



Oxygen XML Developer 18.1

User Guide

Contents

Chapter 1: Introduction.....	1
Chapter 2: Getting Started.....	2
What is Oxygen XML Developer plugin.....	3
Getting Familiar with the Layout.....	3
Supported Document Types.....	3
Resources to Help You Get Started Using Oxygen XML Developer plugin	4
Your First Document or Project.....	5
Creating a New Project.....	5
Getting Help.....	6
Help Menu.....	6
Chapter 3: Installation.....	8
Installation Options.....	9
Windows Installation.....	9
Choosing an Installer.....	9
System Requirements.....	9
Eclipse Plugin Installation - Update Site Method.....	10
Eclipse Plugin Installation - Zip Archive Method.....	10
Mac OS X Installation.....	10
Choosing an Installer.....	10
System Requirements.....	10
Eclipse Plugin Installation - Update Site Method.....	11
Eclipse Plugin Installation - Zip Archive Method.....	11
Linux Installation.....	11
Choosing an Installer.....	11
System Requirements.....	12
Eclipse Plugin Installation - Update Site Method.....	12
Eclipse Plugin Installation - Zip Archive Method.....	12
Site-wide deployment.....	13
Licensing.....	13
Choosing a License Type.....	13
Obtaining a License.....	13
Register a Named-User or Subscription License.....	14
Registering a Floating License.....	14
Setting up a License Server.....	16
Setting up an HTTP Floating License Server.....	17
Setting up a TCP Floating License Server Using a 32-bit Windows Installer.....	20
Setting up a TCP Floating License Server Using All-Platforms Distribution.....	22
Transferring or Releasing a License.....	23

Upgrading.....	24
What is Preserved During an Upgrade?.....	24
Upgrading the Eclipse Plugin.....	24
Uninstalling.....	25
Uninstalling the Eclipse plugin.....	25
Chapter 4: Configuration.....	26
Preferences.....	27
Oxygen XML Developer plugin License.....	28
Archive Preferences.....	28
CSS Validator Preferences.....	29
Custom Editor Variables Preferences.....	29
Data Sources Preferences.....	30
DITA Preferences.....	34
Document Type Association Preferences.....	35
Editor Preferences.....	51
Fonts Preferences.....	72
Network Connection Settings Preferences.....	72
Scenarios Management Preferences.....	74
View Preferences.....	74
XML Preferences.....	75
XML Structure Outline Preferences.....	96
Configuring Options.....	96
Customizing Default Options.....	97
Importing/Exporting/Resetting Global Options.....	97
Associating a File Extension with Oxygen XML Developer plugin.....	98
Scenarios Management.....	98
Editor Variables.....	99
Custom Editor Variables.....	104
Localizing of the User Interface.....	104
Chapter 5: Perspectives.....	105
oXygen XML Perspective.....	106
oXygen XSLT Debugger Perspective.....	107
oXygen XQuery Debugger Perspective	107
oXygen DB Perspective	108
Chapter 6: Editing Modes.....	111
Text Editing Mode.....	112
Navigating the Document Content in Text Mode.....	112
Text Mode Views.....	113
Syntax Highlight Depending on Namespace Prefix.....	124
Presenting Validation Errors in Text Mode.....	125
Grid Editing Mode.....	126
Layouts: Grid and Tree.....	127

Grid Mode Navigation.....	127
Bidirectional Text Support in Grid Mode.....	128
Design Editing Mode (XML Schema Diagram Editor).....	129
Navigation in the XML Schema Design Mode.....	130
XML Schema Outline View.....	131
XML Schema Attributes View.....	133
XML Schema Facets View.....	135
XML Schema Palette View.....	136
Chapter 7: Editing Documents.....	137
Working with Unicode.....	138
Opening and Saving Documents with Unsupported Characters.....	138
Inserting Symbols.....	138
Unicode Fallback Font Support.....	140
Creating and Working with Documents.....	140
Creating New Documents and Templates.....	141
Opening Documents.....	147
Saving Documents.....	148
Opening and Saving Remote Documents.....	148
Closing Documents.....	152
Contextual Menu of the Current Editor Tab.....	153
Viewing File Properties.....	153
Using Projects to Group Documents.....	153
Creating a New Project.....	153
Navigator View.....	154
Master Files Support.....	159
Editing XML Documents.....	160
Editing XML Documents in Text Mode.....	161
Editing XML Documents in Grid Mode.....	189
Validating XML Documents.....	192
Finding and Replacing Text in the Current File.....	207
Search and Refactor Actions for IDs and IDREFS.....	209
Associating a Schema to XML Documents.....	211
Working with Modular XML Files in the Master Files Context.....	218
XML Resource Hierarchy/Dependencies View.....	219
Working with XML Catalogs.....	221
Editing Large XML Documents with DTD Entities or XInclude.....	223
Viewing Status Information.....	225
Editor Highlights.....	226
XML Quick Fixes.....	226
Refactoring XML Documents.....	228
Editing XSLT Stylesheets.....	243
Editing XSLT Stylesheets in the Master Files Context.....	243
Validating XSLT Stylesheets.....	244
Content Completion in XSLT Stylesheets.....	245
Syntax Highlighting in XSLT.....	250

