modified archive, user confirmation is required to create the backup. This is the default setting.

    Note: Backup files have the name `originalArchiveFileName.bak` and are located in the same folder as the original archive.

- **Show archive backup dialog box** - Select this option if you want to be notified for backup when modifying in archives. The last backup option you chose will always be used as the default one.
- **Archive types** - This table contains all known archive extensions mapped to known archive formats. Each row maps a list of extensions to an archive type supported in Oxygen XML Editor plugin. You can edit an existing mapping or create a new one by associating your own list of extensions to an archive format.

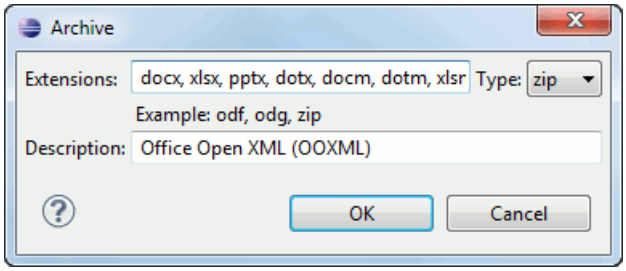


Figure 379: Edit Archive Extension Mappings

Important: You have to restart Oxygen XML Editor plugin after removing an extension from the table in order for that extension to not be recognised anymore as an archive extension.

- **Store Unicode file names in Zip archives** - Use this option when you archive files that contain international (that is, non-English) characters in file names or file comments. If this option is selected and an archive is modified in any way, UTF-8 characters are used in the names of all files in the archive.

CSS Validator Preferences

To configure the **CSS Validator** preferences, *open the **Preferences** dialog box* and go to **CSS Validator**.

You can configure the following options for the built-in **CSS Validator** of Oxygen XML Editor plugin:

- **Profile** - Selects one of the available validation profiles: **CSS 1**, **CSS 2**, **CSS 2.1**, **CSS 3**, **CSS 3 with Oxygen extensions**, **SVG**, **SVG Basic**, **SVG Tiny**, **Mobile**, **TV Profile**, **ATSC TV Profile**. The **CSS 3 with Oxygen extensions** profile includes all the CSS 3 standard properties plus 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** - You can type comma separated patterns that match the names of CSS properties that will be ignored at validation. As wildcards you can use:
  - * to match any string.
  - **?** to match any character.
- **Recognize browser CSS extensions (applies also to content completion)** - If checked, Oxygen XML Editor plugin recognizes (no validation is performed) browser-specific CSS properties. The **Content Completion Assistant** lists these properties at the end of its list, prefixed with the following particles:
  - `-moz-` for Mozilla.
  - `-ms-` for Internet Explorer.
  - `-o-` for Opera.
  - `-webkit-` for Safari/Webkit.

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.

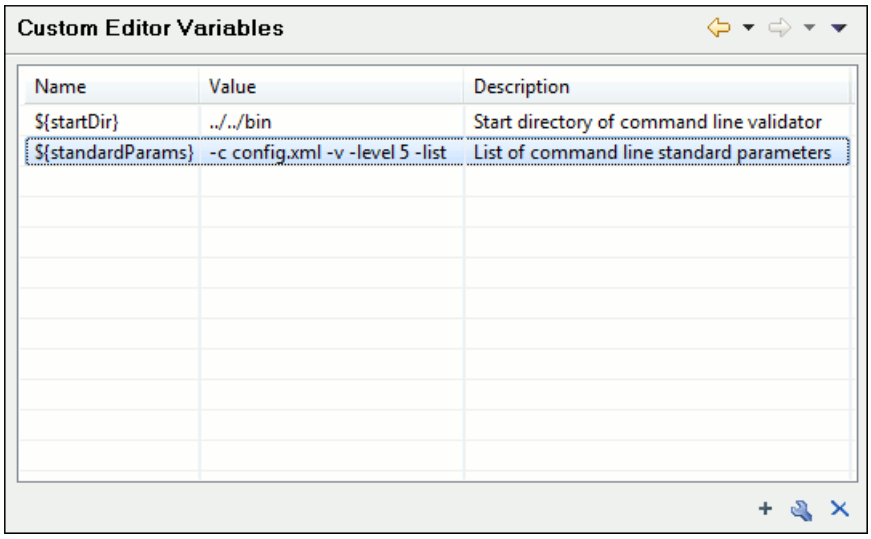


Figure 380: Custom Editor Variables

Data Sources Preferences

To configure the **Data Sources** preferences, *open the **Preferences** dialog box* and go to **Data Sources**.

Data Sources Preferences

To configure the **Data Sources** preferences, *open the **Preferences** dialog box* and go to **Data Sources**. In this preferences page you can configure data sources and connections to relational databases as well as native XML databases. You can check the list of drivers available for the major database servers here: *http://www.oxygenxml.com/database_drivers.html*.

Connection Wizards Section

Create eXist-db XML connection

Click this link to open the dedicated ***Create eXist-db XML connection*** *dialog box* that provides a quick way to create an eXist connection.

Data Sources Section

This section allows you to add and configure data sources.

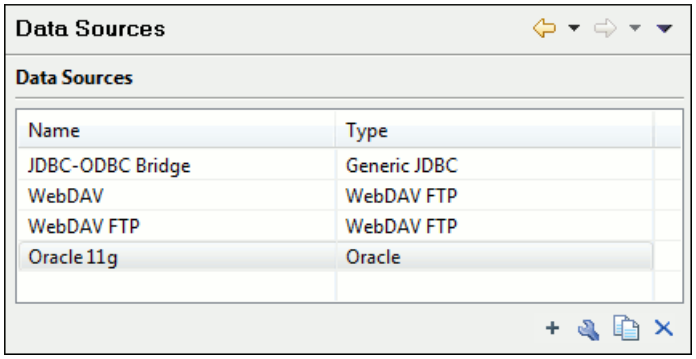


Figure 381: The Data Sources Preferences Panel

The following buttons are available at the bottom of the **Data Sources** panel:

New

Opens the **Data Sources Drivers** dialog box that allows you to configure a new database driver.

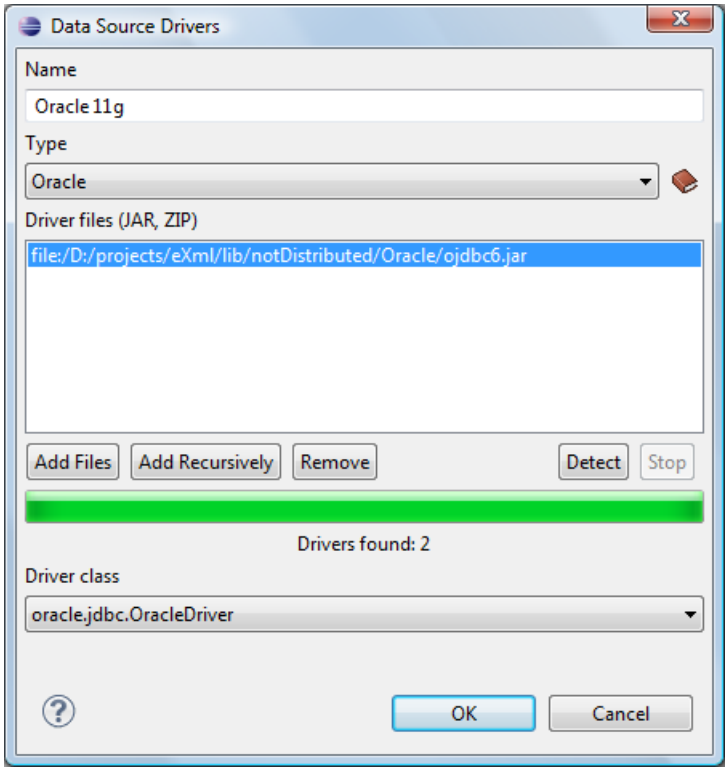


Figure 382: The Data Sources Drivers Dialog Box

The following options are available in the **Data Source Drivers** dialog box:

- **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 button** - 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 files (JAR, ZIP)** - Lists *download links for database drivers* that are necessary for accessing databases in Oxygen XML Editor plugin.
- **Add Files** - Adds the driver class library.
- **Add Recursively** - Adds driver files recursively.
- **Remove** - Removes the selected driver class library from the list.
- **Detect** - Detects driver file candidates.
- **Stop** - Stops the detection of the driver candidates.
- **Driver class** - Specifies the driver class for the data source driver.

Edit

Opens the **Data Sources Drivers** dialog box for editing the selected driver. See above the specifications for the **Data Sources Drivers** dialog box. In order to edit a data source, there must be no connections using that data source driver.

Duplicate

Creates a copy of the selected data source.

Delete

Deletes the selected driver. In order to delete a data source, there must be no connections using that data source driver.

Connections Section

This section allows you to add and configure data source connections.

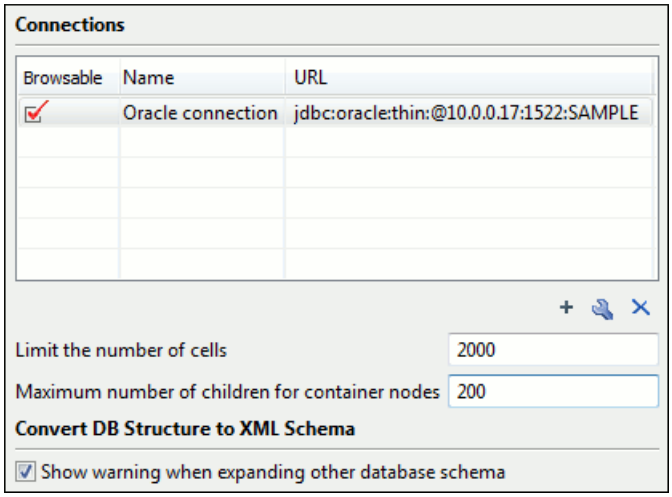


Figure 383: The Connections Preferences Panel

The following buttons are available at the bottom of the **Connections** panel:

New

Opens the **Connection** dialog box that allows you to configure a new database connection.

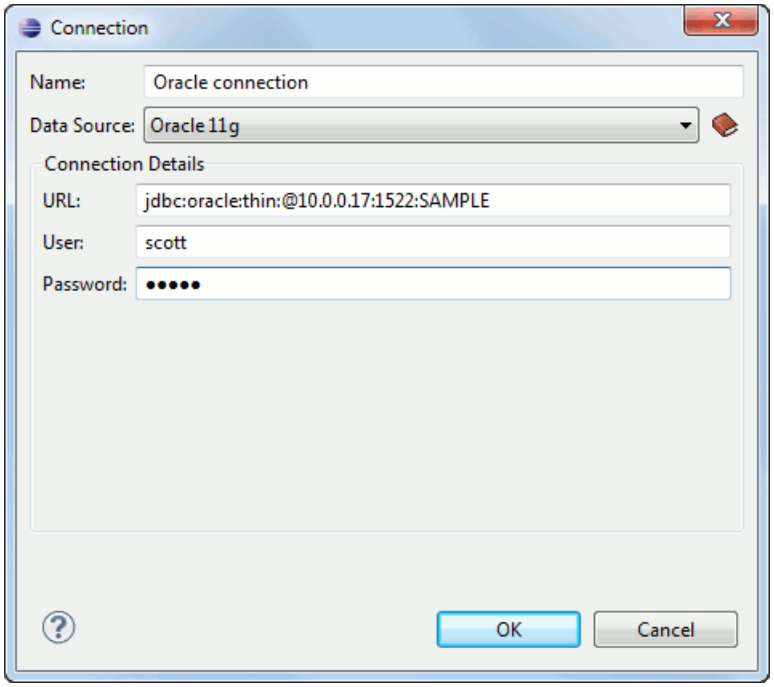


Figure 384: The Connection Dialog Box

The following options are available in the **Connection** dialog box:

- **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 box.

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 box, allowing you to edit the selected connection. See above the specifications for the **Connection** dialog box.

Duplicate

Creates a copy of the selected connection.

Delete

Deletes the selected connection.

Move Up

Moves the selected connection up one row in the list.

Move Down

Moves the selected connection down one row in the list.

Limit the number of cells

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 this field empty if you want the entire content of the table to be displayed. By default, this field is set to 2000. If a table that has more cells than the value set here is displayed in the **Table Explorer** view, a warning dialog box will inform you that the table is only partially shown.

Maximum number of children for container nodes

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 this, 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 field. The default value is 200 nodes.

Show warning when expanding other database schema

Controls if a warning message will be displayed when expanding another database schema and there are tables selected in the current expanded one. This applies to the **Select database table** dialog box in the ***Import Database Data** wizard* and the **Select database table** section of the ***Convert DB Structure to XML Schema** dialog box*.

Download Links for Database Drivers

Below you can find instructions for getting the drivers that are necessary to access databases in Oxygen XML Editor plugin.

- **Berkeley DB XML database** - Copy the *jar* files from the Berkeley database install directory into the Oxygen XML Editor plugin install directory as described in *the procedure for configuring a Berkeley DB data source*.
- **IBM DB2 Pure XML database** - Go to the *IBM website* and in the *DB2 Clients and Development Tools* category select the *DB2 Driver for JDBC and SQLJ* download link. Fill out the download form and download the zip file. Unzip the zip file and use the `db2jcc.jar` and `db2jcc_license_cu.jar` files in Oxygen XML Editor plugin for *configuring a DB2 data source*.
- **eXist database** - Copy the *jar* files from the eXist database install directory to the Oxygen XML Editor plugin install directory as described in *the procedure for configuring an eXist data source*.
- **MarkLogic database** - Download the MarkLogic driver from *MarkLogic Community site*.
- **Microsoft SQL Server 2005 / 2008 database** - Download the appropriate MS SQL JDBC driver from the Microsoft website. For SQL Server 2008 R2 and older go to *http://www.microsoft.com/en-us/download/details.aspx?id=21599*. For SQL Server 2012 and 2014 go to *http://www.microsoft.com/en-us/download/details.aspx?id=11774*.
- **Oracle 11g database** - Go to the *Oracle website* and download the Oracle 11g JDBC driver called `ojdbc6.jar`.
- **PostgreSQL 8.3 database** - Go to the *PostgreSQL website* and download the PostgreSQL 8.3 JDBC driver called `postgresql-8.3-603.jdbc3.jar`.
- **Documentum xDb (X-Hive/DB) 10 XML database** - Copy the *jar* files from the Documentum xDb (X-Hive/DB) 10 database install directory to the Oxygen XML Editor plugin install directory as described in *the procedure* for configuring a Documentum xDb (X-Hive/DB) 10 data source.

Table Filters Preferences

to configure the **Table Filters** preferences, *open the **Preferences** dialog box* and go to **Data Sources** > **Table Filters**.

Here you can choose which of the database table types will be displayed in the **Data Source Explorer** view.

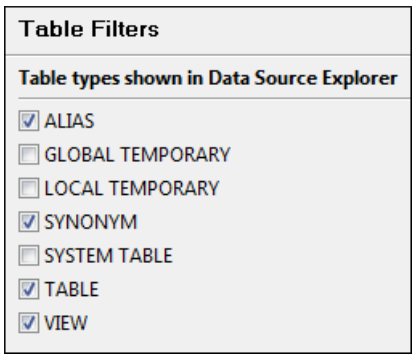


Figure 385: Table Filters Preferences Panel

DITA Preferences

To access the DITA Preferences page, *open the **Preferences** dialog box* and go to **DITA**. This preferences page includes the following sections and options:

DITA Open Toolkit Section

This section allows you to specify the default directory of the DITA Open Toolkit distribution (bundled with the Oxygen XML Editor plugin installation) to be used for validating and publishing DITA content. You can select from the following:

- **Built-in DITA-OT 1.8** - This is the default setting and the default DITA OT directory is: `[OXYGEN_DIR]/frameworks/dita/DITA-OT`.
- **Built-in DITA-OT 2.x (with experimental DITA 1.3 support)** - All defined DITA transformation scenarios will run with DITA-OT 2.x. This also gives you access to new DITA 1.3 file templates when you *create new documents from templates*. The default DITA OT directory is: `[OXYGEN_DIR]/frameworks/dita/DITA-OT2.x`.
- **Custom** - In the **Location** field, you can either provide a new file path for the specific DITA OT that you want to use or you can select a previously used one from the drop-down list.

The **Show console output** option allows you to specify when to display the console output log. The following options are available:

- **When build fails** - displays the console output log if the build fails.
- **Always** - displays the console output log, regardless of whether or not the build fails.

The following options are available for the **Insert Reference** and **Edit Properties** operations:

- **Always fill values for attributes** - Allows you to specify that in the **Insert Reference\Edit Properties** dialog box, the values for these attributes will be automatically populated with a detected value (based on the specifications), even if it is the same as the default value.
  - **Format** - If enabled, the *Format* attribute will always be populated automatically .
  - **Scope** - If enabled, the *Scope* attribute will always be populated automatically.
  - **Type** - If enabled, the *Type* attribute will always be populated automatically.

At the bottom of the page there is a link to the *Profiling Attributes preferences*, where you can configure how profiling and conditional text is displayed in **Author** mode.

Document Type Association Preferences

Oxygen XML Editor plugin uses document type associations to associate a *document type* with a set of functionality provided by a framework. To configure the **Document Type Association** options, *open the **Preferences** dialog box* and go to **Document Type Association**.

The following actions are available in this preferences page:

Discover more frameworks by using add-ons update sites

Click on this link to specify URLs for framework add-on update sites.

Document Type Table

This table presents the currently defined document type associations (frameworks), sorted by priority and alphabetically. Each edited document type has a set of association rules (used by the application to detect the proper document type association to use for an opened XML document). A rule is described by:

- **Namespace** - Specifies the namespace of the root element from the association rules set (* *(any)* by default). If you want to apply the rule only when the root element has no namespace, leave this field empty (remove the **ANY_VALUE** string).
- **Root local name** - Specifies the local name of the root element (* *(any)* by default).
- **File name** - Specifies the name of the file (* *(any)* by default).
- **Public ID** - Represents the Public ID of the matched document.
- **Java class** - Presents the name of the Java class, which is used to determine if a document matches the rule.

New

Opens a ***Document type*** *configuration dialog box* that allows you to add a new association.

Edit

Opens a ***Document type*** *configuration dialog box* that allows you to edit an existing association.

Note: If you try to edit an existing association type when you do not have write permissions to its store location, a dialog box will be shown asking if you want to extend the document type.

Duplicate

Opens a ***Document type*** *configuration dialog box* that allows you to duplicate the configuration of an existing document type association.

Extend

Opens a ***Document type*** *configuration dialog box* that allows you to extend an existing document type. You can add or remove functionality starting from a base document type. All of these changes will be saved as a patch. When the base document type is modified and evolves (for example, from one application version to another) the extension will evolve along with the base document type, allowing it to use the new actions added in the base document type.

Delete

Deletes the selected document type associations.

Enable DTD/XML Schema processing in document type detection

When this option is enabled (default value), 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 Editor plugin to correctly match the DITA customization.

Only for local DTDs/XML Schemas

When this option is enabled (default value), only the local DTDs / XML Schemas will be processed.

Enable DTD/XML Schema caching

When this option is enabled (default value), the associated DTDs or XML Schema are cached when parsed for the first time, improving performance when opening new documents with similar schema associations.

Locations Preferences

Oxygen XML Editor plugin allows you to change the location where *document types (frameworks)* are stored, and to specify additional framework directories. The **Locations** preferences page allows you to specify the main `frameworks` folder location. You can choose between the **Default** directory (`[OXYGEN_DIR]/frameworks`) or a **Custom** specified directory. You can also change the current `frameworks` folder location value using the `com.oxygenxml.editor.frameworks.url` system property.

A list of additional `frameworks` directories can also be specified. The application will look in each of those folders for additional document type configurations to load. Use the **Add**, **Edit** and **Delete** buttons to manage the list of folders.

A document type (configuration) can be loaded from the following locations:

- Internal preferences - The document type configuration is stored in the application ***Internal*** *preferences*.
- Additional frameworks directories - The document type configuration is loaded from one of the specified **Additional frameworks directories** list.
- The `frameworks` folder - The main folder containing framework configurations.

All loaded document type configurations are first sorted by priority, then by document type name and then by load location (in the exact order specified above). When an XML document is opened, the application chooses the first document type configuration from the sorted list which matches the specific document.

All loaded document type configurations are first sorted by priority, then by document type.

The Document Type Configuration Dialog Box

The **Document type** configuration dialog box allows you to create or edit a *Document Type Association* (framework). The following fields are available in this dialog box:

- **Name** - The name of the *Document Type Association*.
- **Storage** - Displays the type of location where the framework configuration file is stored. Can be one of: **External** (framework configuration is saved in a file) or **Internal** (framework configuration is stored in the application's internal options).

  Note: If you set the **Storage** to **Internal** and the document type association settings are already stored in a framework file, the file content is saved in the application's internal options and the file is removed.

- **Description** - A detailed description of the framework.
- **Priority** - Depending on the priority level, Oxygen XML Editor plugin establishes the order in which the existing document type associations are evaluated to determine the type of a document you are opening. It can be one of the following: Lowest, Low, Normal, High, or Highest. You can set a higher priority to *Document Type Associations* you want to be evaluated first.
- **Initial edit mode** - Sets the default edit mode when you open a document for the first time.

You are able to configure the options of each *framework* in the following tabs:

- ***Association rules***
- ***Schema***
- ***Classpath***
- ***Author***
- ***Templates***
- ***Catalogs***
- ***Transformation***
- ***Validation***
- ***Extensions***

The Association Rules Tab

By combining multiple association rules you can instruct Oxygen XML Editor plugin to identify the type of a document. An Oxygen XML Editor plugin *association rule* holds information about *Namespace*, *Root local name*, *File name*, *Public ID*, *Attribute*, and *Java class*. Oxygen XML Editor plugin identifies the type of a document when the document matches at least one of the *association rules*. Using the **Document type rule** dialog box, you can create *association rules* that activate on any document matching all the criteria in the dialog box.

In the **Association rules** tab you can perform the following actions:

New

Opens the **Document type rule** dialog box allowing you to create *association rules*.

Edit

Opens the **Document type rule** dialog box allowing you to edit the properties of the currently selected *association rule*.

Delete

Deletes the currently selected *association rules*.

Move Up

Moves the selection to the previous *association rule*.

Move Down

Moves the selection to the following *association rule*.

The Schema Tab

In the **Schema** tab, you can specify a schema that Oxygen XML Editor plugin uses if an XML document does not contain a schema declaration and no default validation scenario is associated with it.

To set the **Schema URL**, use *editor variables* to specify the path to the Schema file.

Note: It is a good practice to store all resources in the framework directory and use the `${framework}` editor variable to reference them. This is a recommended approach to designing a self-contained document type that can be easily maintained and shared between different users.

The Classpath Tab

The **Classpath** tab displays a list of folders and JAR libraries that hold implementations for API extensions, implementations for custom **Author** mode operations, different resources (such as stylesheets), and framework translation files. Oxygen XML Editor plugin loads the resources looking in the folders in the order they appear in the list.

In the **Classpath** tab you can perform the following actions:

New

Opens a dialog box that allows you to add a resource in the **Classpath** tab.

Edit

Opens a dialog box that allows you to edit a resource in the **Classpath** tab.

Delete

Deletes the currently selected resource.

Move Up

Moves the selection to the previous resource.

Move Down

Moves the selection to the following resource.