XSLT Outline View.....	250
XSLT/XQuery Input View.....	253
XSLT Resource Hierarchy/Dependencies View.....	256
XSLT Component Dependencies View.....	258
Highlight Component Occurrences.....	259
Finding XSLT References and Declarations.....	260
XSLT Stylesheet Component Documentation Support.....	260
Generating Documentation for an XSLT Stylesheet.....	261
XSLT Quick Assist Support.....	267
XSLT Quick Fix Support	269
XSLT Refactoring Actions.....	270
XSLT Unit Test (XSpec).....	272
Editing XML Schemas.....	274
Design Editing Mode (XML Schema Diagram Editor).....	274
Editing XML Schema in Text Editing Mode.....	304
Editing XML Schema in the Master Files Context.....	304
Validating XML Schema Documents.....	304
Content Completion in XML Schema.....	305
XML Schema Outline View.....	305
XML Schema Attributes View.....	307
XML Schema Resource Hierarchy / Dependencies View.....	309
Component Dependencies View for XML Schema.....	311
Highlight Component Occurrences.....	313
Searching and Refactoring Actions in XML Schemas.....	313
Quick Fixes for DTD, XSD, and Relax NG Errors.....	314
XML Schema Quick Assist Support.....	315
Generating Sample XML Files.....	315
Generating Documentation for an XML Schema.....	319
Converting Schema to Another Schema Language.....	327
Converting Database to XML Schema.....	329
Flatten an XML Schema.....	329
XML Schema Regular Expressions Builder.....	331
XML Schema 1.1.....	333
Setting the XML Schema Version.....	334
Editing XQuery Documents.....	335
XQuery Outline View.....	335
Folding in XQuery Documents.....	336
Formatting and Indenting XQuery Documents.....	336
Generating HTML Documentation for an XQuery Document.....	337
Editing WSDL Documents.....	338
Editing WSDL Documents in the Master Files Context.....	338
Validating WSDL Documents.....	339
Content Completion Assistance in WSDL Documents.....	339
WSDL Outline View.....	340
WSDL Resource Hierarchy/Dependencies View in WSDL Documents.....	343
Component Dependencies View in WSDL Documents.....	345
Highlight Component Occurrences in WSDL Documents.....	346

Searching and Refactoring Operations in WSDL Documents.....	346
Quick Assist Support in WSDL Documents.....	348
Generating Documentation for WSDL Documents.....	349
WSDL SOAP Analyzer.....	353
Editing CSS Stylesheets.....	355
Validating CSS Stylesheets.....	356
Content Completion in CSS Stylesheets.....	356
CSS Outline View.....	357
Folding in CSS Stylesheets.....	357
Formatting and Indenting CSS Stylesheets (Pretty Print).....	357
Minifying CSS Stylesheets.....	357
Editing Relax NG Schemas.....	358
Editing Relax NG Schema in the Master Files Context.....	358
Relax NG Schema Diagram Editor.....	359
Validating Relax NG Schema Documents.....	362
Relax NG Outline View.....	362
RNG Resource Hierarchy/Dependencies View.....	364
Component Dependencies View for RelaxNG Schemas.....	366
Searching and Refactoring Actions in RNG Schemas.....	368
Quick Fixes for DTD, XSD, and Relax NG Errors.....	369
RNG Quick Assist Support.....	369
Configuring a Custom Datatype Library for a RELAX NG Schema.....	370
Editing NVDL Schemas.....	371
NVDL Schema Diagram.....	371
NVDL Outline View.....	373
Validating NVDL Schema Documents.....	373
Component Dependencies View for NVDL Schemas.....	374
Searching and Refactoring Actions in NVDL Schemas.....	375
Editing JSON Documents.....	376
Editing JSON Documents in Text Mode.....	376
Editing JSON Documents in Grid Mode.....	378
Validating JSON Documents.....	379
JSON Outline View.....	379
XML to JSON Converter.....	379
Editing StratML Documents.....	380
Editing XLIFF Documents.....	381
Editing JavaScript Documents.....	381
JavaScript Editing Actions.....	381
Validating JavaScript Files.....	383
Content Completion in JavaScript Documents.....	383
JavaScript Outline View.....	384
Editing XProc Scripts.....	385
Editing Schematron Schemas.....	386
Editing Schematron Schema in the Master Files Context.....	387
Validating Schematron Documents.....	387
Presenting Schematron Validation Issues.....	387
Content Completion in Schematron Documents.....	388