The Author Tab

The **Author** tab is a container that holds information regarding the CSS file used to render a document in the **Author** mode, and regarding framework-specific actions, menus, contextual menus, toolbars, and content completion list of proposals.

The options that you configure in the **Author** tab are grouped in the following sub-tabs:***CSS***, ***Actions***, ***Menu***, ***Contextual menu***, ***Toolbar***, ***Content Completion***.

CSS

The **CSS** subtab contains the CSS files that Oxygen XML Editor plugin uses to render a document in the **Author** mode. In this subtab, you can set *main* and *alternate* CSS files. When you are editing a document in the **Author** mode, you can switch between these CSS files from the **Styles** drop-down menu on the **Author Styles** toolbar.

The following actions are available in the **CSS** subtab:

New

Opens a dialog box that allows you to add a CSS file.

Edit

Opens a dialog box that allows you to edit a CSS file.

Delete

Deletes the currently selected CSS file.

Move Up

Moves the selected CSS file up in the list.

Move Down

Moves the selected CSS file down in the list.

Enable multiple selection of alternate CSSs

Allows users to apply multiple alternate styles, as layers, over the main CSS style. This option is enabled by default for DITA document types.

ignore CSSs from the associated document type

The CSS files set in the CSS tab are overwritten by the CSS files specified in the document itself.

merge them with CSSs from the associated document type

The CSS files set in the CSS tab are merged with the CSS files specified in the document itself.

Actions

The **Actions** sub-tab holds the framework specific actions. Each action has an unique ID, a name, a description, and a shortcut key.

The following actions are available in this sub-tab:

New

Opens *the **Action** dialog box* that allows you to add an action.

Duplicate

Duplicates the currently selected action.

Edit

Opens a dialog box that allows you to edit an existing action.

Delete

Deletes the currently selected action.

The Action Dialog Box

To edit an existing document type action or create a new one, *open the **Preferences** dialog box*, go to **Document Type Association**, select a document type, and click **Edit** or **New**. The ***Document type** configuration dialog box* is presented.

In this dialog box, go to the **Author** tab, click **Actions**, select an action, and click **Edit**, or to create a new action click **New**.

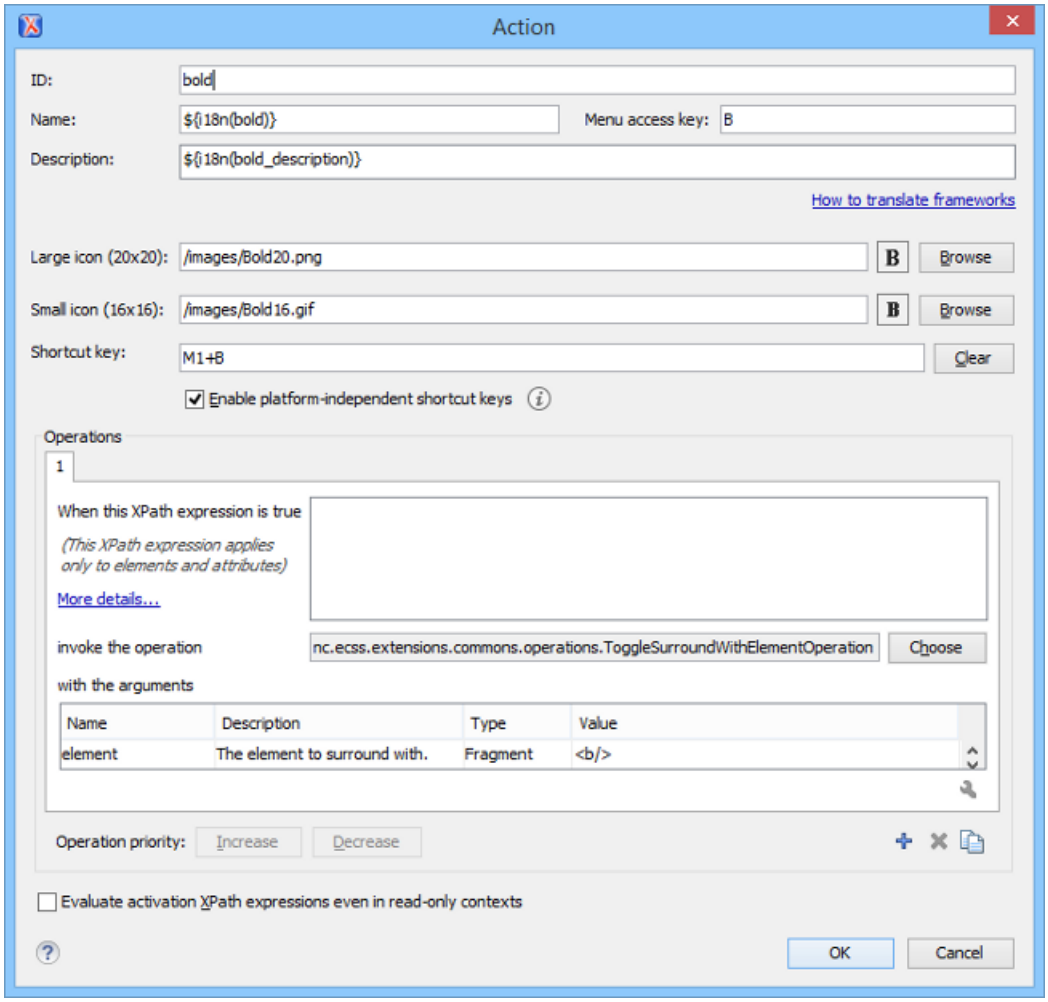


Figure 386: The Action Dialog Box

The following options are available in the **Action** dialog box:

- **ID** - Specifies a unique action identifier.
- **Name** - Specifies the name of the action. This name is displayed as a tooltip or as a menu item.
- **Menu access key** - In Windows, the menu items are accessed using the shortcut when the menu is visible. The letter is visually represented by underlining the first occurrence of the letter in the menu item **Name**.
- **Description** - A description of the action.
- **Large icon** - Allows you to select an image for the icon that Oxygen XML Editor plugin uses for the toolbar action.

  Tip: A good practice is to store the image files inside the framework directory and use the `${frameworks}` editor variable to make the image relative to the framework location. If the images are bundled in a *jar* archive (for instance, along with some Java operations implementation), it is convenient to reference the images by their relative path location in the *class-path*.

- **Small icon** - Allows you to select an image for the icon that Oxygen XML Editor plugin uses for the contextual menu action.

Note: If you are using a Retina or HiDPI display, Oxygen XML Editor plugin automatically searches for higher resolution icons in the path specified in both the **Large icon** and **Small icon** options. For more information, see *the Adding Retina/HiDPI Icons in a Framework section*.

- **Shortcut key** - This field allows you to configure a shortcut key for the action that you are editing. The + character separates the keys. If the **Enable platform-independent shortcut keys** checkbox is enabled, the shortcut that you specify in this field is platform-independent and the following modifiers are used:
  - `M1` represents the **Command** key on MacOS X, and the **Ctrl** key on other platforms.
  - `M2` represents the **Shift** key.
  - `M3` represents the **Option** key on MacOS X, and the **Alt** key on other platforms.
  - `M4` represents the **Ctrl** key on MacOS X, and is undefined on other platforms.
- **Operations**

  In this section of the **Action** dialog box, you configure the functionality of the action that you are editing. An action has one or more operation modes. The evaluation of an XPath expression activates an operation mode. The first enabled operation mode is activated when you trigger the action. The scope of the XPath expression must consist only of element nodes and attribute nodes of the edited document. Otherwise, the XPath expression does not return a match and does not fire the action. For more details see: *Activation of Multiple Functions for Actions using XPath Expressions* on page 921.

  The following options are available in this section:
  - **When this XPath expression is true** - An XPath 2.0 expression that applies to elements and attributes. For more details see:*Activation of Multiple Functions for Actions using XPath Expressions* on page 921.
  - **invoke the operation** - Specifies the invoked operation.
  - **with the arguments** - Specifies the arguments of the invoked operation.
  - **Edit** - Allows you to edit the arguments of the operation.
- **Operation priority** - Increases or decreases the priority of an operation. The operations are invoked in the order of their priority. If more than one XPath expression is true, the operation with the highest priority is invoked.
  - **Add** - Adds an operation.
  - **Remove** - Removes an operation.
  - **Duplicate** - Duplicates an operation.
- **Evaluate activation XPath expressions even in read-only cotnexts** - If this checkbox is enabled, the action can be invoked even when the cursor is placed in a read-only location.

Activation of Multiple Functions for Actions using XPath Expressions

An **Author** mode action can have multiple functions, each function invoking an **Author** mode operation with certain configured parameters. Each function of an action has an XPath 2.0 expression for activating it.

For each function of an action, 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 function 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 cursor 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 if 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 prefix-qualified name of an element.

```
oxy:allows-child-element("prefix:para")
```

  The prefix is resolved in the context of the element where the cursor is located. The function matches on the element with the para local name from the previous resolved namespace. If the prefix is not resolved to a namespace, the function returns a value of `false`.
- A specified namespace-URI-qualified name of an element.

```
oxy:allows-child-element("{namespaceURI}para")
```

  The `namespaceURI` is the namespace of the element. The above example verifies if the para element (of the specified namespace) is allowed in the current context.

- 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 if 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 cursor is located. If the prefix is not resolved to a namespace, the function returns a value of `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. If the value is `false`, the two values must be the same.

The `oxy:current-selected-element()` Function

This function returns the fully selected element. If 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 Editor plugin menu. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the Oxygen XML Editor plugin menu. To add an action in this section as a sibling of the currently selected action, use the **Add as sibling** button. To add an image in this section as a child of the currently selected action use the **Add as child** button.

The following actions are available in the **Current actions** section:

Edit
Edits an item.

Remove
Removes an item.

Move Up
Moves an item up.

Move Down
Moves an item down.

Contextual Menu

In the **Contextual menu** sub-tab you configure what framework-specific action the **Content Completion Assistant** proposes. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the contextual menu of a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the **Add as sibling** button. To add an action in this section as a child of the currently selected action use the **Add as child** button.

The following actions are available in the **Current actions** section:

Edit
Edits an item.

Remove
Removes an item.

Move Up
Moves an item up.

Move Down
Moves an item down.

Toolbar

In the **Toolbar** sub-tab you configure what framework-specific action the Oxygen XML Editor plugin toolbar holds. The sub-tab is divided in two sections: **Available actions** and **Current actions**.

The **Available actions** section presents a table that displays the actions defined in the **Actions** sub-tab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the Oxygen XML Editor plugin

toolbar when you work with a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the **Add as sibling** button. To add an action in this section as a child of the currently selected action use the **Add as child** button.

The following actions are available in the **Current actions** section:

Edit
Edits an item.

Remove
Removes an item.

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** document dialog box.

Use the **Add**, **Edit**, and **Delete** button to add, configure, or remove templates. The **Templates directory** text field specifies the path to the directory of the template. It is recommended to use relative paths since absolute paths will make it difficult to share the templates with others.

You can also use wildcards in the paths for the template directories.

For example, using the following wildcard in the path would add all the template folders found inside the `templates` directory:

```
${frameworkDir}/templates/*
```

The Catalogs Tab

The **Catalogs** tab specifies a list of *XML catalogs* which are added to all the catalogs that Oxygen XML Editor plugin uses to resolve resources.

The Transformation Tab

In the **Transformation** tab you configure the transformation scenarios associated with the framework you are editing. These are the transformation scenarios that are presented in the **Configure Transformation Scenarios** dialog box as associated with the type of the edited document.

You can set one or more of the scenarios from the **Transformation** tab as default. The scenarios set here as default are rendered bold in the **Configure Transformation Scenarios** dialog box and are also displayed on the tooltip of the **Apply transformation Scenario(s)**.

The **Transformation** tab offers the following options:

New

Opens the **New scenario** dialog box allowing you to create a new transformation scenario.

Edit

Opens the **Edit scenario** dialog box allowing you to edit the properties of the currently selected transformation scenario.

Delete

Deletes the currently selected transformation scenario.

Import scenarios

Imports transformation scenarios.

Export selected scenarios

Export transformation scenarios.

Move Up

Moves the selection to the previous scenario.

Move Down

Moves the selection to the next scenario.

The Validation Tab

In the **Validation** tab you configure the validation scenarios associated with the framework you are editing. These are the validation scenarios that are presented in the **Configure Validation Scenarios** dialog box as associated with the type of the edited document.

You can set one or more of the scenarios from the **Validation** tab as default. The scenarios set here as default are rendered bold in the **Configure Transformation Scenarios** dialog box and are also displayed on the tooltip of the **Apply transformation Scenario(s)** button.

The **Validation** tab offers the following options:

New

Opens the **New scenario** dialog box allowing you to create a new validation scenario.

Edit

Opens the **Edit scenario** dialog box allowing you to edit the properties of the currently selected validation scenario.

Delete

Deletes the currently selected validation scenario.

Import scenarios

Imports transformation scenarios.

Export selected scenarios

Export transformation scenarios.

Move Up

Moves the selection to the previous scenario.

Move Down

Moves the selection to the next scenario.

The Extensions Tab

The **Extension** tab specifies implementations of Java interfaces used to provide advanced functionality to the document type.

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.

Editor Preferences

Oxygen XML Editor plugin lets you configure how the editor appears. To configure the appearance of the text editor, *open the **Preferences** dialog box* and go to **Editor** or right click in the editor window and choose **Preferences**.

The following options are available:

- **Editor background** - Sets the background color for both text editor and Diff Files editors.
- **Completion proposal background** - Sets the background color of the content completion window.
- **Completion proposal foreground** - Sets the foreground color of the content completion window.
- **Documentation window background** - Sets the background color of the documentation of elements suggested by the content completion assistant.
- **Documentation window foreground** - Sets the foreground color for the documentation of elements suggested by the content completion assistant.
- **Display quick-assist and quick-fix side hints** - Displays the Quick Assist and Quick Fix icon in the editor's left side line number stripe. Works both in the **Text** and **Author** edit modes.
- **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 Editor plugin disables the **Highlight current line** option.

- **Highlight matching tag** - If you place the cursor on a start or end tag, Oxygen XML Editor plugin highlights the corresponding member of the pair. You can also customize the highlight color.
- **Beep on operation finished** - Oxygen XML Editor plugin emits a short beep when a validate, check well-formedness, or transform action has ended;

  Note: When the validation or the transformation process of a document is successful, the beep signal has a higher audio frequency, as opposed to when the validation fails, and the beep signal has a lower audio frequency. On the Windows platform, for other operations, the default system sound (`Asterisk`) is used . You can configure it by changing the sound theme.

- **Minimum fold range** - You can specify the minimum number of lines in a block for which the folding support becomes active. If you modify this value, the change takes effect next time you open / reopen the editor.

Edit modes Preferences

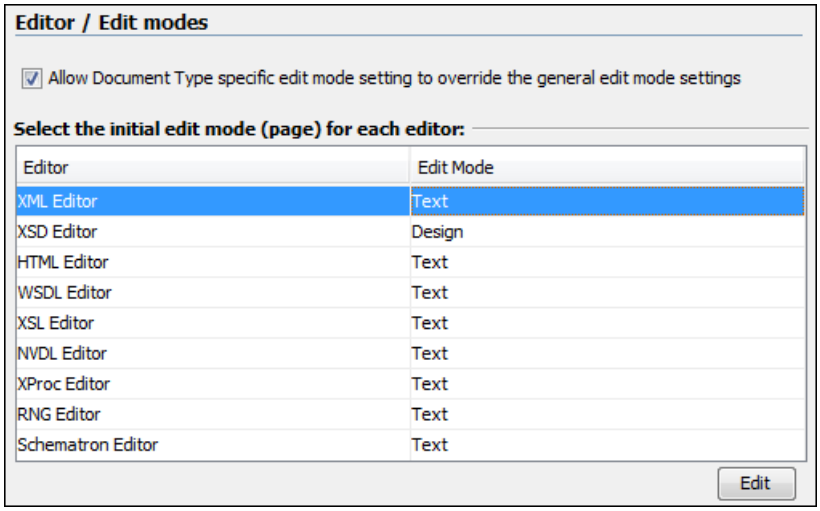
Oxygen XML Editor plugin lets you configure which *edit mode* a file is opened in the first time it is opened. This setting only affects the first time a file is opened. The current editing mode of each file is saved when the file is closed and restored the next time it is opened. To configure the **Edit modes** options, *open the **Preferences** dialog box* and go to **Editor** > **Edit modes** .

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*** *configuration dialog box* overrides the general edit mode setting from the table below.

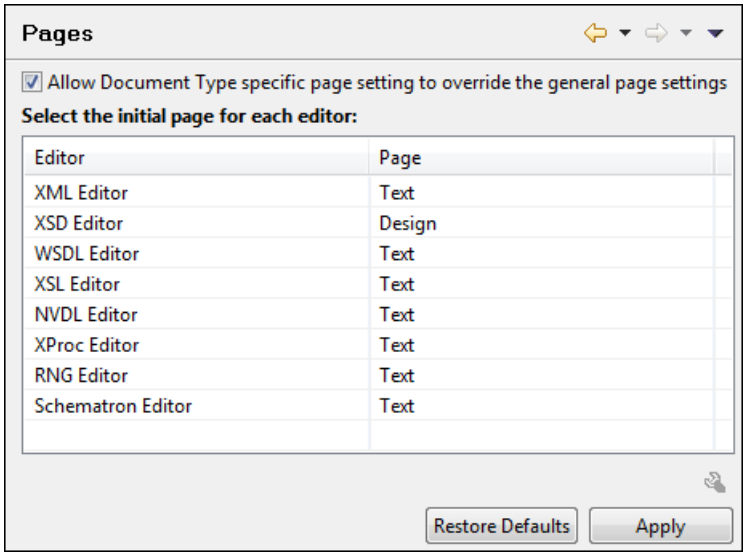
The initial edit mode can be one of the following:

- ***Text***
- ***Author***
- ***Grid***
- **Design** (available only for the W3C XML Schema editor).

The Oxygen XML Editor plugin Edit Modes Preferences Page



The Oxygen XML Editor plugin Edit Modes Preferences Page



Grid Preferences

Oxygen XML Editor plugin provides a *Grid view* of an XML document. To configure the **Grid** options, *open the* ***Preferences*** *dialog box* and go to **Editor** > **Edit modes** > **Grid**.

The following options are available:

- **Compact representation** - If selected, the *compact representation* of the grid is used: a child element is displayed beside the parent element. In the *non-compact representation*, a child element is nested below the parent.

- **Format and indent when passing from grid to text or on save** - If selected, the content of the document is *formatted and indented* each time you switch from the **Grid** view to the **Text** view.
- **Default column width (characters)** - Sets the default width in characters of a table column of the grid. A column can hold:
  - Element names
  - Element text content
  - Attribute names
  - Attribute values

  If the total width of the grid structure is too large you can resize any column by dragging the column margins with the mouse pointer, but the change is not persistent. To make it persistent, set the new column width with this option.
- **Active cell color** - Sets the background color for the active cell of the grid. There is only one active cell at a time. The keyboard input always goes to the active cell and the selection always contains it.
- **Selection color** - Background color for the selected cells of the grid except the active cell.
- **Border color** - The color used for the lines that separate the grid cells.
- **Background color** - The background color of grid cells that are not selected.
- **Foreground color** - The text color of the information displayed in the grid cells.
- **Row header colors - Background color** - The background color of row headers that are not selected.
- **Row header colors - Active cell color** - The background color of the row header cell that is currently active.
- **Row header colors - Selection color** - The background color of the header cells corresponding to the currently selected rows.
- **Column header colors - Background color** - The background color of column headers that are not selected.
- **Column header colors - Active cell color** - The background color of the column header cell that is currently active.
- **Column header colors - Selection color** - The background color of the header cells corresponding to the currently selected columns.

The column headers are painted with two color gradients, one for the upper 1/3 part of the header and the other for the lower 2/3 part. The start and end colors of the first gradient are set with the first two color buttons. The start and end colors of the second gradient are set with the last two color buttons.

Author Preferences

Oxygen XML Editor plugin provides an ***Author** mode* editor that provides a configurable graphical interface for editing XML documents. To configure the options for the **Author** mode, *open the **Preferences** dialog box* 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 currently 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 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.

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 gray box and displays the element name.

Show Author layout messages

When this option is selected, all errors reported while rendering the document in **Author** mode are presented in the **Errors** view at the bottom of the editor.

Display referenced content (entities, XInclude, DITA conref, etc.)

When enabled, the references (such as entities, XInclude, DITA conrefs) also display the content of the resources they reference. If you toggle this option while editing, you need to reload the file for the modification to take effect.

Images Section

The following options in regards to images in **Author** mode are available in this section:

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 Editor plugin may consume all available memory, throwing an *OutOfMemory error*. To resolve this, increase the available memory limit. and restart the application.

Format and Indent Section

In this section you can configure options in regards to the formatting and indenting that is applied when a document is saved in **Author** mode, or when switching the editing mode (from **Text** to **Author**, and vice versa). The following options are available:

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 also applies to the DITA maps opened 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.

Also apply 'Format and Indent' action as in 'Text' edit mode

If this 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.

Compatibility with other tools

Use this option to control how line breaks are handled when a document is serialized. This will help to obtain better compatibility with other tools. You can choose between the following:

- **None** - The default formatting is used.
- **Do not break lines, do not indent** - Choose this option to avoid breaking lines after element start or end tags and indenting will not be used.

  Note: New lines that are added by the user in elements where the `xml:space` attribute is set to `preserve` (such as `pre` elements in HTML, or `codeblock` elements in DITA) are still inserted. Also, selecting this option automatically disables the **Also apply 'Format and Indent' action as in 'Text' edit mode** option, since the formatting from **Text** mode does not take the CSS styles into account.

- **Break lines only after elements displayed as blocks, do not indent** - Choose this option to instruct Oxygen XML Editor plugin to insert new lines only after elements that have a CSS display property set to anything other than `inline` or `none` (for example, `block`, `list-item`, `table`, etc.) and indenting will not be used. When selecting this option, the formatting is dictated by the CSS.

Note: New lines that are added by the user in elements where the `xml:space` attribute is set to `preserve` (such as `pre` elements in HTML, or `codeblock` elements in DITA) are still inserted. Also, selecting this option automatically disables the **Also apply 'Format and Indent' action as in 'Text' edit mode** option, since the formatting from **Text** mode does not take the CSS styles into account.

Tags Section

In this section you can configure the following options in regards to tags that are displayed in **Author** mode:

Tags display mode

Sets the default display mode for element tags presented in **Author** mode. You can choose between the following:

- **Full Tags with Attributes** - All XML tags are displayed, with attribute names and values, making it easier to transition from a Text-based editing to **Author** mode editing.
- **Full Tags** - All XML tags are displayed, but without attributes.
- **Block Tags** - The XML tags that enclose block elements are displayed in full. Compact tags (no element names) are displayed for inline elements.
- **Inline Tags** - The XML tags that enclose inline elements are displayed in full. Block tags are not displayed.
- **Partial Tags** - Partial tags (no names) are displayed 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** mode tags background color.

Tags foreground color

Sets the **Author** mode tags foreground color.

Tags font

Allows you to change the font used to display tags text in the **Author** visual editing mode. The *[Default]* font is computed based on the setting of the ***Author default font*** *option*.

Compact tag layout

When you deselect this option, the **Author** mode displays the tags in a more decompressed layout, where block tags are displayed on separate lines.

Whitespaces Section

The following option is available in this section:

Foreground color

Sets the foreground color of the white spaces in the **Author** mode. To enable this option, *open the* ***Preferences*** *dialog box*, go to **Text Editors** and select **Show whitespaces characters**.

Configure annotation tooltip

Click this link to open the *Annotations Preferences* on page 944 page.

For advanced Author configuration see the Document Type Association settings

Click this link to open the ***Document Type Association*** *preferences page*.

Cursor Navigation Preferences

Oxygen XML Editor plugin allows you to configure the appearance of the cursor in the ***Author*** *mode* editor. To set cursor navigation preferences, *open the* ***Preferences*** *dialog box* and go to **Author** > **Cursor Navigation**. The following options are available:

- **Highlight elements near cursor** - When this option is selected, the element containing the cursor is highlighted. You can use the color picker to choose the color of the highlight.
- **Show cursor position tooltip** - Oxygen XML Editor plugin uses *tool tips in* ***Author*** *mode to indicate the position of the cursor 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 cursor position.

- **Show location tooltip on mouse move** - When this option is selected, Oxygen XML Editor plugin 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 cursor is placed within each of the underlying XML elements between two blocks of text. (The cursor 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 cursor 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 cursor does not stop at these positions, but jumps from one text line to another, similar to how the cursor behaves in a word processor.
- **Arrow keys move the cursor 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 cursor in the reading direction. When this option is not selected, pressing the right arrow will simply move the cursor to the right, regardless of the text direction.

Schema-Aware Preferences

Oxygen XML Editor plugin 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 box* 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 *settings* for **Preserve space elements**, **Default space elements**, and **Mixed content elements**. 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*) . Note that the schema information takes precedence.

- **Indent blocks-only content**

  To avoid accidentally introducing inappropriate white space around inline elements, Oxygen XML Editor plugin does not normally apply indenting to the source of an element with mixed content. If this option is selected, Oxygen XML Editor plugin will apply indenting to the source of mixed content elements that only contain block elements.

- **Schema Aware Editing**

  This setting determines how Oxygen XML Editor plugin will use the schema of a document to control the behavior of the **Author** mode.

  - **On** - Enables all schema-aware editing options.
  - **Off** - Disables all schema-aware editing options.
  - **Custom** - Allows you to select custom 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 Editor plugin 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 a 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**

        A 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 are:

      - **Smart paste and drag and drop**

If the content inserted by a paste or drop action is not valid at the cursor position, according to the schema, Oxygen XML Editor plugin tries to find an appropriate insert position. The 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**

  If this option is enabled and the **Smart paste and drag and drop** option is disabled, Oxygen XML Editor plugin 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 typed characters are not allowed in the element at the cursor position, but the previous element does allow text, then a similar element will be inserted, along with your content.

  - **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 Editor plugin 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 are:

  - **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* for external content.

Review Preferences

Oxygen XML Editor plugin lets you *enter review comments and track changes* in your documents. The Review preferences control how the Oxygen XML Editor plugin review features work. To configure the **Review** options, *open the **Preferences** dialog box* 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 Editor plugin uses the system user name.
- **Initial State** - Specifies whether or not **Track Changes** is enabled when you open a document. You may have some opened documents in which track changes is enabled and others in which it is disabled. You can choose between the following options:
  - **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 Editor plugin 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 Editor plugin 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 Editor plugin 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 Editor plugin can display callouts for *review items such as comments and tracked changes*. To set review callouts preferences, *open the **Preferences** dialog box* and go to **Editor** > **Edit modes** > **Author** > **Review** > **Callouts**.

The options are as follows:

- **Comments** - If selected, callouts are displayed for comments. This option is enabled by default.
- **Track Changes deletion** - If selected, callouts are displayed for deletions.
  - **Show deleted content in callout** - If selected, the deleted content is displayed in the callout.
- **Track Changes insertion** - If selected, callouts are displayed for insertions.
  - **Show inserted content in callout** - If selected, the inserted content is displayed in the callout.
- **Show review time** - When selected, the date and time of a change are displayed 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 Editor plugin lets you configure how *profiling and conditional text* is displayed in **Author** mode. 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 box* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text**.

Note: Note the following when configuring this setting:

- This setting is used to define how profiled elements are treated in **Author** mode. 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 box.

This preferences page contains the following options:

- **Import from DITAVAL** - This 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 Editor plugin displays a dialog box that contains two tables. The first one previews the imported attributes and the second one previews the already defined attributes. You can choose to either keep the existing attributes or replace them with the imported ones.

  Note: When importing profiling attributes from DITAVAL files, Oxygen XML Editor plugin automatically creates condition sets based on these files.

- **Profiling Attributes** section - Allows you to specify a set of allowable value for each profiling or conditional attribute. You can use the **New** button at the bottom of the table *to add profiling attributes*, the **Edit** button to edit existing ones, or the **Delete** button to delete entries from the table. Use the **Up** and **Down** buttons to change the priority of the entries. If you have multiple entries with identical names that match the same document type, Oxygen XML Editor plugin uses the one that is positioned highest in the table.
- **Profiling Condition Sets** section - Allows you to specify a specific set of profiling attributes to be used to specify a particular build configuration for your content. You can use the **New** button at the bottom of the table *to add condition sets*, the **Edit** button to edit existing ones, or the **Delete** button to delete entries from the table. Use the **Up**

and **Down** buttons to change the priority of the entries. If you have multiple entries with identical names that match the same document type, Oxygen XML Editor plugin uses the one that is positioned highest in the table.

Colors and Styles Preferences

Oxygen XML Editor plugin 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 box* 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 Editor plugin 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 box 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 previously defined style, use one of the following actions:

- Double-click the style in the styles table to open the **Edit Profiling Style** dialog box.
- Select the style in the styles table and press **Edit** to open the **Edit Profiling Style** dialog box.

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 Editor plugin 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 box* and go to **Editor** > **Edit modes** > **Author** > **Profiling/Conditional Text** > **Attributes Rendering**. When the ***Show Profiling Attributes*** option is enabled, the **Author** mode 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 displayed with their values. If unchecked, only the values are displayed.
- **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.

MathML Preferences

Oxygen XML Editor plugin allows you to *edit MathML* equations and displays the results in a preview window. For a more specialized *MathML* editor, you can *install Design Science MathFlow*, which is a commercial product that requires a separate license.

To configure the *MathML* editor or to enter your *MathFlow* license information, *open the **Preferences** dialog box* and go to **Editor** > **Edit Modes / Pages** > **Author** > **MathML**. You can configure the following parameters:

- **Equation minimum font size** - The minimum size of the font used for rendering mathematical symbols when editing in the **Author** mode.

The following options can be configured for *MathFlow*:

- **MathFlow installation directory** - The installation folder for the *MathFlow* Components product (*MathFlow* SDK).
- **MathFlow license file** - The license for*MathFlow* Components product (*MathFlow* SDK).
- **MathFlow preferred editor** - A *MathML* formula can be edited in one of three editors of *MathFlow* Components product(*MathFlow* SDK):
  - **Structure Editor** - (default selection) targets professional XML workflow users
  - **Style Editor** - tailored to the needs of content authors
  - **Simple Editor** - designed for applications where end-users can enter mathematical equations without prior training and only the meaning of the math matters
- **Save special characters** - When editing mathematical expressions the special characters can be saved in the XML file:
  - **As entity names** - saves the characters in `&name;` format. It refers to a character by the name of the entity which has the desired character as its replacement text. For example, the Greek *Omega* character is saved as `&Omega;`.
  - **As character entities** (default selection) - saves the characters in a hexadecimal value, using the `&#xNNN` format. For example, the Greek *Omega* character is saved as `&#x3a9;`.
  - **As character values** - saves the characters as the actual symbol. For example, the Greek *Omega* character is saved as Ω.

More documentation is available on the *Design Science MathFlow* website.

AutoCorrect Preferences

Oxygen XML Editor plugin includes an option to automatically correct misspelled words as you type in **Author** mode. To enable and configure this feature, *open the **Preferences** dialog box* and go to **Editor** > **Edit Modes** > **Author** > **AutoCorrect**.

The following options are available:

- **Enable AutoCorrect** - This option is enabled by default. When enabled, while editing in **Author** mode, if you type anything that is listed in the **Replace** column of the Replacements table displayed in this preferences page, Oxygen XML Editor plugin will automatically replace it with the value listed in the **With** column.

- **Use additional suggestions from the spell checker** - If enabled, in addition to anything listed in the Replacements table displayed in this preferences page, Oxygen XML Editor plugin will also use suggestions from the Spell Checker to automatically correct misspelled words. Suggestions from the Spell Checker will only be used if the misspelled word is not found in the Replacements table.

Note: The *AutoCorrect* feature shares the same options configured in the ***Language options*** and ***Ignore elements*** sections in the **Spell Check** preferences page.

Replacements Table

The *AutoCorrect* feature uses the Replacements table to automatically replace anything that is listed in the **Replace** column with the value listed in the **With** column for each language. You can specify the language in the **Replacements for language** drop-down menu, and for each language, you can configure the items listed in the table. The language selected in this page is not the language that will be used by the *AutoCorrect* feature. It is simply the language for which you are configuring the Replacements table.

You can double-click on cells in either column to edit the listed items. Use the **Add** button to insert new items and the **Remove** button to delete rows from the table.

Note: Any changes, additions, or deletions you make to this table are saved to a path that is specified in the *AutoCorrect **Dictionaries** preferences page*.

Smart Quotes Section

You can also choose to automatically convert double and single quotes to a quotation characters of your choice by using the following options in the **Smart quotes** section:

- **Replace "Single quotes"** - Replaces single quotes with the quotation symbols you select with the **Start quote** and **End quote** buttons.
- **Replace "Double quotes"** - Replaces double quotes with the quotation symbols you select with the **Start quote** and **End quote** buttons.

Global and Project Options Section

Selecting **Project Options** in this preferences page will only save your selections in **Enable AutoCorrect**, **Use additional suggestions from the spell checker**, and the options in the **Smart quotes** section. Changes to the Replacements table are not saved in this page. To save changes to the Replacements table at project level you need to specify a custom location in *the **User-defined replacements** section of the AutoCorrect Dictionaries preferences page* and select **Project Options** from that preferences page instead.

Restore Defaults - Restores the options in this preferences page to their default values and also **deletes any changes you have made to the Replacements table**.

AutoCorrect Dictionaries Preferences

To set the Dictionaries preferences for the *AutoCorrect* feature, *open the **Preferences** dialog box* and go to **Editor** > **Edit Modes** > **Author** > **AutoCorrect** > **Dictionaries**. This page allows you to specify the location of the dictionaries that Oxygen XML Editor plugin uses for the *AutoCorrect* feature and the location for saving user-defined replacements.

The following options are available in this preferences page:

- **Dictionaries default folder** - Displays the default location where the dictionaries that Oxygen XML Editor plugin uses for the *AutoCorrect* feature are stored.
- **Include dictionaries from** - Enable this option if you want to specify an additional location for the dictionaries that Oxygen XML Editor plugin will use for the *AutoCorrect* feature.

Note: The *AutoCorrect* feature takes into account dictionaries collected both from the default and custom locations and multiple dictionaries from the same language are merged into a generic dictionary (for example, `en_UK.dat` from the default location is merged with `en_US.dat` from a custom location, and the result is that a third file is created for a generic dictionary called `en.dat`). However, if there is already a generic dictionary (for example, `en.dat`) saved in either the default or custom location, the other specific dictionaries

(for example, `en_UK.dat` and `en_US.dat`) will not be merged and the existing generic dictionary will simply be used. Also, if the additional location contains a file with the same name as one from the default location, the file in the additional location takes precedence over the file from the default location. The user-defined replacements are never merged.

- **Save user-defined replacements in the following location** - Specifies the target where added, edited, or deleted replacements are saved. By default, the target is the application preferences folder, but you can also choose a custom location.

  Tip: To save changes to *the Replacement table (in the AutoCorrect preferences page)* at project level, select a custom location for the **User-defined replacements** and select **Project Options** at the bottom of the page.

Schema Design Preferences

Oxygen XML Editor plugin provides a *graphical schema design editor* to make editing XML schemas easier. To configure the **Schema Design** options, *open the **Preferences** dialog box* and go to **Editor** > **Edit modes** > **Schema Design**

The following options are available in the **Schema Design** preferences page:

- **Show annotation in the diagram** - When selected, Oxygen XML Editor plugin displays the content of `xs:documentation` elements in the XML Schema **Design** view.
- **When trying to edit components from another schema** - The schema diagram editor will combine schemas imported by the current schema file into a single schema diagram. You can choose what happens if you try to edit a component from an imported schema. The options are:
  - **Always go to its definition** - Oxygen XML Editor plugin opens the imported schema file so that you can edit it.
  - **Never go to its definition** - The imported schema file is not opened. The definition cannot be edited in place.
  - **Always ask** - Oxygen XML Editor plugin asks if you want to open the imported schema file.

Properties

Oxygen XML Editor plugin lets you control which properties to display for XML Schema components in the *XML Schema **Design** view*. To configure the schema design properties displayed, *open the **Preferences** dialog box* and go to **Editor** > **Edit modes** > **Schema Design** > **Properties**.

This preferences page contains the following:

Show additional properties in the diagram

If this option is selected, the properties selected in the property table are shown in the XML Schema **Design** mode. This option is selected by default.

Properties Table

Show - Use this column in the table to select the properties that you want to be displayed in the XML Schema **Design** mode.

Only if specified - Use this column to select if you want the property to be displayed only if it is defined in the schema.

Text Diagram Preferences

For certain XML languages, Oxygen XML Editor plugin provides a diagram view as part of the text mode editor. To configure the **Diagram** preferences, *open the **Preferences** dialog box* and go to **Editor** > **Edit modes / Pages** > **Text Diagram**.

The following options are available:

- **Show Full Model XML Schema diagram** - When this option is selected, the **Text** mode editor for XML Schemas is displayed with a split screen view that shows a diagram of the schema structure. This may be useful for seeing the effect of schema changes you make in text view. For editing a the schema using diagram rather than text, use the *schema **Design** view*.

Note: When handling very large schemas, displaying the schema diagram might affect the performance of your system. In such cases, disabling the schema diagram view improves the speed of navigation through the edited schema.

- **Enable Relax NG diagram and related views** - Enables the Relax NG schema diagram and synchronization with the related views (**Attributes**, **Model**, **Elements**, **Outline**).
  - **Show Relax NG diagram** - Displays the Relax NG schema diagram in **Full Model View** and **Logical Model View**.
- **Enable NVDL diagram and related views** - Enables the NVDL schema diagram and synchronization with the related views (**Attributes**, **Model**, **Elements**, **Outline**).
  - **Show NVDL diagram** - Displays the NVDL schema diagram in **Full Model View** and **Logical Model View**.
- **Location relative to editor** - Allows you to specify the location of the schema diagram panel relative to the diagram **Text** editor.

Format Preferences

This preferences page contains various formatting options which influence editing and formatting both in the **Text** and **Author** modes. To control additional options specifically for the *Author mode* editor, see *Whitespace Handling in Author Mode* on page 80.

Note: These settings apply to the formatting of source documents. The formatting of output documents is determined by the *transformation scenarios that create them*.

The following options are available:

- **Detect indent on open** - Oxygen XML Editor plugin detects how a document is indented when it is opened. Oxygen XML Editor plugin uses a heuristic method of detection by computing a weighted average indent value from the initial document content. You can disable this setting if the detected value does not work for your particular case and you want to use a fixed-size indent for all the edited documents.

  Tip: If you want to minimize the formatting differences created by the **Format and Indent** operation in a document edited in the **Text** edited mode, make sure that both the **Detect indent on open** and **Detect line width on open** options are enabled.

  - **Use zero-indent, if detected** - By default, if no indent was detected in the document, the fixed-size indent is used. Enable this option if all your document have no indentation and you want to keep them that way.
- **Indent with tabs** - If selected, indents are created using tab characters. If unchecked, lines are indented using space characters.
- **Indent size** - A fixed number of spaces used for indenting a line.
- **Hard line wrap (Limit to "Line width - Format and Indent")** - If selected, when typing content in the **Text** editing mode when the maximum line width is reached, a line break is automatically inserted.
- **Indent on enter** - If disabled, when you press the `Enter` key to insert a line break in the **Text**editing mode, no indentation will be added to the new line.
- **Enable smart enter** - If selected, when you press the `Enter` key between a start and an end XML tag in the **Text** editing mode, the cursor is placed in an indented position on the empty line formed between the start and end tag.
- **Detect line width on open** - When selected, Oxygen XML Editor plugin detects the line width automatically when the document is opened.
- **Format and indent the document on open** - When selected, an XML document is formatted and indented before opening it in Oxygen XML Editor plugin.
- **Line width - Format and Indent** - Defines the number of characters after which the **Format and Indent** (pretty-print) action performs hard line wrapping. For example, if set to 100, after a **Format and Indent** action, the longest line will have at most 100 characters.

Note: In order to avoid having an indent that is longer than the line width setting and without having sufficient space available for the text content, the indent limit is actually set at half the value of the **Line width - Format and Indent** setting. The remaining space is reserved for text.

- **Clear undo buffer before Format and Indent** - The **Format and Indent** operation can be undone, but if used intensively, a considerable amount of the memory allocated for Oxygen XML Editor plugin will be used for storing the undo states. If this option is selected, Oxygen XML Editor plugin empties the undo buffer before doing a **Format and Indent** operation. This means you will not be able to undo any changes you made before the format and indent operation. Select this option if you encounter out of memory problems (**OutOfMemoryError**) when performing the **Format and Indent** operation.

The indent size and line width limit settings are used in various places in the application:

- When the **Format and Indent** action is used in the **Text** editing mode.
- When you press `ENTER` in the **Text** editing mode to break a line.
- When editing in the **Text** mode with **Hard line wrap** enabled.
- When the XML is serialized by saving content in the **Author** editing mode.

To watch our video demonstration about the formatting options offered by Oxygen XML Editor plugin, go to *http://oxygenxml.com/demo/Autodetect_Formating.html*.

XML Formatting Preferences

To configure the **XML** Formatting options, *open the **Preferences** dialog box* and go to **Editor** > **Format** > **XML**.

The following options are available:

Format Section

This section includes the following drop down boxes:

Preserve empty lines

The **Format and Indent** operation preserves all empty lines found in the document.

Preserve text as it is

The **Format and Indent** operation preserves text content as it is, without removing or adding any white space.

Preserve line breaks in attributes

Line breaks found in attribute values are preserved.

Note: When this option is enabled, the **Break long attributes** option is automatically disabled.

Break long attributes

The **Format and Indent** operation breaks long attribute values.

Indent inline elements

The *inline elements* are indented on separate lines if they are preceded by white spaces and they follow another element start or end tag. For 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 (for example, `<a atr1="v1"></a>`). When not enabled, the same operation represents an empty element in a more compact form (`<a atr1="v1"/>`).

Sort attributes

The **Format and Indent** operation sorts the attributes of an element lexicographically.

Add space before slash in empty elements

Inserts a space character before the trailing / and > of empty elements.

Break line before an attribute name

The **Format and Indent** operation breaks the line before the attribute name.

Element Spacing Section

This section controls how the application handles whitespaces found in XML content. You can **Add** or **Remove** element names or simplified XPath expressions in the various tabs.

Note: The XPath expressions can accept the following attribute conditions (default attribute values are not taken into account):

- *element[@attr]* - Matches all instances of the specified element that include the specified attribute.
- *element[not(@attr)]* - Matches all instances of the specified element that do not include the specified attribute.
- *element[@attr = "value"]* - Matches all instances of the specified element that include the specified attribute with the given value.
- *element[@attr != "value"]* - Matches all instances of the specified element that include the specified attribute and its value is different than the one given.

The tabs are as follows:

Preserve space

List of elements for which the **Format and Indent** operation preserves the whitespaces (such as blanks, tabs, and newlines). The elements can be specified by name or by simplified XPath expressions:

- `elementName`
- `//elementName`
- `/elementName1/elementName2/elementName3`
- `//xs:localName`

Note: The namespace prefixes (such as `xs`) are treated as part of the element name without taking its binding to a namespace into account.

Default space

List of elements for which the content is normalized (multiple contiguous whitespaces are replaced by a single space), before applying the **Format and Indent** operation.

Mixed content

The elements from this list are treated as mixed content when applying the **Format and Indent** operation. The lines are split only when whitespaces are encountered.

Line break

List of elements for which line breaks will be inserted, regardless of their content. You can choose to break the line *before* the element, *after*, or both.

Schema aware format and indent

The **Format and Indent** operation takes the schema information into account with regards to the *space preserve*, *mixed*, or *element only* properties of an element.

Indent Section

Includes the following options:

Indent (when typing) in preserve space elements

Normally, the *Preserve space* elements (identified by the `xml:space` attribute set to `preserve` or by their presence in the **Preserve space** elements list) are ignored by the **Format and Indent** operation. When this option is enabled and you edit one of these elements, its content is formatted.

Indent on paste - sections with number of lines less than 300

When you paste a chunk of text that has less than 300 lines, the inserted content is indented. To keep the original indent style of the document you copy content from, disable this option.

Whitespaces Preferences

When Oxygen XML Editor plugin formats and indents XML documents, a whitespace normalization process is applied, thus replacing whitespace sequences with single space characters. Oxygen XML Editor plugin allows you to configure which Unicode characters are treated as spaces during the normalization process.

To configure the **Whitespace** preferences, *open the **Preferences** dialog box* and go to **Editor** > **Format** > **XML** > **Whitespaces**.

This table lists the Unicode whitespace characters. Check any that you want to have treated as whitespace when formatting and indenting an XML document.

The whitespaces are normalized when:

- The **Format and Indent** action is applied on an XML document.
- You switch from **Text** mode to **Author** mode.
- You switch from **Author** mode to **Text** mode.

Note: The whitespace normalization process replaces any sequence of characters declared as whitespaces in the **Whitespaces** table with a single space character (`U+0020`). If you want to be sure that a certain whitespace character will not be removed in the normalization process, deselect it in the **Whitespaces** table.

Important: The characters with the codes `U+0009` (TAB), `U+000A` (LF), `U+000D` (CR) and `U+0020` (SPACE) are always considered to be whitespace characters and cannot be deselected.

XQuery Formatting Preferences

To configure the **XQuery** Formatting options, *open the **Preferences** dialog box* and go to **Editor** > **Format** > **XQuery**.

The following options are available:

- **Preserve line breaks** - All initial line breaks are preserved.
- **Break line before an attribute name** - Each attribute of an XML element is written on a new line and properly indented.

XPath Formatting Preferences

To configure the **XPath** Formatting options, *open the **Preferences** dialog box* and go to **Editor** > **Format** > **XPath**.

The following option is available:

- **Format XPath code embedded in XSLT, XSD and Schematron files** - If enabled, the **Format and Indent** action applied on a XSD, XSLT, or Schematron document will perform an XPath-specific formatting on the values of the attributes that accept XPath expressions.

  Note: For XSLT documents, the formatting is not applied to *attribute value templates*.

CSS Properties Formatting Preferences

Oxygen XML Editor plugin can format and indent your CSS files. To configure the **CSS** formatting options, *open the **Preferences** dialog box* and go to **Editor** > **Format** > **CSS**.