XML Schema or RELAX NG with Embedded Schematron Rules.....	389
Schematron Outline View.....	390
Schematron Resource Hierarchy/Dependencies View.....	391
Highlight Component Occurrences in Schematron Documents.....	393
Searching and Refactoring Operations in Schematron Documents.....	393
Quick Assist Support in Schematron Documents.....	395
Editing Schematron Quick Fixes.....	395
Schematron Quick Fixes (SQF).....	396
Defining Schematron Quick Fixes.....	396
Validating Schematron Quick Fixes.....	401
Content Completion in SQF.....	401
Highlight Quick Fix Occurrences in SQF.....	402
Searching and Refactoring Operations in SQF.....	402
Embed Schematron Quick Fixes in Relax NG or XML Schema.....	403
Integrating SQF in a Framework.....	404
Editing XHTML Documents.....	404
Editing Markdown Documents.....	405
Markdown Editor.....	405
Creating New Markdown Documents.....	406
Actions Available in the Markdown Editor.....	407
Syntax Highlighting in the Markdown Editor.....	408
Automatic Validation in Markdown Documents.....	409
Working with Markdown Documents in DITA.....	409
Markdown Editor Syntax Rules and Specifications.....	409
Spell Checking.....	418
Spell Check Dictionaries and Term Lists.....	420
Learned Words.....	423
Ignored Words (Elements).....	423
Automatic Spell Check.....	424
Spell Check Multiple Files.....	424
Handling Read-Only Files.....	425
XML Digital Signatures.....	425
Digital Signatures Overview.....	425
Certificates.....	427
Canonicalizing Files.....	427
Signing Files.....	428
Verifying Signature.....	430
Example of How to Digitally Sign XML Files or Content.....	430
Chapter 8: Predefined Document Types.....	432
Predefined Document Types (Frameworks).....	433
DocBook 4 Document Type.....	434
DocBook 5 Document Type.....	440
DocBook Targetset Document Type.....	447
DITA Topics Document Type.....	447
DITA Map Document Type.....	448

XHTML Document Type.....	471
TEI ODD Document Type.....	472
TEI P4 Document Type.....	472
TEI P5 Document Type.....	473
JATS Document Type.....	474
EPUB Document Type.....	475
 Chapter 9: Publishing.....	 477
Transformation Scenarios.....	478
Built-in Transformation Scenarios.....	478
Creating New Transformation Scenarios.....	509
Editing a Transformation Scenario.....	543
Duplicating a Transformation Scenario.....	545
Configure Transformation Scenario(s) Dialog Box.....	545
Apply Batch Transformations.....	547
Sharing Transformation Scenarios.....	547
Transformation Scenarios View.....	548
Debugging PDF Transformations.....	550
Configuring Calabash with XEP	550
Integration of an External XProc Engine.....	551
XSLT Processors.....	551
XSL-FO Processors.....	554
WebHelp System Output.....	557
WebHelp Responsive System	558
WebHelp Classic System.....	602
WebHelp Classic Mobile System.....	634
Using the Oxygen XML WebHelp Plugin to Automate Output.....	653
 Chapter 10: Querying Documents.....	 664
Working with XPath Expressions.....	665
XPath/XQuery Builder View.....	665
XPath Expression Results View.....	667
XPath and XML Catalogs.....	668
XPath Prefix Mapping.....	669
Working with XQuery.....	669
XQuery Syntax Highlights and Content Completion.....	669
XQuery Outline View.....	670
XPath/XQuery Builder View.....	671
XSLT/XQuery Input View.....	673
XQuery Validation.....	676
Transforming XML Documents Using XQuery.....	677
 Chapter 11: Working with Archives.....	 681
Browsing Archives.....	682
Working with Archive Files.....	683