The following options control how your CSS files are formatted and indented:

Class body on new line

If enabled, the *class* body (including the curly brackets) is placed on a new line. Disabled by default.

Indent class content

When enabled, the *class* content is indented. Enabled by default.

Add space before the value of a CSS property

When enabled, whitespaces are added between the : (colon) and the value of a style property. Enabled by default.

Add new line between classes

If enabled, an empty line is added between two classes. Disabled by default.

Preserve empty lines

When enabled, the empty lines from the CSS content are preserved. Enabled by default.

Allow formatting embedded CSS

When enabled, CSS content that is embedded in XML is also formatted when the XML content is formatted. Enabled by default.

JavaScript Properties Formatting Preferences

To configure the **JavaScript** format options, *open the **Preferences** dialog box* and go to **Editor** > **Format** > **JavaScript**.

The following options control the behavior of the **Format and Indent** action:

- **Start curly brace on new line** - Opening curly braces start on a new line.
- **Preserve empty lines** - Empty lines in the JavaScript code are preserved. This option is enabled by default.
- **Allow formatting embedded JavaScript** - Applied only to XHTML documents, this option allows Oxygen XML Editor plugin to format embedded JavaScript code, taking precedence over the *Schema aware format and indent* option. This option is enabled by default.

Content Completion Preferences

Oxygen XML Editor plugin provides a ***Content Completion Assistant*** that list available options at any point in a document and can auto-complete structures, elements, and attributes. These options control how the **Content Completion Assistant** works.

To configure the **Content Completion** preferences, *open the **Preferences** dialog box* and go to **Editor** > **Content Completion**.

The following options are available:

- **Auto close the last opened tag** - Oxygen XML Editor plugin closes the last open tag when you type `</`.
- **Automatically rename/delete/comment matching tag** - If you rename, delete, or comment out a start tag, Oxygen XML Editor plugin automatically renames, deletes, or comments out the matching end tag.

  Note: If you select **Toggle comment** for multiple starting tags and the matching end tags are on the same line as other start tags, the end tags are not commented.

- **Use content completion** - Turns content completion on or off.
- **Close the inserted element** - When you choose an entry from the **Content Completion Assistant** list of proposals, Oxygen XML Editor plugin inserts both start and end tags.
  - **If it has no matching tag** - The end tag of the inserted element is automatically added only if it is not already present in the document.
  - **Add element content** - Oxygen XML Editor plugin inserts the required elements specified in the DTD, XML Schema, or RELAX NG schema that is *associated with the edited XML document*.
    - **Add optional content** - Oxygen XML Editor plugin inserts the optional elements specified in the DTD, XML Schema, or RELAX NG schema.
    - **Add first Choice particle** - Oxygen XML Editor plugin inserts the first **choice** particle specified in the DTD, XML Schema, or RELAX NG schema.

- **Case sensitive search** - When enabled, the search in the content completion assistant window when you type a character is case-sensitive ('a' and 'A' are different characters).

  Note: This option is ignored when the current language itself is not case sensitive. For example, the case is ignored in the CSS language.

- **Cursor position between tags** - When selected, Oxygen XML Editor plugin automatically moves the cursor between start and end tag after inserting the element. This only applies to:
  - Elements with only optional attributes or no attributes at all.
  - Elements with required attributes, but only when the **Insert the required attributes** option is disabled.
- **Show all entities** - Oxygen XML Editor plugin displays a list with all the internal and external entities declared in the current document when you type the start character of an entity reference (for example, &).
- **Insert the required attributes** - Oxygen XML Editor plugin 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 Editor plugin 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 Editor plugin remembers the last inserted items from the **Content Completion Assistant** window. The number of items to be remembered is limited by the **Maximum number of recent items shown** list box. These most frequently used items are displayed on the top of the content completion window their icon is decorated with a small red square.. 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 Editor plugin 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 Editor plugin 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 Editor plugin 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.

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 Editor plugin can display these annotations when offering content completion suggestions. To configure the **Annotations** preferences, *open the **Preferences** dialog box* and go to **Editor** > **Content Completion** > **Annotations**.

The following options are available:

Show annotations in Content Completion Assistant

Oxygen XML Editor plugin displays the schema annotations of an element, attribute, or attribute value currently selected in the **Content Completion Assistant** proposals list.

Show annotations in tooltip

Oxygen XML Editor plugin displays the annotation of elements and attributes as a tooltip when you hover over them with the cursor in the editing area or in the **Elements** view.

Show annotation in HTML format, if possible

This option allows you to view the annotations associated with an element or attribute in HTML format. It is available when editing XML documents that have associated an XML Schema or Relax NG schema. When this option is disabled the annotations are converted and displayed as plain text.

Prefer DTD comments that start with "doc:" as annotations

To address the lack of dedicated annotation support in DTD documents, Oxygen XML Editor plugin recommends prefixing with the `doc:` particle all comments intended to be shown to the developer who writes an XML validated against a DTD schema.

When this option is enabled, Oxygen XML Editor plugin uses the following mechanism to collect annotations:

- If at least one `doc:` comment is found in the entire DTD, only comments of this type are displayed as annotations.
- If no `doc:` comment is found in the entire DTD, all comments are considered annotations and displayed as such.

When the option is disabled, all comments, regardless of their type, are considered annotations and displayed as such.

Use all Relax NG annotations as documentation

When this option is selected, any element outside the Relax NG namespace, that is `http://relaxng.org/ns/structure/1.0`, is considered annotation and is displayed in the annotation window next to the **Content Completion Assistant** window and in the **Model** view. When this option is not selected, only elements from the Relax NG annotations namespace, that is `http://relaxng.org/ns/compatibility/annotations/1.0` are considered annotations.

XSL Preferences

XSL stylesheets are often used to create output in XHTML or XSL-FO. In addition to suggesting content completion options for XSLT stylesheet elements, Oxygen XML Editor plugin can suggest elements from these vocabularies. To configure the XSL content completion options, *open the **Preferences** dialog box* and go to **Editor** > **Content Completion** > **XSL**.

The following options are available:

- **Automatically detect XHTML transitional or Formatting objects** - Detects if the output being generated is XHTML or FO and provides content completion for those vocabularies. Oxygen XML Editor plugin analyzes the namespaces declared in the root element to find an appropriate schema.

  If the detection fails, Oxygen XML Editor plugin uses one of the following options:

  - **None** - The **Content Completion Assistant** suggests only XSLT elements.
  - **XHTML transitional** - The **Content Completion Assistant** includes XHTML transitional elements as substitutes for `xsl:element`.
  - **Formatting objects** - The **Content Completion Assistant** includes Formatting Objects (XSL-FO) elements as substitutes for `xsl:element`.
  - **Custom schema** - If you want content completion hints for a different output vocabulary, enter the path to the schema for that vocabulary here. The supported schema types are DTD, XML Schema, RNG schema, or NVDL schema for inserting elements from the target language of the stylesheet.

- **Documentation schema** - You can choose an additional schema that will be used for documenting XSL stylesheets. Choose between the following:

  - **Build-in schema** - Uses the built-in schema for documentation.
  - **Custom schema** - Allows you to specify a custom schema for documentation. The supported schema types are XSD, RNG, RNC, DTD, and NVDL.

XPath Preferences

Oxygen XML Editor plugin provides content-completion support for XPath expressions. To configure the options for the content completion in XPath expressions, *open the **Preferences** dialog box* and go to **Editor** > **Content Completion** > **XPath**.

The following options are available:

- **Enable content completion for XPath expressions** - Enables *the **Content Completion Assistant** in XPath expressions* that you enter in the `match`, `select`, and `test` XSL attributes and also in the XPath toolbar.

- **Include XPath functions** - When this option is selected, XPath functions are included in the content completion suggestions.
- **Include XSLT functions** - When this option is selected, XSLT functions are included in the content completion suggestions.
- **Include axes** - When this option is selected, XSLT axes are included in the content completion suggestions.

- **Show signatures of XSLT / XPath functions** - Makes the editor indicate the signature of the XPath function located at the cursor position in a tooltip. See the *XPath Tooltip Helper* section for more information.

XSD Preferences

Oxygen XML Editor plugin provides content completion assistance when you are writing an XML Schema (XSD). To configure XSD preferences, *open the **Preferences** dialog box* and go to **Editor** > **Content Completion** > **XSD**. The options in this preferences page define what elements are suggested by the **Content Completion Assistant**, in addition to the ones from the XML Schema (defined by the `xs:annotation/xs:appinfo` elements).

The following options are available:

- **None** - The **Content Completion Assistant** offers only the XML Schema schema information.
- **ISO Schematron** - The **Content Completion Assistant** includes ISO Schematron elements in `xs:appinfo`.
- **Schematron 1.5** - The **Content Completion Assistant** includes Schematron 1.5 elements in `xs:appinfo`.
- **Other** - The **Content Completion Assistant** includes in `xs:appinfo` elements from an XML Schema identified by an URL.

Syntax Highlight Preferences

Oxygen XML Editor plugin supports syntax highlighting of XML in the *Text mode* editor, DTD, Relax NG (XML and Compact Syntax), Java, JavaScript / JSON, PHP, CSS, XQuery, C++, C, Perl, Properties, SQL, Shell and Batch documents.

To configure syntax highlighting, *open the **Preferences** dialog box* and go to **Editor** > **Syntax Highlight**.

To set syntax colors for a language, expand the listing for that language in the top panel to show the list of syntax items for that language. Use the color and style selectors to change how each syntax item is displayed. The results of your changes are displayed in the preview panel. If you do not know the name of the syntax token that you want to configure, click that token in the **Preview** area to select it.

Note: All default color sets come with a high-contrast variant, which is automatically used when you switch to a black-background or white-background high-contrast theme in your Windows operating system settings. The high-contrast theme will not overwrite any default color you set in **Editor** > **Syntax Highlight** preferences page.

The settings for XML documents are used also in XSD, XSL, RNG documents. The **Preview** area has separate tabs for XML, XSD, XSL, RNG.

The **Enable nested syntax highlight** option controls if different content types mixed in the same file (like PHP, JS and CSS scripts inside an HTML file) are highlighted according with the color schemes defined for each content type.

Elements / Attributes by Prefix Preferences

Oxygen XML Editor plugin lets you specify different colors for XML elements and attributes with specific namespace prefixes. To configure the **Elements / Attributes by Prefix** preferences, *open the **Preferences** dialog box* and go to **Editor** > **Syntax Highlight** > **Elements / Attributes by Prefix**.

To change the syntax coloring for a specific namespace prefix, choose the prefix from the list, or add a new one using the **New** button, and use the color and style selectors to set the syntax highlighting style for that namespace prefix.

Note: Syntax highlighting is based on the literal namespace prefix, not the namespace that the prefix is bound to in the document.

If you want only the prefix, and not the whole element or attribute name, to be styled differently, select **Draw only the prefix with a separate color**.

Open / Save Preferences

Oxygen XML Editor plugin lets you control how files are opened and saved. To configure the **Open / Save** options, *open the **Preferences** dialog box* and go to **Editor** > **Open / Save**.

Open

The following options apply to opening files:

- **Format document when longest line exceeds** - Oxygen XML Editor plugin will create line breaks if the characters in a line exceed the specified value. You can choose one of the following:
  - **Always format**
  - **Never format**
  - **Always ask**

Save

The following options apply to saving files:

- **Check errors on save** - If enabled, Oxygen XML Editor plugin runs a validation that checks your document for errors before saving it.
- **Save all files before transformation or validation** - Saves all open files before validating or transforming an XML document. This ensures that any dependencies are resolved when modifying the XML document and its XML Schema.

Performance

The following options cover performance issues when dealing with large files:

- **Clear undo buffer on save** - If selected, Oxygen XML Editor plugin clears its undo buffer when you save a document. Therefore, modifications made prior to saving the document cannot be undone. Select this option if you frequently encounter *out of memory* errors when editing large documents.

Save Hooks Preferences

Oxygen XML Editor plugin includes an option for automatically compiling LESS stylesheets. To set this option, *open the **Preferences** dialog box* and go to **Editor** > **Open / Save** > **Save hooks**.

The following option can be enabled or disabled:

- **Automatically compile LESS to CSS when saving** - If enabled, when you save a LESS file it will automatically be compiled to CSS (disabled by default).

  Important: If this option is enabled, when you save a LESS file, the CSS file that has the same name as the LESS file is overwritten without warning. Make sure all your changes are made in the LESS file. Do not edit the CSS file directly, as your changes might be lost.

Templates Preferences

This page groups the preferences for code templates and document templates:

- *Code Templates*
- *Document Templates*

Code Templates Preferences

Code templates are code fragments that can be inserted at the current editing position. Oxygen XML Editor plugin comes with a set of built-in templates for CSS, LESS, Schematron, XSL, XQuery, and XML Schema document types. You can also define your own code templates and share them with your colleagues using the template export and import functions.

To configure **Code Templates**, *open the **Preferences** dialog box* and go to **Editor** > **Templates** > **Code Templates**.

This preferences page contains a list of all the available code templates (both built-in and custom created ones) and a code preview area. You can disable any code template by deselecting it.

The following actions are available:

New

Opens the **Code template** dialog box that allows you to define a new code template. You can define the following fields:

- **Name** - The name of the code template.
- **Description** - The description of the code template that will appear in the **Code Templates** preferences page and in the tooltip message when selecting it from the **Content Completion Assistant**. HTML markup can be used for better rendering.
- **Associate with** - You can choose to set the code template to be associated with a specific type of editor or for all editor types.
- **Shortcut key** - Allows you to configure a shortcut key that can be used to insert the code template. The `+` character separates keys. If the **Enable platform-independent shortcut keys** checkbox is enabled, the shortcut is platform-independent and the following modifiers are used:
  - `M1` represents the **Command** key on MacOS X, and the **Ctrl** key on other platforms.
  - `M2` represents the **Shift** key.
  - `M3` represents the **Option** key on MacOS X, and the **Alt** key on other platforms.
  - `M4` represents the **Ctrl** key on MacOS X, and is undefined on other platforms.
- **Content** - Text box where you define the content that is used when the code template is inserted.

Edit

Opens the **Code template** dialog box and allows you to edit an existing code template. You can edit the following fields:

- **Description** - The description of the code template that will appear in the **Code Templates** preferences page and in the tooltip message when selecting it from the **Content Completion Assistant**. HTML markup can be used for better rendering.
- **Shortcut key** - Allows you to configure a shortcut key that can be used to insert the code template. The `+` character separates keys. If the **Enable platform-independent shortcut keys** checkbox is enabled, the shortcut is platform-independent and the following modifiers are used:
  - `M1` represents the **Command** key on MacOS X, and the **Ctrl** key on other platforms.
  - `M2` represents the **Shift** key.
  - `M3` represents the **Option** key on MacOS X, and the **Alt** key on other platforms.
  - `M4` represents the **Ctrl** key on MacOS X, and is undefined on other platforms.
- **Content** - Text box where you define the content that is used when the code template is inserted.

Duplicate

Creates a duplicate of the currently selected code template.

Delete

Deletes the currently selected code template. This action is disabled for the built-in code templates.

Export

Exports a file with code templates.

Import

Imports a file with code templates that was created by the **Export** action.

You can use the following *editor variables* when you define a code template in the **Content** text box:

- *${caret}* - The position where the cursor is located. This variable can be used in a code template, in **Author** mode operations, or in a selection plugin.

- *${selection}* - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** mode operations, or in a selection plugin.
- *${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}* - To prompt for values at runtime, use the *ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value'')* editor variable. You can set the following parameters:
  - `'message'` - The displayed message. Note the quotes that enclose the message.
  - `type` - Optional parameter, with one of the following values:

Parameter	
`url`	**Format:** `${ask('message', url, 'default_value')}`
	Description: Input is considered a URL. Oxygen XML Editor plugin checks that the provided URL is valid.
	Example: • `${ask('Input URL', url)}` - The displayed dialog box has the name `Input URL`. The expected input type is URL. • `${ask('Input URL', url, 'http://www.example.com')}` - The displayed dialog box has the name `Input URL`. The expected input type is URL. The input field displays the default value `http://www.example.com`.
`password`	**Format**: *${ask('message', password, 'default')}*
	Description: The input is hidden with bullet characters.
	Example: • `${ask('Input password', password)}` - The displayed dialog box has the name `'Input password'` and the input is hidden with bullet symbols. • `${ask('Input password', password, 'abcd')}` - The displayed dialog box has the name `'Input password'` and the input hidden with bullet symbols. The input field already contains the default `abcd` value.
`generic`	**Format**: *${ask('message', generic, 'default')}*
	Description: The input is considered to be generic text that requires no special handling.
	Example: • `${ask('Hello world!')}` - The dialog box has a `Hello world!` message displayed. • `${ask('Hello world!', generic, 'Hello again!')}` - The dialog box has a `Hello world!` message displayed and the value displayed in the input box is `'Hello again!'`.
`relative_url`	**Format:** `${ask('message', relative_url, 'default')}`
	Description: Input is considered a URL. Oxygen XML Editor plugin tries to make the URL relative to that of the document you are editing. **Note:** If the `$ask` editor variable is expanded in content that is not yet saved (such as an *untitled* file, whose path cannot be determined), then Oxygen XML Editor plugin will transform it into an absolute URL.
	Example: • *${ask('File location', relative_url, 'C:/example.txt')}* - The dialog box has the name `'File location'`. The URL inserted in the input box is made relative to the current edited document location.

Parameter	
`combobox`	**Format:** `${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a drop-down menu. The drop-down menu is populated with the given `rendered_value` values. Choosing such a value will return its associated value (`real_value`). **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The drop-down menu displays the three given operating systems. The associated value will be returned based upon your selection. **Note:** In this example, the default value is indicated by the `osx` key. However, the same result could be obtained if the default value is indicated by `Mac OS X`, like in the following example: `${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'Mac OS X')}` • `${ask('Mobile OS', combobox, ('win':'Windows Mobile';'ios':'iOS';'and':'Android'), 'Android')}`
`editable_combobox`	**Format:** `${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a drop-down menu with editable elements. The drop-down menu is populated with the given `rendered_value` values. Choosing such a value will return its associated real value (`real_value`) or the value inserted when you edit a list entry. **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The drop-down menu displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.
`radio`	**Format:** `${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a series of radio buttons. Each radio button displays a `'rendered_value` and will return an associated `real_value`.

Parameter	
	Note: The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The radio button group allows you to choose between the three operating systems. **Note:** In this example `Mac OS X` is the default selected value and if selected it would return `osx` for the output.

  - `'default-value'` - optional parameter. Provides a default value.

- *${timeStamp}* - Time stamp, that is the current time in Unix format. It can be used for example to save transformation results in different output files on each transform.
- *${uuid}* - Universally unique identifier, a 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 Editor plugin installation folder as file path.
- *${homeDir}* - The path (as file path) of the user home folder.
- *${pn}* - Current project name.
- *${env(VAR_NAME)}* - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the *${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 Editor plugin provides a selection of document templates that make it easier to create new documents in a variety of formats. The list of available templates is presented when you create a new document. You can also create your own templates and share them with others. You can add your custom file templates by creating template files in a directory and adding that directory to the list of template directories that Oxygen XML Editor plugin uses. To add a template directory, *open the **Preferences** dialog box* and go to **Editor** > **Templates** > **Document Templates**.

You can add new document template location folders and manage existing ones. You can also alter the order in which Oxygen XML Editor plugin looks into these directories by using the **Up** and **Down** buttons.

Spell Check Preferences

Oxygen XML Editor plugin provides support for spell checking in the *text* and *author* editing modes. To configure the **Spell Check** options, *open the **Preferences** dialog box* and go to **Editor** > **Spell Check**.

The following options are available:

- **Spell checking engine** - Oxygen XML Editor plugin ships with two spell check engines, *Hunspell* and *Java spell checker*. The two engines come with different dictionaries. When you select an engine here, the list of languages in the **Default language** option changes based on the available dictionaries for the engine you have chosen.
- **Automatic Spell Check** - This option is disabled by default. When enabled, Oxygen XML Editor plugin automatically checks the 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, such as CSS and DTD, are excluded by default.

Language Options Section

- **Default language** - The default language list allows you to choose the language used by the spell check engine when the language is not specified in the source file. You can *add additional dictionaries to the spell check engines*.
- **Use "lang" and "xml:lang" attributes** - When this option is selected, the contents of an element with one of the `lang` or `xml:lang` attributes is checked in that language. When this option is enabled, choose between the following two options for instances **when these attributes are missing**:
  - **Use the default language** - If the `lang` and `xml:lang` attributes are missing, the selection in the **Default language** list is used.
  - **Do not check** - If the `lang` and `xml:lang` attributes are missing, the element is not checked.

XML Spell Checking in Section

You can choose to check the spelling inside the following XML items:

- **Comments**
- **Attribute values**
- **Text**
- **CDATA**

Options Section

- **Check capitalization** - When selected, the spell checker reports capitalization errors (for example, a word that starts with lowercase after *etc.* or *i.e.*.
- **Check punctuation** - When selected, the spell checker checks punctuation. Misplaced white space and unusual sequences, like a period following a comma, are highlighted as errors.
- **Ignore mixed case words** - When selected, the spell checker does not check words containing mixed case characters (for example, *SpellChecker*).
- **Ignore acronyms** - Available only for the *Hunspell spell checker*. When selected, acronyms are not reported as errors.
- **Ignore words with digits** - When selected, the spell checker does not check words containing digits (for example, *b2b*).
- **Ignore duplicates** - When selected, the spell checker does not signal two successive identical words as an error.
- **Ignore URL** - When selected, the spell checker ignores words looking like URLs or file names (for example, *www.oxygenxml.com* or *c:\boot.ini*).

- **Allow compounds words** - When selected, all words formed by concatenating two legal words with a hyphen (hyphenated compounds) are accepted. If recognized by the language, two words concatenated without hyphen (closed compounds) are also accepted.
- **Allow general prefixes** - Available only for the *Java spell checker*. When selected, a word formed by concatenating a recognized prefix and a legal word is accepted. For example if *mini-* is a registered prefix, the spell check engine accepts the word *mini-computer*.
- **Allow file extensions** - When selected, the spell checker accepts any word ending with recognized file extensions (for example, *myfile.txt* or *index.html*).

Ignore Elements Section

You can use the **Add** and **Remove** buttons to configure a list of element names or XPath expressions to be ignored by the spell checker. The following restricted set of XPath expressions are supported:

- '/' and '//' separators
- '*' wildcard

An example of an allowed XPath expression is: */a/*/b*.

To change the color used by the spell check engine to highlight spelling errors, go to **Window** (**Eclipse** on Mac OSX) and choose **Preferences**. Then go to **General** > **Editors** > **Text Editors** > **Annotations** > **Spell Check Annotation**.

Spell Check Dictionaries Preferences

To set the Dictionaries preferences, *open the **Preferences** dialog box* and go to **Editor** > **Spell Check** > **Dictionaries**. This page allows you to configure the dictionaries and term lists (`.tdi` files) that Oxygen XML Editor plugin uses and to choose where to save new learned words.

The following options are valid when Oxygen XML Editor plugin uses the Hunspell spell checking engine:

- **Dictionaries and term lists default folder** - Displays the default location where the dictionaries and term lists that Oxygen XML Editor plugin uses are stored.
- **Include dictionaries and term list from** - Selecting this option allows you to specify a location where you have stored dictionaries and term lists that you want to include, along with the default ones.

Note: The spell checker takes into account dictionaries and term lists collected both from the default and custom locations and multiple dictionaries and term lists from the same language are merged into generic ones (for example, `en_UK.dic` from the default location is merged with `en_US.dic` from a custom location, and the result is that a third file is created for a generic dictionary called `en.dic`). However, if there is already a generic dictionary (for example, `en.dic`) saved in either the default or custom location, the other specific dictionaries (for example, `en_UK.dic` and `en_US.dic`) will not be merged and the existing generic dictionary will simply be used. Also, if the additional location contains a file with the same name as one from the default location, the file in the additional location takes precedence over the file from the default location.

- **Save learned words in the following location** - Specifies the target where the newly learned words are saved. By default, the target is the application preferences folder, but you can also choose a custom location.
- **Delete learned words** - Opens the list of learned words, allowing you to select the items you want to remove, without deleting the dictionaries and term lists.

Document Checking Preferences

To configure the **Document Checking** options, *open the **Preferences** dialog box* and go to **Editor** > **Document Checking**. This preferences page contains preferences for configuring how a document is checked for both well-formedness errors and validation errors.

The following options are available:

- **Maximum number of validation highlights** - If validation generates more errors than the number from this option only the first errors up to this number are highlighted in editor panel and on stripe that is displayed at right side of editor panel. This option is applied both for *automatic validation* and *manual validation*.

- **Clear validation markers on close** - If this option is selected all the error markers added in the **Problems** view for that document are removed when a document edited with the Oxygen XML Editor plugin plugin is closed.
- **Enable automatic validation** - Validation of edited document is executed in background as the document is modified by editing in Oxygen XML Editor plugin.
- **Delay after the last key event (s)** - The period of keyboard inactivity which starts a new validation (in seconds).

Mark Occurrences Preferences

To configure the **Mark Occurrences** options, *open the **Preferences** dialog box* and go to **Editor** > **Mark Occurrences**:

The following preferences are available in this preferences page:

- **XML files** - Activates the *Highlight IDs Occurrences* feature in XML files.
- **XSLT files** - Activates the *Highlight Component Occurrences* feature in XSLT files.
- **XML Schema files and WSDL files** - Activates the *Highlight Component Occurrences* feature in XSD and WSDL files.
- **RNG files** - Activates the highlight component occurrences feature in RNG files.
- **Schematron files** - Activates the *Highlight Component Occurrences* feature in Schematron files.
- **Declaration highlight color** - Color used to highlight the component declaration.
- **Reference highlight color** - Color used to highlight component references.

Custom Validation Engines Preferences

To configure the options for *Custom Validation Engines*, *open the **Preferences** dialog box* and go to **Editor** > **Custom Validations**.

If you want to add a new custom validation tool or edit the properties of an exiting one you can use the **Custom Validator** dialog box displayed by pressing the **New** button or the **Edit** button.

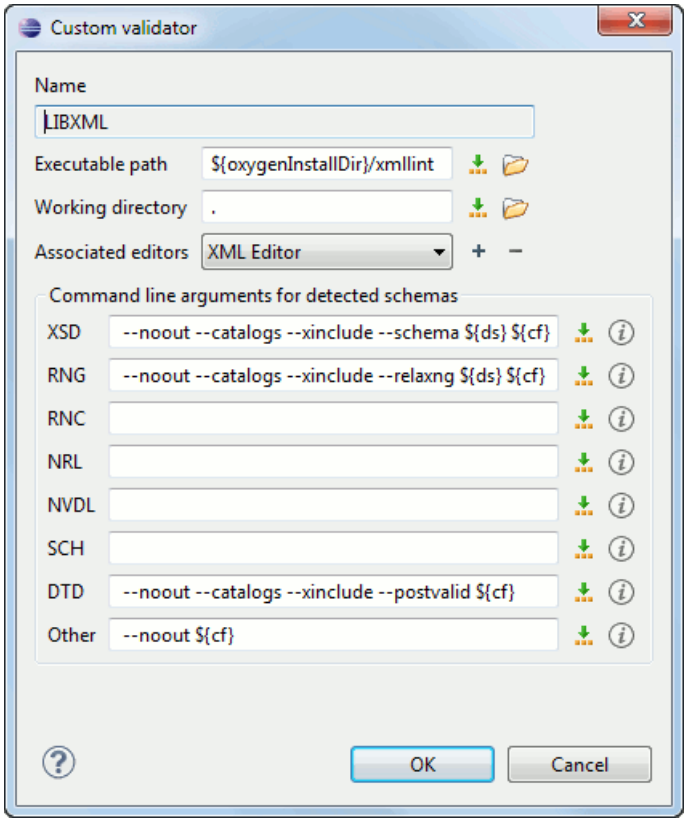


Figure 387: Edit a Custom Validator

The configurable parameters of a custom validator are as follows:

- **Name** - Name of the custom validation engine that will be displayed in the **Validation** toolbar drop-down menu.
- **Executable path** - Path to the executable file of the custom validation tool. You can insert *editor variables*, such as *${home}*, *${pd}*, *${oxygenInstallDir}*.
- **Working directory** - The working directory of the custom validation tool.
- **Associated editors** - The editors that can perform validation with the external tool (XML editor, XSL editor, XSD editor, etc.)
- **Command line arguments for detected schemas** - Command line arguments used in the commands that validate the current edited file against different types of schema (W3C XML Schema, Relax NG full syntax, Relax NG compact syntax, NVDL, Schematron, DTD, etc.) The arguments can include any custom switch (like -rng) and the following editor variables:
  - *${cf}* - Current file as file path, that is the absolute file path of the current edited document.
  - *${currentFileURL}* - Current file as URL, that is the absolute file path of the current edited document represented as URL.
  - *${ds}* - The path of the detected schema as a local file path for the current validated XML document.
  - *${dsu}* - The path of the detected schema as an URL for the current validated XML document.

Increasing the stack size for validation engines

To prevent the appearance of a *StackOverflowException*, use one of the following methods:

- Use the **com.oxygenxml.stack.size.validation.threads** property to increase the size of the stack for validation engines. The value of this property is specified in bytes. For example, to set a value of one megabyte specify 1x1024x1024=1048576.
- Increase the value of the **-Xss** parameter.

Note: Increasing the value of the **-Xss** parameter affects all the threads of the application.

Fonts Preferences

Oxygen XML Editor plugin allows you to choose the fonts to be used in the **Text**, **Design**, and **Grid** editor modes, and fonts for the **Author** mode that are not specified in the associated CSS stylesheet. To configure the font options, *open the **Preferences** dialog box* and go to **Fonts**.

The following options are available:

Text

This option allows you to choose the font used in **Text** mode. There are two options available:

- **Map to text font** - Uses the same font for the basic text editor as the one set in **General** > **Appearance** > **Colors and Fonts**.
- **Customize** - Allows you to choose a specific font.

Author default font

This option allows you to choose the default font that will be used in **Author** mode. The default font will be overridden by the fonts specified in any CSS file associated with the opened document.

Schema default font

This option allows you to choose the font to be used in:

- The **Design** mode of the *XML Schema editor*.
- Images with schema diagram fragments that are included in the HTML documentation generated from an XML Schema.

Note: You must restart the application for your changes to be applied.

Network Connection Settings Preferences

This section presents the options available in the **Network Connection Settings** preferences pages.

HTTP(S)/WebDAV Preferences

To set the **HTTP(S)/WebDAV** preferences, *open the **Preferences** dialog box* and go to **Network Connection Settings** > **HTTP(S)/WebDAV**. The following options are available:

- **Enable the HTTP(S)/WebDAV Protocols** - Activates the HTTP(S)/WebDAV protocols bundled with Oxygen XML Editor plugin. Any adjustment to this option requires a restart of the application.
- **Internal Apache HttpClient Version** - Oxygen XML Editor plugin uses the Apache HttpClient to establish connections to HTTP servers. To enable Oxygen XML Editor plugin to benefit from particular sets of features provided by different versions, you may choose between v3 and v4.

  Note: For a full list of features, go to *http://hc.apache.org/httpclient-3.x/* and *http://hc.apache.org/httpcomponents-client-ga/*

- **Maximum number of simultaneous connections per host** - Defines the maximum number of simultaneous connections established by the application with a distinct host. Servers might consider multiple connections opened from the same source to be a **Denial of Service** attack. You can avoid that by lowering the value of this option.

  Note: This option accepts a minimum value of 5.

- **Read Timeout (seconds)** - The period in seconds after which the application considers that an HTTP server is unreachable if it does not receive any response to a request sent to that server.
- **Enable HTTP 'Expect: 100-continue ' handshake for HTTP/1.1 protocol** - Activates *Expect: 100-Continue* handshake. The purpose of the *Expect: 100-Continue* handshake is to allow a client that is sending a request message with a request body to determine if the origin server is willing to accept the request (based on the request headers) before the client sends the request body. The use of the *Expect: 100-continue* handshake can result in noticeable performance improvement when working with databases. The *Expect: 100-continue* handshake should be used with caution, as it may cause problems with HTTP servers and proxies that do not support the HTTP/1.1 protocol.
- **Use the 'Content-Type' header field to determine the content type** - When checked, tries to determine a resource type using the **Content-Type** header field. This header indicates the *Internet media type* of the message content, consisting of a type and subtype, for example:

  ```
  Content-Type: text/xml
  ```

  When unchecked, the resource type is determined by analyzing its extension. For example, a file ending in `.xml` is considered to be an XML file.
- **Automatic retry on recoverable error** - If enabled, if an HTTP error occurs when communicates with a server via HTTP, for example sending / receiving a SOAP request / response to / from a Web services server, and the error is recoverable, tries to send again the request to the server.
- **Automatically accept a security certificate, even if invalid** - When enabled, the HTTPS connections that Oxygen XML Editor plugin attempts to establish will accept all security certificates, even if they are invalid.

  Important: By accepting an invalid certificate, you accept at your own risk a potential security threat, since you cannot verify the integrity of the certificate's issuer.

- **Encryption protocols (SVN Client only)** - this option is available only if you run the application with Java 1.6 or older. Sets a specific encryption protocol used when a repository is accessed through HTTPS protocol. You can choose one of the following values:
  - **SSLv3, TLSv1** (default value);
  - **SSLv3 only**;
  - **TLSv1 only**.
- **Lock WebDAV files on open** - If checked, the files opened through WebDAV are locked on the server so that they cannot be edited by other users while the lock placed by the current user still exists on the server.

(S)FTP Preferences

To configure the **(S)FTP** options, *open the **Preferences** dialog box* and go to **Network Connection Settings** > **(S)FTP**. You can customize the following options:

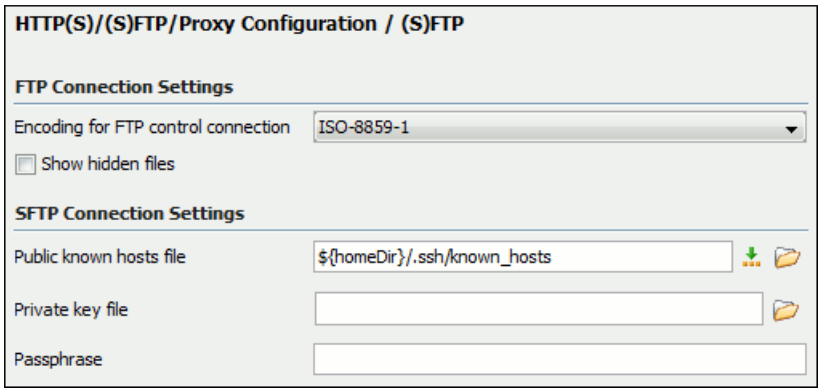


Figure 388: The (S)FTP Configuration Preferences Panel

- **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 Editor plugin will use it for communication. Otherwise it will use ISO-8859-1.
- **Public known hosts file** - File containing the list of all SSH server host keys that you have determined are accurate. The default value is `${homeDir}/.ssh/known_hosts`.
- **Private key file** - The path to the file containing the private key used for the private key method of authentication of the secure FTP (SFTP) protocol.
- **Passphrase** - The passphrase used for the private key method of authentication of the secure FTP (SFTP) protocol.
- **Show SFTP certificate warning dialog** - If checked, a warning dialog box will be displayed each time when the authenticity of the host cannot be established.

Trusted Hosts Preferences

Holds a list with domains that have been marked as trusted by the user. Oxygen XML Editor plugin will connect to these hosts without requesting user confirmation.

To configure the **Trusted Hosts** options, *open the **Preferences** dialog box* and go to **Network Connection Settings** > **Trusted Hosts**. The following options are available:

- **New** - Allows you to manually add a new entry in the list of trusted hosts.
- **Delete** - Allows you to remove an entry from the trusted hosts list.

Scenarios Management Preferences

To configure Scenarios Management options, *open the **Preferences** dialog box* and go to **Scenarios Management**. This allows you to share the global transformation scenarios with other users by exporting them to an external file that can also be imported in this preferences panel.

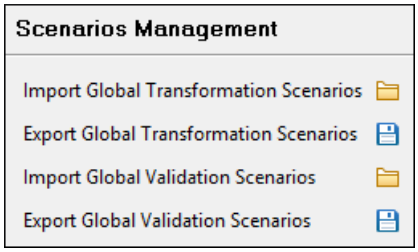


Figure 389: The Scenarios Management Preferences Panel

The actions available in this panel are as follows:

- **Import Global Transformation Scenarios** - Allows you to import all global-level transformation scenarios from a file created with the export scenario action. The names of the imported scenarios will appear in the **Configure Transformation Scenario** dialog box followed by *(import)*. This way there are no scenario name conflicts.
- **Export Global Transformation Scenarios** - Allows you to export all global transformation scenarios available in the **Configure Transformation Scenario** dialog box.
- **Import Global Validation Scenarios** - Allows you to import all global-level scenarios from a file created with the export scenario action. The names of the imported scenarios will appear in the **Configure Validation Scenario** dialog box followed by *(import)*. This way there are no scenario name conflicts.
- **Export Global Validation Scenarios** - Allows you to export all global validation scenarios available in the **Configure Validation Scenario** dialog box.

View Preferences

To configure the view options, *open the **Preferences** dialog box* and go to **Views** and contains the following preferences:

- **Fixed width console** - If checked, a line in the **Console** view will be hard wrapped after the maximum numbers of characters allowed on a line.
- **Limit console output** - If checked, the content of the **Console** view will be limited to a configurable number of characters.
- **Console buffer** - Specifies the maximum number of characters that can be written in the **Console** view.
- **Tab width** - Specifies the number of spaces used for depicting a tab character.

XML Preferences

This section describes the panels that contain the user preferences related with XML.

XML Catalog Preferences

To configure the **XML Catalog** options, *open the **Preferences** dialog box* and go to **XML** > **XML Catalog**.

The following options are available:

- **Prefer** - the prefer setting determines whether public identifiers specified in the catalog are used in favor of system identifiers supplied in the document. Suppose you have an entity in your document for which both a public identifier and a system identifier has been specified, and the catalog only contains a mapping for the public identifier (for example, 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 Editor plugin analyzes both *uri* and *system* mappings in order to resolve the location of schema.

  Note: This option is not applicable for DTD schemas since the public and system catalog mappings are always considered.

- 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 Editor plugin will use to resolve references at document validation and transformation will be a default built-in catalog. This catalog maps such references to the built-in local copies of the schemas of the Oxygen XML Editor plugin frameworks (DocBook, DITA, TEI, XHTML, SVG, etc.)

You can also add or configure catalogs at framework level from the **Catalogs** tab in the *Document Type configuration dialog box*.

When you add, delete or edit an XML catalog to / from the list, reopen the currently edited files which use the modified catalog or run *the Validate action* so that the XML catalog changes take full effect.

XML Parser Preferences

To configure the **XML Parser** options, *open the Preferences dialog box* and go to **XML** > **XML Parser**.

The configurable options of the built-in XML parser are as follows:

- **Enable parser caching (validation and content completion)** - Enables re-use of internal models when validating and provides content completion in opened XML files which reference the same schemas (grammars) like DTD, XML Schema, RelaxNG.
- **Ignore the DTD for validation if a schema is specified** - Forces validation against a referenced schema (W3C XML Schema, Relax NG schema) even if the document includes also a DTD DOCTYPE declaration. This option is useful when the DTD declaration is used only to declare DTD entities and the schema reference is used for validation against a W3C XML Schema or a Relax NG schema.

  Note: Schematron schemas are treated as additional schemas. The validation of a document associated with a DTD and referencing a Schematron schema is executed against both the DTD and the Schematron schema, regardless of the value of the **Ignore the DTD for validation if a schema is specified** option.

- **Enable XInclude processing** - Enables XInclude processing. If checked, the XInclude support in Oxygen XML Editor plugin is turned on for validation, rendering in **Author** mode and transformation of XML documents.
- **Base URI fix-up** - According to the specification for XInclude, processors must add an `xml:base` attribute to elements included from locations with a different base URI. Without these attributes, the resulting infoset information would be incorrect.

  Unfortunately, these attributes make XInclude processing not transparent to Schema validation. One solution to this is to modify your schema to allow `xml:base` attributes to appear on elements that might be included from different base URIs.

  If the addition of `xml:base` and / or `xml:lang` is undesired by your application, you can disable base URI fix-up.

- **Language fix-up** - The processor will preserve language information on a top-level included element by adding an `xml:lang` attribute if its include parent has a different [language] property. If the addition of `xml:lang` is undesired by your application, you can disable the language fix-up.
- **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. If you disable this option, the current XML file is validated against the associated DTD before all the content merged to the current XML file using `XInclude` is resolved.

XML Schema Preferences

To configure the **XML Schema** options, *open the Preferences dialog box* and go to **XML** > **XML Parser** > **XML Schema**.

This preferences page allows you to configure the following options:

- **Default XML Schema version** - Allows you to select the version of W3C XML Schema: XML Schema **1.0** or XML Schema **1.1**.

Note: You are also able to set the XML Schema version using the **Customize** option in *New dialog box*.

- **Default XML Schema validation engine** - Allows you to set the default XML Schema validation engine either to **Xerces** or **Saxon EE**.

Xerces validation features

- **Enable full schema constraint checking** (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 enabled (default value), you can use the full XPath support in assertions and alternative types. Otherwise, only the XPath support offered by the Xerces engine is available.
- **Assertions can see comments and processing instructions** (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 box* and go to **XML** > **XML Parser** > **Relax NG**.

The following options are available in this page:

- **Check feasibly valid** - Checks whether Relax NG documents can be transformed into valid documents by inserting any number of attributes and child elements anywhere in the tree.

  Note: Enabling this option disables the **Check ID/IDREF** option.

- **Check ID/IDREF** - Checks the ID/IDREF matches when a Relax NG document is validated.
- **Add default attribute values** - Default values are given to the attributes of documents validated using Relax NG. These values are defined in the Relax NG schema.

Schematron Preferences

To configure the **Schematron** options, *open the **Preferences** dialog box* and go to **XML** > **XML Parser** > **Schematron**.

The following options are available in this preferences page:

ISO Schematron Section

- **Optimize (visit-no-attributes)** - If your ISO Schematron assertion tests do not contain the attributes axis, you should check this option for faster ISO Schematron validation.
- **Allow foreign elements (allow-foreign)** - Enables support for `allow-foreign` on ISO Schematron. This option is used to pass non-Schematron elements to the generated stylesheet.

- **Use Saxon EE (schema aware) for xslt2/xslt3 query language binding** - When enabled, Saxon EE is used for `xslt2`/`xslt3` query binding. If this option is disabled, Saxon PE is used.
- **Enable Schematron Quick Fixes (SQF) support** - Allows you to enable or disable the support for quick fixes in Schematron files. This option is enabled by default.
- **Embedded rules query language binding** - You can control the query language binding used by the ISO Schematron embedded rules. You can choose between: **xslt1**, **xslt2**, or **xslt3**.

  Note: To control the query language binding for standalone ISO Schematron, you need to set the query language to be used with a `queryBinding` attribute on the schema root element.

Schematron 1.5 Section

- **XPath Version** - Allows you to select the version of XPath for the expressions that are allowed in Schematron assertion tests. You can choose between: **1.0**, **2.0**, or **3.0**. This option is applied in both standalone Schematron 1.5 schemas and embedded Schematron 1.5 rules.

XML Instances Generator Preferences

To configure the **XML Instances Generator** options, *open the **Preferences** dialog box* and go to **XML** > **XML Instances Generator**. It sets the default parameters of the **Generate Sample XML Files** tool that is available on the **Tools** menu.

The options of the tool that generates XML instance documents based on a W3C XML Schema are the following:

- **Generate optional elements** - If checked, the elements declared optional in the schema will be generated in the XML instance.
- **Generate optional attributes** - If checked, the attributes declared optional in the schema will be generated in the XML instance.
- **Values of elements and attributes** - Specifies what values are generated in elements and attributes of the XML instance. It can have one of the values:
  - **None** - No values for the generated elements and attributes.
  - **Default** - The value is the element name or attribute name.
  - **Random** - A random value.
- **Preferred number of repetitions** - If the values set here is greater than `maxOccurs`, then the `maxOccurs` is used.
- **Maximum recursivity level** - For recursive type definitions this parameter specifies the number of levels of recursive elements inserted in the parent element with the same name.
- **Type alternative strategy** - Specifies how the element type alternatives are generated in the XML instance:
  - **First** - The first element type alternative whose XPath condition is true is used.
  - **Random** - A random element type alternative whose XPath condition is true is used.

    Note: If no XPath condition is true, the default element type alternative is used.

  - **Note:** To evaluate an XPath expression, either the values of the attributes defined in the **Options** tab of the **XML Instance Generator** dialog box, or the attributes values defined in the XML Schema are used.

- **Choice strategy** - For choice element models specifies what choice will be generated in the XML instance:
  - **First** - The first choice is selected from the choice definition and an instance of that choice is generated in the XML instance document.
  - **Random** - A random choice is selected from the choice definition and an instance of that will be generated.
- **Generate the other options as comments** - If checked, the other options of the choice element model (the options which are not selected) will be generated inside an XML comment in the XML instance.
- **Use incremental attribute / element names as default** - If checked, the value of an element or attribute starts with the name of that element or attribute. For example, for an `a` element the generated values are: `a1`, `a2`, `a3`, and so on. If not checked, the value is the name of the type of that element / attribute (for example: `string`, `decimal`, etc.)

- **Maximum length** - The maximum length of string values generated for elements and attributes.
- **Discard optional elements after nested level** - The optional elements that exceed the specified nested level are discarded. This option is useful for limiting deeply nested element definitions that can quickly result in very large XML documents.

XProc Engines Preferences

Oxygen XML Editor plugin comes with a built-in XProc engine called *Calabash*. You can add or configure external XProc engines by using the **XProc Engines** preferences page. *Open the **Preferences** dialog box* and go to **XML** > **XProc**.

When **Show XProc messages** is selected all messages emitted by the XProc processor during a transformation will be presented in the results view.

To add an external engine click the **New** button. To configure an existing engine, click the **Edit** button. This opens the **Custom Engine** dialog box that allows you to configure an external engine.

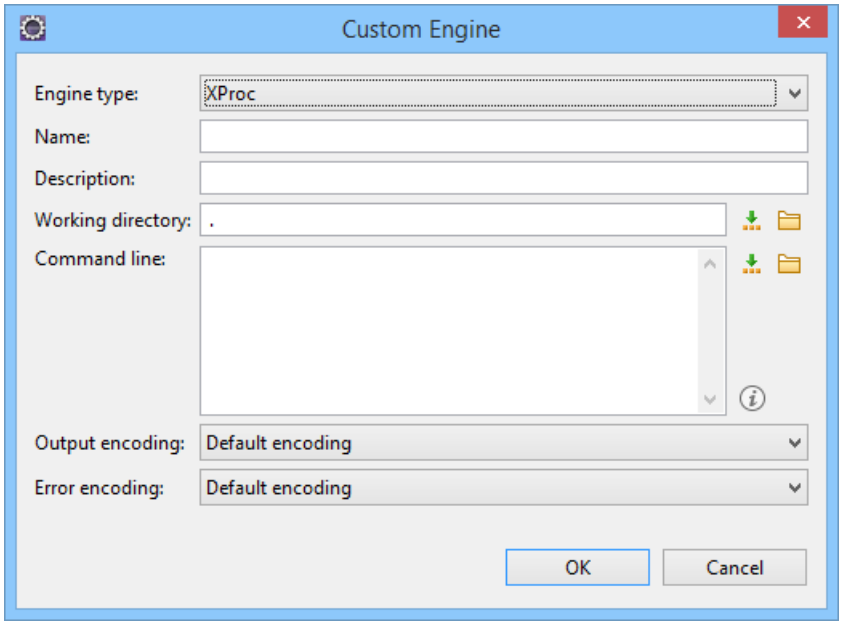


Figure 390: Creating an XProc external engine

The following options can be configure in this dialog box:

- **Name** - The value of this field will be displayed in the XProc transformation scenario and in the command line that will start it.
- **Description** - A textual description that will appear as a tooltip where the XProc engine will be used.
- **Working directory** - The working directory for resolving relative paths. You can use *built-in editor variables* and *custom editor variables* for the path.
- **Command line** - The command line that will run the XProc engine as an external process. You can use *built-in editor variables* and *custom editor variables* for parametrizing a file path.
- **Output encoding** - The encoding for the output stream of the XProc engine, used for reading and displaying the output messages.
- **Error encoding** - The encoding for the error stream of the XProc engine, used for reading and displaying the messages from the error stream.

Note: You can configure the built-in Saxon processor using the `saxon.config` file. For further details about configuring this file go to *http://www.saxonica.com/documentation9.5/index.html#!configuration/configuration-file*. You can configure the built-in Calabash processor by using the `calabash.config` file. These files are located in `[OXYGEN_DIR]\lib\xproc\calabash\lib`. If they do not exist, you have to create them.

XSLT-FO-XQuery Preferences

To configure the **XSLT/FO/XQuery** options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery**. This panel contains only the most generic options for working with XSLT / XSL-FO / XQuery processors. The more specific options are grouped in other panels linked as child nodes of this panel in the tree of this **Preferences** page.

There is only one generic option available:

Create transformation temporary files in system temporary directory - It should be selected only when the temporary files necessary for performing transformations are created in the same folder as the source of the transformation (the default behavior, when this option is not selected) and this breaks the transformation. An example of breaking the transformation is when the transformation processes all the files located in the same folder as the source of the transformation, which will include the temporary files, and the result is incorrect or the transformation fails due to this fact.

XSLT Preferences

To configure the **XSLT** options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XSLT**.

Oxygen XML Editor plugin gives you the possibility to use an XSLT transformer implemented in Java (other than the XSLT transformers that come bundled with Oxygen XML Editor plugin). To use a different XSLT transformer, specify the name of the transformer factory class. Oxygen XML Editor plugin sets this transformer factory class as the value of the Java property `javax.xml.transform.TransformerFactory`.

You can customize the following XSLT preferences:

- **Value** - Allows you to set the value of the `TransformerFactory` Java class.
- **XSLT 1.0 Validate with** - Allows you to set the XSLT engine used for validation of XSLT 1.0 documents.
- **XSLT 2.0 Validate with** - Allows you to set the XSLT Engine used for validation of XSLT 2.0 documents.
- **XSLT 3.0 Validate with** - Allows you to set the XSLT Engine used for validation of XSLT 3.0 documents.

Note: Saxon-HE does not implement any XSLT 3.0 features. Saxon-PE implements a selection of XSLT 3.0 (and XPath 3.0) features, with the exception of schema-awareness and streaming. Saxon-EE implements additional features relating to streaming (processing of a source document without constructing a tree in memory. For further details about XSLT 3.0 conformance, go to *http://www.saxonica.com/documentation/index.html#!conformance/xslt30*.

Saxon6 Preferences

To configure the **Saxon 6** options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **Saxon** > **Saxon 6**.

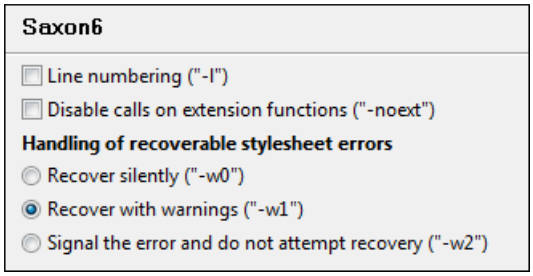


Figure 391: The Saxon 6 XSLT Preferences Panel

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 you to choose how dynamic errors are handled. One of the following options can be selected:
  - **recover silently** - Continue processing without reporting the error.
  - **recover with warnings** - Issue a warning but continue processing.
  - **signal the error and do not attempt recovery** - Issue an error and stop processing.

Saxon-HE/PE/EE Preferences

To configure the **Saxon HE/PE/EE** options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **Saxon** > **Saxon HE/PE/EE**.

Oxygen XML Editor plugin allows you to configure the following XSLT options for the Saxon 9.6.0.7 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE):

- **Use a configuration file ("-config")** - Sets a Saxon 9.6.0.7 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 numbers are included in the output messages.
- **Debugger trace into XPath expressions (applies to debugging sessions)** - Instructs the *XSLT Debugger*XSLT Debugger to step into XPath expressions.
- **Expand attributes defaults ("-expand")** - Specifies whether or not the attributes defined in the associated DTD or XML Schema are expanded in the output of the transformation you are executing.
- **DTD validation of the source ("-dtd")** - The following options are available:
  - **On** - Requests *DTD* validation of the source file and of any files read using the `document()` function.
  - **Off** - (default setting) Suppresses DTD validation.
  - **Recover** - Performs DTD validation but treats the errors as non-fatal.

    Note: Any external DTD is likely to be read even if not used for validation, since DTDs can contain definitions of entities.

- **Recoverable errors ("-warnings")** - Policy for handling recoverable errors in the stylesheet: Allows you to choose how dynamic errors are handled. One of the following options can be selected:
  - **Recover silently ("silent")** - Continues processing without reporting the error.
  - **Recover with warnings ("recover")** - (default setting) Issues a warning but continues processing.
  - **Signal the error and do not attempt recovery ("fatal")** - Issues an error and stops processing.
- **Strip whitespaces ("-strip")** - Strip whitespaces feature can be one of the following three options:
  - **All ("all")** - Strips all whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes in the source document.
  - **Ignorable ("ignorable")** - Strips all ignorable whitespace text nodes from source documents before any further processing, regardless of any `xsl:strip-space` declarations in the stylesheet, or any `xml:space` attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
  - **None ("none")** - default setting. No whitespaces are stripped before further processing. However, whitespace are stripped if this is specified in the stylesheet using `xsl:strip-space`.
- **Optimization level ("-opt")** - Set the optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization). This option allows optimization to be suppressed when reducing the compiling time is important, optimization conflicts with debugging, or optimization causes extension functions with side-effects to behave unpredictably.

The advanced options available only in Saxon Professional Edition (PE) and Enterprise Edition (EE) are:

- **Allow calls on extension functions ("-ext")** - If checked, the stylesheet is allowed to call external Java functions. This does not affect calls on integrated extension functions, including Saxon and EXSLT extension functions. This

option is useful when loading an untrusted stylesheet, such as from a remote site using an http://[URL]. It ensures that the stylesheet cannot call arbitrary Java methods and thus gain privileged access to resources on your machine.

- **Register Saxon-CE extension functions and instructions** - Registers the Saxon-CE extension functions and instructions when compiling a stylesheet using the Saxon 9.6.0.7 processors.

  Note: Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Editor plugin only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about *executing such a stylesheet on Saxonica's website*.

The advanced options available only in Saxon Enterprise Edition (EE) are:

- **XML Schema version** - Use this option to change the default XML Schema version. To change the default XML Schema version, *open the **Preferences** dialog box* and go to **XML** > **XML Parser** > **XML Schema**.

  Note: This option is available when you configure the Saxon EE advanced options from a transformation scenario.

- **Validation of the source file ("-val")** - Requests schema-based validation of the source file and of any files read using the `document()` or similar functions. It can have the following values:
  - **Schema validation ("strict")** - This mode requires an XML Schema and specifies that the source documents should be parsed with strict schema-validation enabled.
  - **Lax schema validation ("lax")** - If an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled but the validation will not fail if, for example, element declarations are not found.
  - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.
- **Validation errors in the results tree treated as warnings ("-outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.
- **Write comments for non-fatal validation errors of the result document** - The validation messages are written (where possible) as a comment in the result document itself.
- **Generate bytecode ("--generateByteCode:(on|off)")** - If you enable this option, Saxon-EE attempts to generate Java bytecode for evaluation of parts of a query or stylesheet that are amenable to such an action. For further details regarding this option, go to *http://www.saxonica.com/documentation9.5/index.html#!javadoc*.

Saxon HE/PE/EE Advanced Preferences

To configure the **Saxon HE/PE/EE Advanced** preferences, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **Saxon** > **Saxon HE/PE/EE** > **Advanced**.

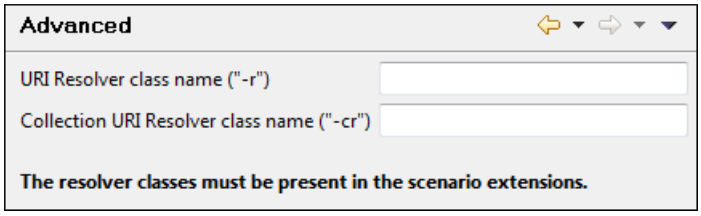


Figure 392: The Saxon HE/PE/EE XSLT Advanced Preferences Panel

You can configure the following advanced XSLT options for the Saxon 9.6.0.7 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 9.6.0.7 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 box for configuring the XSLT extension* for the particular transformation scenario.
- **Collection URI Resolver class name ("-cr")** - Specifies a custom implementation for the Collection URI resolver used by the XSLT Saxon 9.6.0.7 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 box for configuring the XSLT extension* for the particular transformation scenario.

XSLTProc Preferences

To configure **XSLTProc** options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **XSLTProc**.

The options of the XSLTProc processor are the same as the ones available in the command line:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when XSLTProc is used as transformer in *XSLT transformation scenarios*.
- **Skip loading the document's DTD** - If checked, the DTD specified in the DOCTYPE declaration will not be loaded.
- **Do not apply default attributes from document's DTD** - If checked, the default attributes declared in the DTD and not specified in the document are not included in the transformed document.
- **Do not use Internet to fetch DTD's, entities or docs** - If checked, the remote references to DTD's and entities are not followed.
- **Maximum depth in templates stack** - If this limit of maximum templates depth is reached the transformation ends with an error.
- **Verbosity** - If checked, the transformation will output detailed status messages about the transformation process in the **Warnings** view.
- **Show version of libxml and libxslt used** - If checked, Oxygen XML Editor plugin will display in the **Warnings** view the version of the **libxml** and **libxslt** libraries invoked by XSLTProc.
- **Show time information** - If checked, the **Warnings** view will display the time necessary for running the transformation.
- **Show debug information** - If checked, the **Warnings** view will display debug information about what templates are matched, parameter values, and so on.
- **Show all documents loaded during processing** - If checked, Oxygen XML Editor plugin will display in the **Warnings** view the URL of all the files loaded during transformation.
- **Show profile information** - If checked, Oxygen XML Editor plugin will display in the **Warnings** view a table with all the matched templates, and for each template will display: the match XPath expression, the template name, the number of template modes, the number of calls, the execution time.
- **Show the list of registered extensions** - If checked, Oxygen XML Editor plugin will display in the **Warnings** view a list with all the registered extension functions, extension elements and extension modules.
- **Refuses to write to any file or resource** - If checked, the XSLTProc processor will not write any part of the transformation result to an external file on disk. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.
- **Refuses to create directories** - If checked, the XSLTProc processor will not create any directory during the transformation process. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.

MSXML Preferences

To configure the MSXML options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **MSXML**.

The options of the MSXML 3.0 and 4.0 processors are the same as *the ones available in the command line for the MSXML processors*:

- **Validate documents during parse phase** - If checked and either the source or stylesheet document has a DTD or schema against which its content can be checked, validation is performed.
- **Do not resolve external definitions during parse phase** - By default, MSXSL instructs the parser to resolve external definitions such as document type definition (DTD), external subsets or external entity references when parsing the source and style sheet documents. If this option is checked the resolution is disabled.
- **Strip non-significant whitespaces** - If checked, strips non-significant white space from the input XML document during the load phase. Enabling this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.
- **Show time information** - If checked, the relative speed of various transformation steps can be measured:

- The time to load, parse, and build the input document.
- The time to load, parse, and build the stylesheet document.
- The time to compile the stylesheet in preparation for the transformation.
- The time to execute the stylesheet.

- **Start transformation in this mode** - Although stylesheet execution usually begins in the empty mode, this default may be changed by specifying another mode. Changing the start mode allows execution to jump directly to an alternate group of templates.

MSXML.NET Preferences

To configure the **MSXML.NET** options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XSLT** > **MSXML.NET**.

The options of the MSXML.NET processor are:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when MSXML.NET is used as transformer in the *XSLT transformation scenario*.
- **Validate documents during parse phase** - If checked and either the source or stylesheet document has a DTD or schema against which its content can be checked, validation is performed.
- **Do not resolve external definitions during parse phase** - By default MSXML.NET resolves external definitions such as DTD external subsets or external entity references when parsing source XML document and stylesheet document. Using this option you can disable this behavior. 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 (for example, `Acme.Resolvers.CacheResolver`). Such name requires additional assembly specification using this option or the next option, but fully qualified class name (which always includes an assembly specifier) is all-sufficient. See MSDN for more info about *fully qualified class names*. This option specifies a file name of the assembly, where the specified resolver class can be found.
- **Assembly GAC name for URI resolver class** - This option specifies partially or fully qualified name of the assembly in the *global assembly cache* (GAC), where the specified resolver class can be found. See MSDN for more info about *partial assembly names*. Also see the previous option.
- **List of extension object class names** - This option allows to specify *extension object* classes, whose public methods then can be used as extension functions in an XSLT stylesheet. It is a comma-separated list of namespace-qualified extension object class names. Each class name must be bound to a namespace URI using prefixes, similar to providing XSLT parameters.

- **Use specified EXSLT assembly** - MSXML.NET supports a rich library of the *EXSLT* and *EXSLT.NET extension functions* embedded or in a plugged in EXSLT.NET library. EXSLT support is enabled by default and cannot be disabled in this version. If you want to use an external EXSLT.NET implementation instead of a built-in one use this option.
- **Credential loading source xml** - This option allows to specify user credentials to be used when loading XML source documents. The credentials should be provided in the *username:password@domain* format (all parts are optional).
- **Credential loading stylesheet** - This option allows to specify user credentials to be used when loading XSLT stylesheet documents. The credentials should be provided in the *username:password@domain* format (all parts are optional).

XQuery Preferences

To configure the **XQuery** options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XQuery**.

The generic XQuery preferences are the following:

- **XQuery validate with** - Allows you to select the processor that validates XQuery documents. If you are validating an XQuery file that has an associated validation scenario, Oxygen XML Editor plugin uses the processor specified in the scenario. If no validation scenario is associated, but the file has an associated transformation scenario, the processor specified in the scenario is used. If the processor does not support validation or if no scenario is associated, then the value from this combo box will be used as validation processor.
- **Size limit of Sequence view (MB)** - When the result of an XQuery transformation is *set in the transformation scenario as sequence* the size of one chunk of the result that is fetched from the database in lazy mode in one step is set in this option. If this limit is exceed, go to the **Sequence** view and click **More results available** to extract more data from the database.
- **Format transformer output** - Specifies whether the output of the transformer is formatted and indented (pretty printed).

  Note: This option is ignored if you choose **Sequence** (lazy extract data from a database) from the associated transformation scenario.

- **Create structure indicating the type nodes** - If checked, Oxygen XML Editor plugin takes the results of a query and creates an XML document containing copies of all items in the sequence, suitably wrapped.

  Note: This option is ignored if you choose **Sequence** (lazy extract data from a database) from the associated transformation scenario.

Saxon HE/PE/EE Preferences

To configure the **Saxon HE/PE/EE** options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XQuery** > **Saxon HE/PE/EE**.

The XQuery preferences for the Saxon 9.6.0.7 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

- **Use a configuration file ("-config")** - Sets a Saxon 9 configuration file that is used for XQuery transformation and validation
- **Recoverable errors ("-warnings")**- Allows you to choose how dynamic errors are handled. The following options can be selected:
  - **recover silently ("silent")** - Continues processing without reporting the error.
  - **recover with warnings ("recover")** - Issues a warning but continues processing.
  - **signal the error and do not attempt recovery ("fatal")** - Issues an error and stops processing.
- **Strip whitespaces ("-strip")** - Can have one of the following values:
  - **All ("all")** - Strips *all* whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document.

  - **Ignore ("ignorable")** - Strips all *ignorable* whitespace text nodes from source documents before any further processing, regardless of any `xml:space` attributes in the source document. Whitespace text nodes are ignorable if they appear in elements defined in the DTD or schema as having element-only content.
  - **None ("none")** - Strips *no* whitespace before further processing.
- **Optimization level ("-opt")** - Set the optimization level. The value is an integer in the range 0 (no optimization) to 10 (full optimization). This option allows optimization to be suppressed when reducing the compiling time is important, optimization conflicts with debugging, or optimization causes extension functions with side-effects to behave unpredictably.
- **Use linked tree model ("-tree:linked")** - This option activates the linked tree model.
- **Enable XQuery 3.0 support ("-qversion:(1.0|3.0)")** - If checked, Saxon runs the XQuery transformation with the XQuery 3.0 support (this option is enabled by default).

The following option is available for Saxon 9.6.0.7 Professional Edition (PE) and Enterprise Edition (EE) only:

- **Allow calls on extension functions ("-ext")** - If checked, calls on external functions are allowed. Checking this option is recommended in an environment where untrusted stylesheets may be executed. It also disables user-defined extension elements and the writing of multiple output files, both of which carry similar security risks.

The options available specifically for Saxon 9.6.0.7 Enterprise Edition (EE) are as follows:

- **Validation of the source file ("-val")** - Requests schema-based validation of the source file and of any files read using the `document()` or similar functions. It can have the following values:
  - **Schema validation ("strict")** - This mode requires an XML Schema and enables parsing the source documents with strict schema-validation enabled.
  - **Lax schema validation ("lax")** - If an XML Schema is provided, this mode enables parsing the source documents with schema-validation enabled but the validation will not fail if, for example, element declarations are not found.
  - **Disable schema validation** - This specifies that the source documents should be parsed with schema-validation disabled.
- **Validation errors in the results tree treated as warnings ("-outval")** - Normally, if validation of result documents is requested, a validation error is fatal. Enabling this option causes such validation failures to be treated as warnings.
- **Enable XQuery update ("-update:(on|off)")** - This option controls whether or not XQuery update syntax is accepted. The default value is off.
- **Backup files updated by XQuery ("-backup:(on|off)")** - If checked, backup versions for any XML files updated with XQuery Update are generated. This option is available when the **Enable XQuery update** option is enabled.

Saxon HE/PE/EE Advanced Preferences

To configure **Saxon HE/PE/EE Advanced** preferences, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XQuery** > **Saxon HE/PE/EE** > **Advanced**.

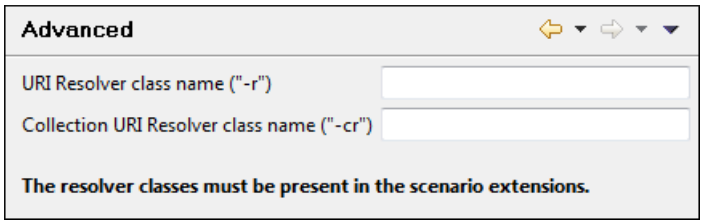


Figure 393: The Saxon HE/PE/EE XQuery Advanced Preferences Panel

The advanced XQuery options which can be configured for the Saxon 9.6.0.7 XQuery transformer (all editions: Home Edition, Professional Edition, Enterprise Edition) are the following:

- **URI Resolver class name** - Allows you to specify a custom implementation for the URI resolver used by the XQuery Saxon 9.6.0.7 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 box for configuring the XQuery extension* for the particular transformation scenario.

Note: If your `URIResolver` implementation does not recognize the given resource, the `resolve(String href, String base)` method should return a `null` value. Otherwise, the given resource will not be resolved through the *XML catalog*.

- **Collection URI Resolver class name** - Allows you to specify a custom implementation for the Collection URI resolver used by the XQuery Saxon 9.6.0.7 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 box for configuring the XQuery extension* for the particular transformation scenario.

Debugger Preferences

To configure the **Debugger** preferences, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **Debugger**.

The following preferences are available:

- **Show xsl:result-document output** - if checked, the debugger presents the output of `xsl:result-document` instructions into the debugger output view.
- **Infinite loop detection** - set this option to receive notifications when an infinite loop occurs during transformation.
- **Maximum depth in templates stack** - sets how many `xsl:template` instructions can appear on the current stack. This setting is used by the infinite loop detection.
- **Debugger layout** - a horizontal layout means that the stack of XML editors takes the left half of the editing area and the stack of XSL editors takes the right one. A vertical layout means that the stack of XML editors takes the upper half of the editing area and the stack of XSL editors takes the lower one.
- **XWatch evaluation timeout (seconds)** - specifies the maximum time that Oxygen XML Editor plugin allocates to the evaluation of XPath expressions while debugging.
- **Messages** - specifies whether a debugging session is stopped, is continued, or you are asked what to do when you are editing the source document involved in a debugging session.

Annotations Preferences

To configure the **Annotations** options, go to **Window** (**Eclipse** on Mac OSX) and choose **Preferences**. Then go to **General** > **Editors** > **Text Editors** > **Annotations**.

The following Oxygen XML Editor plugin preferences are available:

- **XSLT/XQuery Debug Current Instruction Pointer** - Controls the background color of the current execution node, both in the document (XML) and XSL/XQuery views.

Profiler Preferences

This section explains the settings available for the XSLT Profiler. To access and modify these settings, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **Profiler** (see *Debugger Preferences* on page 970).

The following profile settings are available:

- **Show time** - Shows the total time that was spent in the call.
- **Show inherent time** - Shows the inherent time that was spent in the call. The inherent time is defined as the total time of a call minus the time of its child calls.
- **Show invocation count** - Shows how many times the call was called in this particular call sequence.
- **Time scale** - The time scale options determine the unit of time measurement, which may be milliseconds or microseconds.
- ***Hotspot*** threshold - The threshold below which hot spots are ignored (milliseconds).
- ***Ignore invocation less than*** - The threshold below which invocations are ignored (microseconds).
- **Percentage calculation** - The percentage base that determines what time span percentages are calculated against:
  - **Absolute** - Percentage values show the contribution to the total time.
  - **Relative** - Percentage values show the contribution to the calling call.

FO Processors Preferences

Oxygen XML Editor plugin includes a built-in formatting objects processor (Apache FOP), but you can also configure other external processors and use them in the transformation scenarios for processing XSL-FO documents.

Oxygen XML Editor plugin provides an easy way to add two of the most commonly used commercial FO processors: *RenderX XEP* and *Antenna House 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 Formatter*, Oxygen XML Editor plugin uses the environment variables set by the XSL formatter installation to detect and use it for XSL-FO transformations. If the environment 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 options for the FO Processors, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **FO Processors**. The **FO Processors** preferences page is displayed.

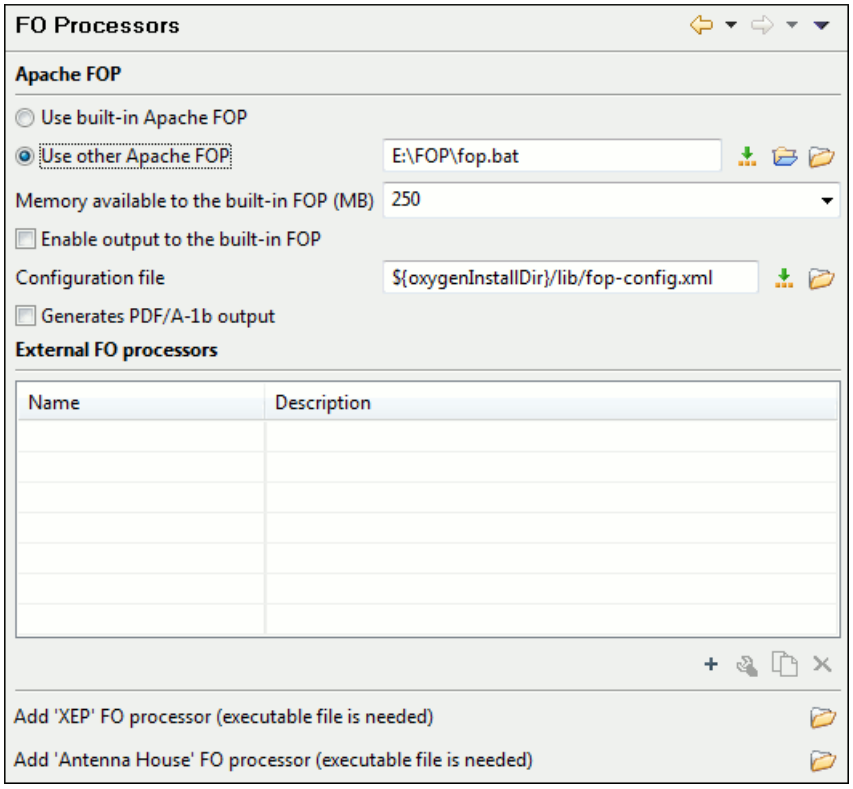


Figure 394: The FO Processors Preferences Page

Apache FOP Section

In this section you can configure options for the built-in Apache processor. The following options are available:

Use built-in Apache FOP

Instructs Oxygen XML Editor plugin to use the built-in Apache FO processor.

Use other Apache FOP

Instructs Oxygen XML Editor plugin to use another Apache FO processor that is installed on your computer.

Enable the output of the built-in FOP

All Apache FOP output is displayed in a results pane at the bottom of the Oxygen XML Editor plugin window, including warning messages about FO instructions not supported by Apache FOP.

Memory available to the built-in FOP

If your Apache FOP transformations fail with an Out of Memory error (**OutOfMemoryError**), use this combo box to select a bigger value for the amount of memory reserved for FOP transformations.

Configuration file for the built-in FOP

Use this option to specify the path to an Apache FOP configuration file (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 since the FOP would generate the following error: *The Filter key is prohibited when PDF/A-1 is active*.

External FO Processors Section

In this section you can manage the external FO processors you want to use in transformation scenarios. You can use the two options at the bottom of the section to use the *RenderX XEP* or *Antenna House Formatter* commercial FO processors.

Add 'XEP' FO processor (executable file is needed)

If *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)

If *Antenna House 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, and `run.sh` for Linux/Mac OS).

Note: The built-in **Antenna House Formatter GUI** transformation scenario requires that you configure an external FO processor that runs `AHFormatter.exe` (Windows only). In the *external **FO Processor** configuration dialog box*, you could use `"${env(AHF62_64_HOME)}\AHFormatter.exe" -d ${fo} -s` for the value in the **Command line** field, although the environment variable name changes for each version of the AH Formatter and for each system architecture (you can install multiple versions side-by-side). For more information, see *https://github.com/AntennaHouse/focheck/wiki/focheck*.

You can also add external processors or configure existing ones. Press the **New** button to open a configuration dialog box that allows you to add a new external FO processor. Use the other buttons (**Edit**, **Duplicate**, **Delete**) to configure existing external processors.

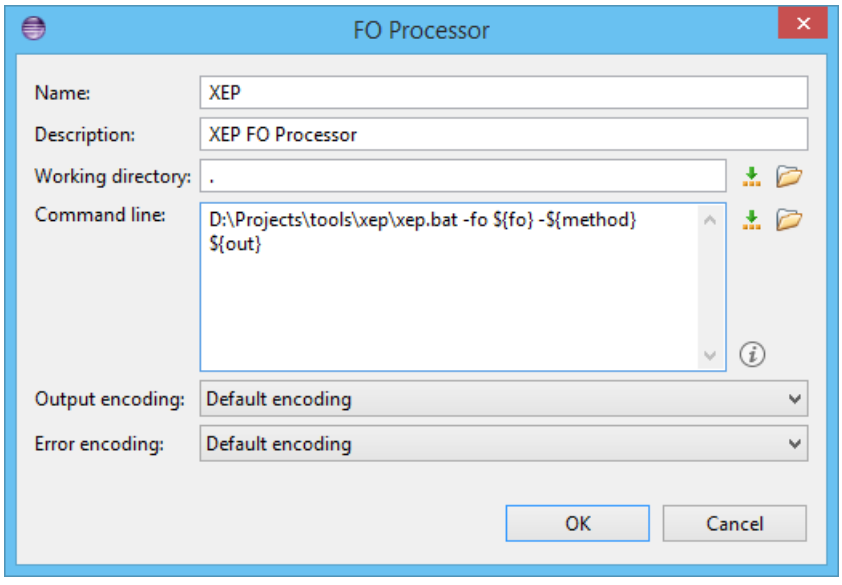


Figure 395: The External FO Processor Configuration Dialog Box

The external **FO Processor** configuration dialog box includes the following options:

Name

The name that will be displayed in the list of available FO processors on the FOP tab of the transformation scenario dialog box.

Description

A textual description of the FO processor that will be displayed in the FO processors table and in tooltips of UI components where the processor is selected.

Working directory

The directory where the intermediate and final results of the processing is stored. You can use one of the following editor variables:

- **${homeDir}** - The path to the user home directory.
- **${cfd}** - The path of the current file directory. If the current file is not a local file, the target is the user desktop directory.
- **${pd}** - The project directory.
- **${oxygenInstallDir}** -The Oxygen XML Editor plugin installation directory.

Command line

The command line that starts the FO processor, specific to each processor. You can use one of the following editor variables:

- **${method}** - The FOP transformation method: **pdf**, **ps**, or **txt**.
- **${fo}** - The input FO file.
- **${out}** - The output file.
- **${pd}** - The project directory.
- **${frameworksDir}** - The path of the `frameworks` subdirectory of the Oxygen XML Editor plugin installation directory.
- **${oxygenInstallDir}** - The Oxygen XML Editor plugin installation directory.
- **${ps}** - The platform-specific path separator. It is used between the library files specified in the class path of the command line.

Output Encoding

The encoding of the FO processor output stream that is displayed in a results panel at the bottom of the Oxygen XML Editor plugin window.

Error Encoding

The encoding of the FO processor error stream that is displayed in a results panel at the bottom of the Oxygen XML Editor plugin window.

XPath Preferences

To configure the **XPath** options, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **XPath**.

Oxygen XML Editor plugin allows you to customize the following options:

- **Unescape XPath expression** - the entities of an XPath expressions that you type in the XPath/XQuery Builder are unescaped during their execution. For example the expression

  ```
  //varlistentry[starts-with(@os,'&#x73;')]
  ```

  is equivalent with:

  ```
  //varlistentry[starts-with(@os,'s')]
  ```

- **No namespace** - if you enable this option, Oxygen XML Editor plugin considers unprefixed element names of the evaluated XPath 2.0 / 3.0 expressions as belonging to no namespace.
- **Use the default namespace from the root element** - if you enable this option, Oxygen XML Editor plugin considers unprefixed element names of the evaluated XPath expressions as belonging to the default namespace declared on the root element of the XML document you are querying.

- **Use the namespace of the root** - if you enable this option, Oxygen XML Editor plugin considers unprefixed element names of the evaluated XPath expressions as belonging to the same namespace as the root element of the XML document you are querying.
- **This namespace** - in this field you can enter the namespace of the unprefixed elements.
- **Default prefix-namespace mappings** - in this field you can associate prefixes with namespaces. Use these mappings when you want to define them globally, not for each document.

Custom Engines Preferences

You can configure and run XSLT and XQuery transformations with processors other than *the ones which come with the Oxygen XML Editor plugin distribution*.

Note: You can not use these custom engines in *the Debugger perspective*.

To configure the **Custom Engines** preferences, *open the **Preferences** dialog box* and go to **XML** > **XSLT/FO/XQuery** > **Custom Engines**.

The following parameters can be configured for a custom engine:

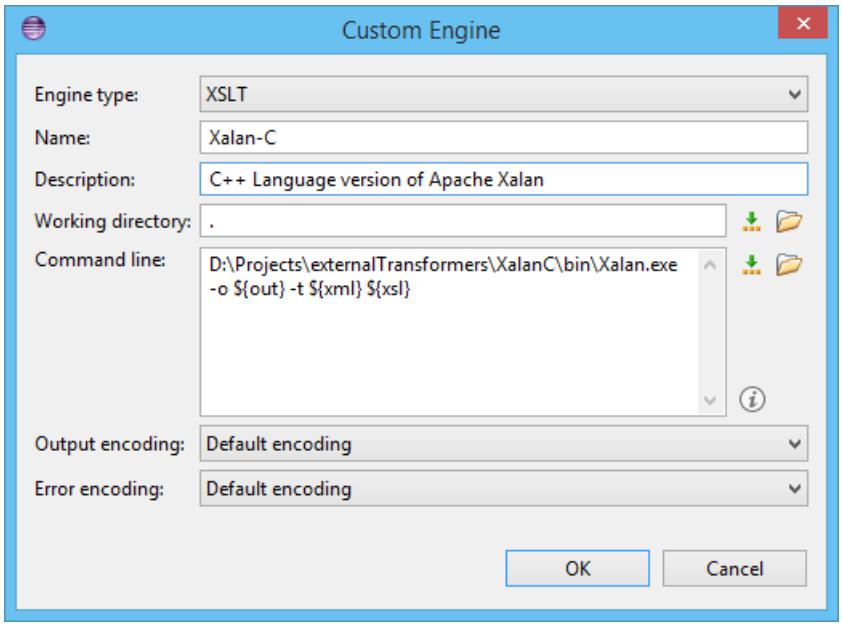


Figure 396: Parameters of a Custom Engine

- **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 box for editing transformation scenarios
- **Description** - A textual description of the transformer.
- **Working directory** - The start directory of the transformer executable program. The following editor variables are available for making the path to the working directory independent of the location of the input files:
  - **${homeDir}** - The user home directory in the operating system.
  - **${cfd}** - The path to the directory of the current file.
  - **${pd}** - The path to the directory of the current project.
  - **${oxygenInstallDir}** - The Oxygen XML Editor plugin install directory.
- **Command line** - The command line that must be executed by Oxygen XML Editor plugin to perform a transformation with the engine. The following editor variables are available for making the parameters in the command line (the transformer executable, the input files) independent of the location of the input files:
  - **${xml}** - The XML input document as a file path.
  - **${xmlu}** - The XML input document as a URL.

- **${xsl}** - The XSL / XQuery input document as a file path.
- **${xslu}** - The XSL / XQuery input document as a URL.
- **${out}** - The output document as a file path.
- **${outu}** - The output document as a URL.
- **${ps}** - The platform separator which is used between library file names specified in the class path.

- **Output Encoding** - The encoding of the transformer output stream.
- **Error Encoding** - The encoding of the transformer error stream.

Import Preferences

To configure the **Import** options, *open the **Preferences** dialog box* and go to **XML** > **Import**. This page allows you to configure how empty values and `null` values are handled when they are encountered in imported database tables or Excel sheets. Also you can configure the format of date / time values recognized in the imported database tables or Excel sheets.

The following options are available:

- **Create empty elements for empty values** - If checked, an empty value from a database column or from a text file is imported as an empty element.
- **Create empty elements for null values** - If checked, `null` values from a database column are imported as empty elements.
- **Escape XML content** - Enabled by default, this option instructs Oxygen XML Editor plugin to escape the imported content to an XML-safe form.
- **Add annotations for generated XML Schema** - If checked, the generated XML Schema contains an annotation for each of the imported table columns. The documentation inside the annotation tag contains the remarks of the database columns (if available) and also information about the conversion between the column type and the generated XML Schema type.

The section **Date / Time Format** specifies the format used for importing date and time values from Excel spreadsheets or database tables and in the generated XML schemas. The following format types are available:

- **Unformatted** - If checked, the date and time formats specific to the database are used for import. When importing data from Excel a string representation of date or time values are used. The type used in the generated XML Schema is `xs:string`.
- **XML Schema date format** - If checked, the XML Schema-specific format ISO8601 is used for imported date / time data (`yyyy-MM-dd'T'HH:mm:ss` for `datetime`, `yyyy-MM-dd` for `date` and `HH:mm:ss` for `time`). The types used in the generated XML Schema are `xs:datetime`, `xs:date` and `xs:time`.
- **Custom format** - If checked, you 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 16: 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

Letter	Date or Time Component	Presentation	Examples
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 Editor plugin provides two types of *keystores* for certificates that are used for digital signatures of XML documents: Java KeyStore (**JKS**) and Public-Key Cryptography Standards version 12 (**PKCS-12**). A *keystore* file is protected by a password. To configure a certificate *keystore*, *open the **Preferences** dialog box* and go to **XML** > **XML Signing Certificates**. You can customize the following parameters of a *keystore*:

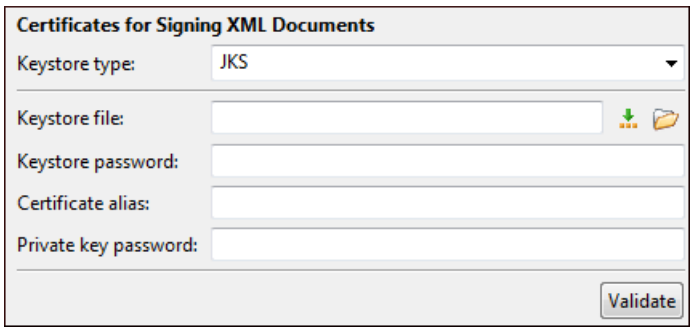


Figure 397: The Certificates Preferences Panel

- **Keystore type** - The type of *keystore* that Oxygen XML Editor plugin uses (**JKS** or **PKCS-12**).
- **Keystore file** - The location of the imported file.
- **Keystore password** - The password that is used for protecting the privacy of the stored keys.
- **Certificate alias** - The alias used for storing the key entry (the certificate or the private key) inside the *keystore*.
- **Private key password** - The private key password of the certificate (required only for JKS *keystores*).
- **Validate** - Press this button to verify the configured *keystore* and the validity of the certificate.

XML Structure Outline Preferences

To configure the **XML Structure Outline** options, *open the **Preferences** dialog box* and go to **XML Structure Outline**, which contains the following preferences:

- **Preferred attribute names for display** - The preferred attribute names when displaying the attributes of an element in the **Outline** view. If there is no preferred attribute name specified, the first attribute of an element is displayed.
- **Enable outline drag and drop** - Drag and drop is disabled for the tree displayed in the **Outline** view only if there is a possibility to accidentally change the structure of the document by such operations.

Configuring Options

A set of options controls the behavior of Oxygen XML Editor plugin, allowing you to configure most of the features. To offer you the highest degree of flexibility in customizing the application to fit the needs of your organization, Oxygen XML Editor plugin comes with several distinct layers of option values.

The option layers are as follows (sorted from high priority to low):

Global Options

Allows individual users to personalize Oxygen XML Editor plugin according to their specific needs.

Customized Default Options

Designed to customize the initial option values for a group of users, this layer allows an administrator to deploy the application preconfigured with a standardized set of option values.

Note: Once this layer is set, it represents the initial state of Oxygen XML Editor plugin when an end-user uses the ***Restore defaults*** or ***Reset Global Options*** actions.

Default Options

The predefined default or built-in values, tuned so that Oxygen XML Editor plugin behaves optimally in most working environments.

Note: If you set a specific option in one of the layers, but it is not applied in the application, make sure that one of the higher priority layers does not overwrite it.

Customizing Default Options

Oxygen XML Editor plugin has an extensive set of options that you can configure. When Oxygen XML Editor plugin in installed, these options are set to default values. You can provide a different set of default values for an installation using an XML *options file*.

Creating an XML *Options File*

To create an *options file*, follow these steps:

1. It is recommended that you use a fresh install for this procedure, to make sure that you do not copy personal or local preferences.
2. Open Oxygen XML Editor plugin and *open the **Preferences** dialog box*.
3. Go through the options and set them to the desired defaults.
4. Go to back to the main preferences page and click **Export Global Options** to create an XML options file.

Using Customized Default Options

Use either one of the following ways to configure an Oxygen XML Editor plugin installation to use customized default options from a created XML options file:

- In the [OXYGEN_DIR] installation folder, create a folder called `preferences` and copy the XML options file into it (for example, `[Eclipse-platform-install-folder]/plugins/com.oxygenxml.editor/preferences/default.xml` or `[Eclipse-platform-install-folder]/dropins/[OXYGEN_DIR]/preferences/default.xml` if the plugin was installed as a drop-in).
- Set the path to the XML options file as the value of the `com.oxygenxml.default.options` system property in the Eclipse configuration file.

  Add the following line in the `[Eclipse-platform-install-folder]/configuration/config.ini` file:

  ```
  com.oxygenxml.default.options=file\:@config.dir/../default.xml
  ```

Importing / Exporting Global Options

Actions for importing, exporting, and resetting global options are available in the preferences page of the Oxygen XML Editor plugin. To open this page, *open the **Preferences** dialog box*. The export operation allow you to save global preferences as an XML options file and the import operation allows you to load the options file. You can use this file to reload the options on your computer or to share with others.

The following buttons are available at the bottom of the preferences page:

Reset Global Options

Restores the preference to the factory defaults or to *customized defaults*.

Import Global Options

Allows you to import a set of *Global Options* from an exported XML options file. You can also select a project file (`.xpr`) to import all the *Global Options* that are set in that project file. After you select a file, the **Import Global Options** dialog box is displayed and it informs you that the operation will only override the options that are included in the imported file. You can enable the **Reset all other options to their default values** option to reset all options to the default values before the file is imported.

Export Global Options

Allows you to export *Global Options* to an XML `options` file. Some user-specific options that are private are not included. For example, passwords and the name of the *Review Author* is not included in the export operation.

Reset Global Options

To reset all global preferences to their default values, *open the **Preferences** dialog box* and click the **Reset Global Options** button.

This action also resets the transformation and validation scenarios to the default scenarios.

Scenarios Management

You can export global transformation and validation scenarios into specialized *scenarios* files. You can import transformation and validation scenarios from various sources (such as project files, framework option files, or exported scenario files). To access these import and export actions, *open the **Preferences** dialog box* and go to **Scenarios Management**. The following actions are available:

Import Global Transformation Scenarios **Button**

Loads a set of transformation scenarios from a project file, framework options file, or exported scenarios file.

Export Global Transformation Scenarios **Button**

Stores a set of global (not project-level) transformation scenarios in a specialized *scenarios* file.

Import Global Validation Scenarios **Button**

Loads a set of validation scenarios from a project file, framework options file, or exported scenarios file.

Export Global Validation Scenarios **Button**

Stores a set of global (not project-level) Validation scenarios in a specialized *scenarios* file.

The **Export Global Transformation Scenarios** and **Export Global Validation Scenarios** options are used to store all the scenarios in a separate file. Associations between document URLs and scenarios are also saved in this file. You can load the saved scenarios using the **Import Global Transformation Scenarios** and **Import Global Validation Scenarios** actions. To distinguish the existing scenarios and the imported ones, the names of the imported scenarios contain the word *import*.

Editor Variables

An editor variable is a shorthand notation for context-dependent information, such as a file or folder path, a time-stamp, or a date. It is used in the definition of a command (for example, the input URL of a transformation, the output file path of a transformation, or the command line of an external tool) to make a command or a parameter generic and re-usable with other input files. When the same command is applied to different files, the notation is expanded at the execution of the command so that the same command has different effects depending on the actual file.

You can use the following editor variables in Oxygen XML Editor plugin commands of external engines or other external tools, in transformation scenarios, and in validation scenarios:

- *${oxygenHome}* - Oxygen XML Editor plugin installation folder as URL.
- *${oxygenInstallDir}* - Oxygen XML Editor plugin installation folder as file path.
- *${framework}* - The path (as URL) of the current framework, as part of the `[OXYGEN_DIR]/frameworks` directory.
- *${framework(fr_name)}* - The path (as URL) of the `fr_name` framework.
- *${frameworkDir(fr_name)}* - The path (as file path) of the `fr_name` framework.

Note: Since 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 Editor plugin employs the following algorithm when searching for a given framework name:

- All frameworks are sorted, from high to low, according to their ***Priority*** setting from the ***Document Type** configuration dialog box*. Only frameworks that have the **Enabled** checkbox set are taken into account.

    - Next, if the two or more frameworks have the same name and priority, a further sorting based on the **Storage** setting is made, in the exact following order:
        - Frameworks stored in the internal Oxygen XML Editor plugin options.
        - Additional frameworks added in the ***Locations*** *preferences page*.
        - Frameworks installed using the add-ons support.
        - Frameworks found in the *main frameworks location* (**Default** or **Custom**).

- *${frameworks}* - The path (as URL) of the `[OXYGEN_DIR]` directory.
- *${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.
- *${cf}* - Current file as file path, that is the absolute file path of the current edited document.
- *${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.
- *${caret}* - The position where the cursor is located. This variable can be used in a code template, in **Author** mode operations, or in a selection plugin.
- *${selection}* - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** mode 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, a 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, with one of the following values:

Parameter	
`url`	**Format:** `${ask('message', url, 'default_value')}`
	Description: Input is considered a URL. Oxygen XML Editor plugin checks that the provided URL is valid.
	Example: • `${ask('Input URL', url)}` - The displayed dialog box has the name `Input URL`. The expected input type is URL. • `${ask('Input URL', url, 'http://www.example.com')}` - The displayed dialog box has the name `Input URL`. The expected input type is URL. The input field displays the default value `http://www.example.com`.
`password`	**Format**: *${ask('message', password, 'default')}*
	Description: The input is hidden with bullet characters.
	Example: • `${ask('Input password', password)}` - The displayed dialog box has the name `'Input password'` and the input is hidden with bullet symbols. • `${ask('Input password', password, 'abcd')}` - The displayed dialog box has the name `'Input password'` and the input hidden with bullet symbols. The input field already contains the default `abcd` value.
`generic`	**Format**: *${ask('message', generic, 'default')}*
	Description: The input is considered to be generic text that requires no special handling.
	Example: • `${ask('Hello world!')}` - The dialog box has a `Hello world!` message displayed. • `${ask('Hello world!', generic, 'Hello again!')}` - The dialog box has a `Hello world!` message displayed and the value displayed in the input box is `'Hello again!'`.
`relative_url`	**Format:** `${ask('message', relative_url, 'default')}`
	Description: Input is considered a URL. Oxygen XML Editor plugin tries to make the URL relative to that of the document you are editing. **Note:** If the `$ask` editor variable is expanded in content that is not yet saved (such as an *untitled* file, whose path cannot be determined), then Oxygen XML Editor plugin will transform it into an absolute URL.
	Example: • *${ask('File location', relative_url, 'C:/example.txt')}* - The dialog box has the name `'File location'`. The URL inserted in the input box is made relative to the current edited document location.

Parameter	
`combobox`	**Format:** `${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a drop-down menu. The drop-down menu is populated with the given `rendered_value` values. Choosing such a value will return its associated value (`real_value`). **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The drop-down menu displays the three given operating systems. The associated value will be returned based upon your selection. **Note:** In this example, the default value is indicated by the `osx` key. However, the same result could be obtained if the default value is indicated by `Mac OS X`, like in the following example: `${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'Mac OS X')}` • `${ask('Mobile OS', combobox, ('win':'Windows Mobile';'ios':'iOS';'and':'Android'), 'Android')}`
`editable_combobox`	**Format:** `${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a drop-down menu with editable elements. The drop-down menu is populated with the given `rendered_value` values. Choosing such a value will return its associated real value (`real_value`) or the value inserted when you edit a list entry. **Note:** The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The drop-down menu displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.
`radio`	**Format:** `${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}`
	Description: Displays a dialog box that offers a series of radio buttons. Each radio button displays a `'rendered_value` and will return an associated `real_value`.

Parameter	
	Note: The `'default'` parameter specifies the default selected value and can match either a key or a value.
	Example: • `${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}` - The dialog box has the name `'Operating System'`. The radio button group allows you to choose between the three operating systems. **Note:** In this example `Mac OS X` is the default selected value and if selected it would return `osx` for the output.

  - `'default-value'` - optional parameter. Provides a default value.

- *${date(pattern)}* - Current date. The allowed patterns are equivalent to the ones in the *Java SimpleDateFormat class*. **Example:** `yyyy-MM-dd`;

  Note: This editor variable supports both the xs:date and xs:datetime parameters. For details about xs:date, go to *http://www.w3.org/TR/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 a parameter of an **Author** mode operation.

    Example:

    ```
    ${ask('Set new ID attribute', generic, '${xpath_eval(@id)}')}
    ```

- *${i18n(key)}* - Editor variable used only at framework level to allow translating names and descriptions of **Author** mode actions in multiple actions. For more details see the *Localizing Frameworks* on page 585 section.

Custom Editor Variables

An editor variable can be created 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, or a custom FO processor.

You can create or configure custom editor variables in the ***Custom Editor Variables*** *preferences page*. To create a custom editor variable, click the **New** button and 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** of that variable.

Localizing of the User Interface

To localize the Oxygen XML Editor plugin, you can use one of the following methods:

- Localization through the update site:

  Start Eclipse, go to **Help** > **Install New Software**. Press Add Site in the **Available Software** tab of the **Software Updates** dialog box. Enter *http://www.oxygenxml.com/InstData/Editor/Eclipse/site.xml* in the location field of the **Add Site** dialog box. Press OK. Select the language pack checkbox.

- Localization through the zip archive:

  Go to *http://www.oxygenxml.com/download.html* and download the zip archive with the plugin language pack. Unzip the downloaded zip archive in the `dropins` subdirectory of the Eclipse install directory. Restart Eclipse.

If your operating system is running in the language you want to start Eclipse in (for example, you are using Japanese version of Windows, and you want to start Eclipse in Japanese), Oxygen XML Editor plugin matches the appropriate language from the language pack. However, if your operating system is running in a language other than the one you want to start Eclipse in (for example, you are using the English version of Windows, and you want to start Eclipse in Japanese, if you have the required operating system language support including the keyboard layouts and input method editors installed), specify the `-nl <locale>` command line argument when you launch Eclipse. Oxygen XML Editor plugin uses the translation file which matches the specified <locale>.

You can also localize the Eclipse plugin to a different language that the initial languages in the language pack. Duplicate the `plugin.properties` file from the Oxygen XML Editor plugin plugin installation directory, translate all the keys in the file and change its name to plugin_<locale>.properties.

Chapter 21

Common Problems

Topics:

This section provides a variety of common problems and their solutions.

Performance Problems

This section contains solutions for some common performance problems that may appear when running Oxygen XML Editor plugin.

Performance Issues with Large Documents

While editing large documents in Oxygen XML Editor plugin, if you see that performance slows down considerably over time, then a possible cause is that the application needs more memory to run properly. You can increase the maximum amount of memory available to Oxygen XML Editor plugin by setting the `-vmargs` and `-Xmx` parameters in the command used to launch the Eclipse platform.

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.

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 Editor plugin throws an *Out Of Memory* error, *open the **Preferences** dialog box*, go to **XML** > **XSLT-FO-XQuery** > **FO Processors**, and increase the value of the *Memory available to the built-in FOP* option.

For external XSL-FO processors, XSLT processors, and external tools, the maximum value of the allocated memory is set in the command line of the tool using the -Xmx parameter set to the Java virtual machine.

Common Problems and Solutions

This chapter presents common problems that may appear when running the application and the solutions for these problems.

Details to Submit in a Request for Technical Support Using the Online Form

What details should I add to my request for technical support on the online form in the product website?

When completing a request for Technical Support using the online form, include as many details as possible about your problem. For problems where a simple explanation may not be enough for the Technical Support team to reproduce or address the issue (such as server connection errors, unexpected delays while editing a document, an application crash, etc.), you should generate a log file and attach it to the problem report. In the case of a crash, you should also attach the crash report file generated by your operating system.

To generate an Oxygen XML Editor plugin log file, follow these steps:

1. Create a text file called `log4j.properties` in the `lib` folder of the installed plugin folder, with the following content:

```
log4j.rootCategory= debug, R2

log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=${user.home}/Desktop/oxygenLog/oxygen.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
```

2. Restart the application.
3. Reproduce the error.
4. Close the application.
5. Delete the `log4j.properties` file because it might cause performance issues if you leave it in the `lib` folder.

Important: The logging mode may severely decrease the performance of the application. Therefore, please do not forget to delete the `log4j.properties` file when you are done with the procedure.

The resulting log file is named `oxygen#.log` (for example, `oxygen.log`, `oxygen.log.1`, `oxygen.log.2`, etc.) and is located in the `Desktop\oxygenLog` folder.

Oxygen XML Editor plugin Takes Several Minutes to Start on Mac

If Oxygen XML Editor plugin takes several minutes to start, the Java framework installed on the Mac may have a problem. One solution for this is to update Java to the latest version: go to **Apple symbol** > **Software Update**. After it finishes to check for updates, click **Show Details**, select the Java Update (if one is available) and click **Install**. If no Java updates are available, reset the Java preferences to their defaults. Start **Applications > Utilities > Java Preferences** and click **Restore Defaults**.

XSLT Debugger Is Very Slow

When I run a transformation in the **XSLT Debugger** perspective it is very slow. Can I increase the speed?

If the transformation produces HTML or XHTML output you should *disable rendering of output in the XHTML output view* during the transformation process. To view the XHTML output result do one of the following:

- Run the transformation in the **Editor** perspective and make sure the ***Open in Browser/System Application*** *option* is enabled.
- Run the transformation in the **XSLT Debugger** perspective, save the text output area to a file, and use a browser application for viewing it (for example Firefox or Internet Explorer).

Syntax Highlight Not Available in Eclipse Plugin

I associated the `.ext` extension with Oxygen XML Editor plugin in Eclipse. Why does an `.ext` file opened with the Oxygen XML Editor plugin plugin not have syntax highlight?

Associating an extension with Oxygen XML Editor plugin in Eclipse versions 3.6-3.8, 4.2-4.5 requires three steps:

1. Associate the `.ext` extension with the Oxygen XML Editor plugin plugin.
   a) *Open the **Preferences** dialog box* and go to **General** > **Editors** > **File Associations**.
   b) Add `*.ext` to the list of file types.
   c) Select `*.ext` in the list by clicking it.
   d) Add Oxygen XML Editor plugin to the list of **Associated editors** and make it the default editor.
2. Associate the `.ext` extension with the **Oxygen XML** content type.
   a) *Open the **Preferences** dialog box* and go to **General** > **Content Types**.
   b) Add `*.ext` to the **File associations** list for the **Text** > **XML** > **Oxygen XML Editor plugin** content type.
3. Press the **OK** button in the Eclipse preferences dialog box.

Now when an `*.ext` file is opened the icon of the editor and the syntax highlight should be the same as for XML files opened with the Oxygen XML Editor plugin plugin.

Damaged File Associations on OS X

After upgrading OS X and Oxygen XML Editor plugin, it is no longer associated to the appropriate file types (such as XML, XSL, XSD, etc.) How can I create the file associations again?

The upgrade damaged the file associations in the LaunchService Database on your OS X machine. You can 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 Editor plugin installations on your hard drive.
2. Delete them by dragging them to the Trash.
3. Clear the Trash.

4. Unpack the Oxygen XML Editor plugin installation kit on your desktop.
5. Copy the contents of the archive into the folder `/ Applications / Oxygen`.
6. Run the following command in a Terminal:

```
/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 have the Oxygen XML Editor plugin icon.
9. Double-click the file.
10. Accept the confirmation.

When you start Oxygen XML Editor plugin the file associations should work correctly.

Signature Verification Failed Error on Open or Edit a Resource from Documentum

When I try to open/edit a resource from Documentum, I receive the following error:

`signature verification failed: certificate for All-MB.jar.checksum not signed by a certification authority`.

The problem is that the certificates from the Java Runtime Environment 1.6.0_22 or later no longer validate the signatures of the UCF jars.

Edit the `eclipse.ini` file from the Eclipse directory and add the following parameter to the -vmargs: `-Drequire.signed.ucf.jars=false`, for example:

```
-vmargs
-Xms40m
-Xmx256m
-Drequire.signed.ucf.jars=false
```

Compatibility Issue Between Java and Certain Graphics Card Drivers

Under certain settings, a compatibility issue can appear between Java and some graphics card drivers, which results in the text from the editor (in **Author** or **Text** mode) being displayed garbled. If you encounter this problem, update your graphics card driver. Another possible workaround is, *open the **Preferences** dialog box*, go to **Fonts**, and set the value of **Text antialiasing** option to ON.

Note: If this workaround does not resolve the problem, set the **Text antialiasing** option to other values.

An Image Appears Stretched Out in the PDF Output

When publishing XML content (DITA, DocBook, etc.), images are sometimes 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:

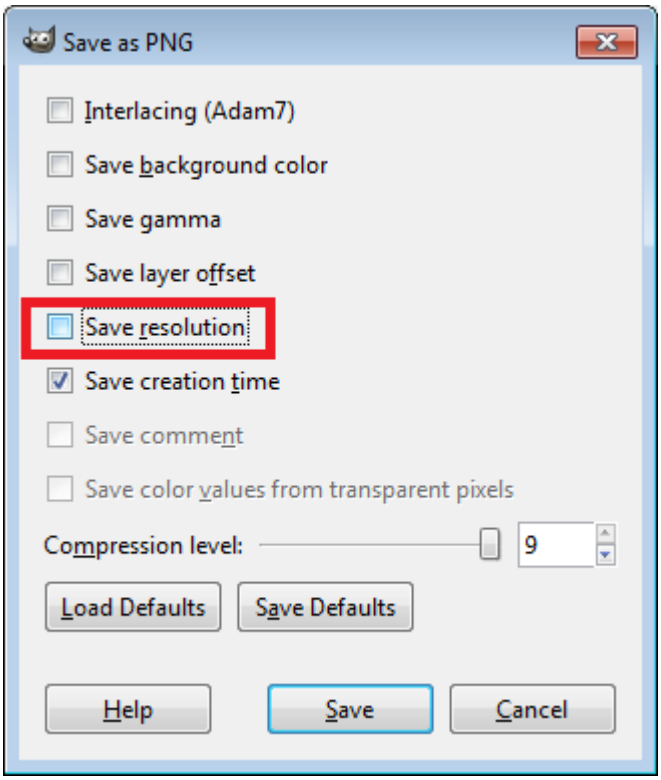
1. If the XSL-FO output contains width, height or a scale specified for the image `external-graphic` tag, then these dimensions are used. This means that if in the XML (DITA, DocBook, etc.) you set explicit dimensions to the image they will be used as such in the PDF output.

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 that has the Apache FOP configuration file located in `[DITA_OT_DIR/plugins/org.dita.pdf2/fop/conf/fop.xconf`. The configuration file contains two XML elements called `source-resolution` and `target-resolution`. The values set to those elements can be increased, usually a DPI value of 110 or 120 should render the image in PDF just like in the HTML output.

The commercial **RenderX XEP** XSL-FO processor behaves similarly but as a fallback it uses 120 as the DPI value instead of using a configuration file.

Tip:

As a conclusion, it is best to save your images without any DPI resolution information in them. For example the open-source GIMP image editor allows you when saving a PNG image whether to save the resolution to it or not:



Having images without any resolution information saved in them allows you to control the image resolution from the configuration file for all referenced images.

The DITA PDF Transformation Fails

To generate the PDF output, Oxygen XML Editor plugin 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.

The *DITA to CHM* Transformation Fails

Oxygen XML Editor plugin uses the DITA Open Toolkit and the HTML Help compiler (part of the Microsoft HTML Help Workshop) to transform DITA content into *Compiled HTML Help* (or *CHM* in short).

It is a good practice to validate the DITA map before executing the transformation scenario. To do so, run *the* ***Validate and Check for Completeness*** *action*. Depending on the selected options, this action reports errors, such as topics referenced in other topics (but not in the DITA map), broken links, and missing external resources.

However, the execution of the transformation scenario may still fail. Reported errors include:

- `[exec] HHC5010: Error: Cannot open "fileName.chm". Compilation stopped.` - This error occurs when the CHM output file is opened and the transformation scenario cannot rewrite its content. To solve this issue, close the CHM help file and run the transformation scenario again.
- `[exec] HHC5003: Error: Compilation failed while compiling fileName` - Possible causes of this error are:
  - The processed file does not exist. Fix the file reference before executing the transformation scenario again.
  - The processed file has a name that contains space characters. To solve the issue, remove any spacing from the file name and run the transformation scenario again.

DITA Map Ant Transformation Because it Cannot Connect to External Location

The transformation is run as an external Ant process so you can continue using the application as the transformation unfolds. All output from the process appears in the **DITA Transformation** tab.

The HTTP proxy settings are used for the Ant transformation so if the transformation fails because it cannot connect to an external location you can check the Network Connections.

Topic References Outside the Main DITA Map Folder

Referencing to a DITA topic, map or to a binary resource (for example: image) which is located outside of the folder where the main DITA map is located usually leads to problems when publishing the content using the DITA Open Toolkit. The DITA OT does not handle well links to topics which are outside the directory where the published DITA map is found. By default it does not even copy the referenced topics to the output directory.

You have the following options:

1. Create another DITA map which is located in a folder path above all referenced folders and reference from it the original DITA map. Then transform this DITA map instead.
2. Edit the transformation scenario and in the **Parameters** tab edit the **fix.external.refs.com.oxygenxml** parameter. This parameter is used to specify whether the application tries to fix up such references in a temporary files folder before the DITA Open Toolkit is invoked on the fixed references. The fix up has no impact on your edited DITA content. Only "false" and "true" are valid values. The default value is false.

The PDF Processing Fails to Use the DITA OT and Apache FOP

There are cases when publishing DITA content fails when creating a PDF file. This topic lists some common problems and solutions.

- The FO processor cannot save the PDF at the specified target. The console output contains messages like:

```
[fop] [ERROR] Anttask - Error rendering fo file: C:\samples\dita\temp\pdf\oxygen_dita_temp\topic.fo <Failed
to open C:\samples\dita\out\pdf\test.pdf>
Failed to open samples\dita\out\pdf\test.pdf
```

```
.............
[fop] Caused by: java.io.FileNotFoundException: C:\Users\radu_coravu\Desktop\bev\out\pdf\test.pdf
(The process cannot access the file because it is being used by another process)
```

Such an error message usually means that the PDF file is already opened in a PDF reader application. The solution is to close the open PDF before running the transformation.

- One of the DITA tables contains more cells in a table row than the defined number of *colspec* elements. The console output contains messages like:

```
[fop] [ERROR] Anttask - Error rendering fo file:
D:\projects\eXml\samples\dita\flowers\temp\pdf\oxygen_dita_temp\topic.fo
<net.sf.saxon.trans.XPathException: org.apache.fop.fo.ValidationException:
The column-number or number of cells in the row overflows the number of fo:table-columns specified for the
table. (See position 179:-1)>net.sf.saxon.trans.XPathException: org.apache.fop.fo.ValidationException: The
column-number or number of cells in the row overflows the number of fo:table-columns specified for the table.
 (See position 179:-1)
[fop]  at org.apache.fop.tools.anttasks.FOPTaskStarter.renderInputHandler(Fop.java:657)
[fop]  at net.sf.saxon.event.ContentHandlerProxy.startContent(ContentHandlerProxy.java:375)
............
[fop] D:\projects\samples\dita\flowers\temp\pdf\oxygen_dita_temp\topic.fo ->
D:\projects\samples\dita\flowers\out\pdf\flowers.pdf
```

To resolve this issue, correct the *colspec* attribute on the table that caused the issue. To locate the table that caused the issue:

1. Edit the transformation scenario and set the parameter *clean.temp* to *no*.
2. Run the transformation, open the `topic.fo` file in Oxygen XML Editor plugin, and look in it at the line specified in the error message `(See position 179:-1)`.
3. Look around that line in the `XSL-FO` file to find relevant text content which you can use, for example, with the **Find/Replace in Files** action in the **DITA Maps Manager** view to find the original DITA topic for which the table was generated.

- There is a broken link in the generated `XSL-FO` file. The PDF is generated but contains a link that is not working. The console output contains messages like:

```
[fop] 1248 WARN [ main ] org.apache.fop.apps.FOUserAgent - Page 6: Unresolved ID reference
"unique_4_Connect_42_wrongID" found.
```

To resolve this issue:

1. Use the  **Validate and Check for Completeness** action available in the **DITA Maps Manager** view to find such problems.
2. If you publish to PDF using a `DITAVAL` filter, select the same DITAVAL file in the **DITA Map Completeness Check** dialog box.
3. If the  **Validate and Check for Completeness** action does not discover any issues, edit the transformation scenario and set the *clean.temp* parameter to *no*.
4. Run the transformation, open the `topic.fo` file in Oxygen XML Editor plugin, and search in it for the `unique_4_Connect_42_wrongID id`.
5. Look around that line in the `XSL-FO` file to find relevant text content which you can use, for example, with the **Find/Replace in Files** action in the **DITA Maps Manager** view to find the original DITA topic for which the table was generated.

The *TocJS* Transformation Does not Generate All Files for a Tree-Like TOC

The *TocJS* transformation of a DITA map does not generate all the files needed to display the tree-like table of contents. To get a complete working set of output files you should follow these steps:

1. Run the *XHTML* transformation on the same DITA map. Make sure the output gets generated in the same output folder as for the *TocJS* transformation.
2. Copy the content of *`DITA_OT_DIR`*`/plugins/com.sophos.tocjs/basefiles` folder in the transformation's output folder.

3. Copy the *DITA_OT_DIR*`/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"`.

Navigation to the web page was canceled when viewing CHM on a Network Drive

When viewing a CHM on a network drive, if you only see the TOC and an empty page displaying “Navigation to the web page was canceled” note that this is normal behavior. The Microsoft viewer for CHM does not display the topics for a CHM opened on a network drive.

As a workaround, copy the CHM file on your local system and view it there.

Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x

On some Linux systems based on Gnome 3.x (Ubuntu 11.x, 12.x), the main menu of Oxygen XML Editor plugin has alignment issues when you navigate it using your mouse.

This is a known problem caused by Java SE 6 1.6.0_32 and earlier. You can resolve this problem using the latest Java SE 6 JRE from Oracle. To download the latest version, go to *http://www.oracle.com/technetwork/java/javase/downloads/index.html*.

To bypass the JRE bundled with Oxygen XML Editor plugin, go to the installation directory of Oxygen XML Editor plugin and rename or move the `jre` folder. If Oxygen XML Editor plugin does not seem to locate the system JRE, either set the JAVA_HOME environment variable to point to the location where you have installed the JRE, or you can simply copy that folder with the JRE to the installation directory and rename it to `jre` to take the place of the bundled JRE.

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 Editor plugin uses the **Apache Batik** open source library to render SVG images. The **Batik** library only has partial support for SVG 1.2: *http://xmlgraphics.apache.org/batik/dev/svg12.html*.

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

If 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.

Increasing the Memory for the Ant Process

For details about setting custom JVM arguments to the Ant build process see *this section*.

Chapter 22

Glossary

Topics:

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.

Apache Ant

Apache Ant (Another Neat Tool) is a software tool for automating software build processes.

Ant

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.

Bookmap

A bookmap is a specialized *DITA map* used for creating books. A bookmap supports book divisions such as chapters and book lists such as indexes.

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.

DITA_OT_DIR

`DITA_OT_DIR` is the default directory that is specified for your DITA Open Toolkit distribution in the ***Window*** > ***Preferences*** > ***Oxygen XML Editor plugin*** > ***DITA*** *preferences page*.

For example, if you are using DITA 1.8, the default DITA OT directory is:
`[OXYGEN_DIR]/frameworks/dita/DITA-OT`.

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.

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

Named User

Named User is defined as an individual full or part-time employee who is authorized by *You* (the individual or entity who owns the rights to Oxygen XML Editor plugin) to use the software regardless of whether the individual is actively using the software at any given time. To avoid any doubt, Named User licenses cannot be shared amongst multiple individuals and separate Named User licenses must be purchased for each individual user.

A Named User license may not be reassigned to another employee except in the following circumstances:

- (a) Upon termination of the Named User's employment with your company.
- (b) Permanent reassignment of a Named User to a position that does not involve the use of the Software.

For example, suppose Jane has been assigned an Oxygen XML license and she leaves your company. When she leaves, you can simply reassign her license to John, her replacement. In the event that you do reassign the Named User license in accordance with the restrictions above, you do not need to notify Syncro of such a reassignment.

Note: This definition is taken from the Oxygen XML Editor plugin *End User License Agreement*.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X