Creating an Archive.....	684
Editing and Saving Files Inside an Archive.....	685
Migrating Archives to DITA or TEI.....	685
Chapter 12: Databases and CMS.....	686
Working with Databases.....	687
Data Source Explorer View.....	687
Table Explorer View.....	688
Database Connection Support.....	690
WebDAV Connections.....	729
SQL Execution Support.....	731
XQuery and Databases.....	733
Content Management System (CMS) Integration.....	737
Integration with Documentum (CMS) (deprecated).....	738
Integration with Microsoft SharePoint.....	743
Chapter 13: Importing Data.....	747
Import from Text Files.....	748
Import from MS Excel Files.....	749
Import Data from MS Excel 2007 or Newer.....	751
Import Database Data as an XML Document.....	751
Import from HTML Files.....	754
Import Content Dynamically.....	754
Chapter 14: XSLT and XQuery.....	757
Debugger Layout.....	758
Control Toolbar.....	758
Debugging Information Views.....	760
Multiple Output Documents in XSLT 2.0 and XSLT 3.0.....	769
Working with the XSLT / XQuery Debugger.....	769
Steps in a Typical Debugging Process.....	769
Using Breakpoints.....	770
Identify the XSLT / XQuery Expression that Generated Particular Output.....	771
Debugging Java Extensions.....	772
Supported Processors for XSLT / XQuery Debugging.....	773
Performance Profiling of XSLT Stylesheets and XQuery Documents.....	773
XSLT/XQuery Performance Profiling Overview.....	773
Working with XSLT/XQuery Profiler.....	773
Chapter 15: Extending Oxygen XML Developer plugin Using the SDK.....	776
Extension points for Oxygen XML Developer plugin.....	777
Chapter 16: Tools.....	778
XML Refactoring.....	779

Predefined Refactoring Operations.....	781
Custom Refactoring Operations.....	785
Storing and Sharing Refactoring Operations.....	793
Localizing XML Refactoring Operations.....	793
Generate Sample XML Files.....	794
Schema Tab (Generate Sample XML Files Tool).....	794
Options Tab (Generate Sample XML Files Tool).....	796
Advanced Tab (Generate Sample XML Files Tool).....	798
Generate/Convert Schema.....	798
Convert DB Structure to XML Schema.....	800
XML to JSON.....	801
Generate Documentation.....	802
XML Schema Documentation.....	802
XSLT Stylesheet Documentation.....	805
XQuery Documentation.....	807
WSDL Documentation.....	808
Canonicalize.....	811
Sign.....	812
Verify Signature.....	813
WSDL SOAP Analyzer.....	813
Composing a SOAP Request.....	813
Testing Remote WSDL Files.....	815
UDDI Registry Browser.....	815
XML Schema Regular Expressions Builder.....	816

Chapter 17: Common Problems.....819

Performance Problems and Solutions.....	820
Out of Memory on External Processes.....	820
Performance Issues with Large Documents.....	820
Misc Problems and Solutions.....	820
'Address Family Not Supported by Protocol Family; Connect' Error.....	820
Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x.....	821
Compatibility Issue Between Java and Certain Graphics Card Drivers.....	821
Damaged File Associations on OS X.....	821
Details to Submit in a Request for Technical Support Using the Online Form.....	822
DITA Map Transformation Fails (Cannot Connect to External Location).....	822
DITA PDF Transformation Fails.....	822
DITA to CHM Transformation Fails.....	823
Image Appears Stretched Out in the PDF Output.....	823
MSXML 4.0 Transformation Issues.....	823
Navigation to the web page was canceled when viewing CHM on a Network Drive.....	824
Oxygen XML Developer plugin Takes Several Minutes to Start.....	824
PDF Processing Fails to Use the DITA OT and Apache FOP.....	824
Signature Verification Failed Error on a Resource from Documentum.....	825
SVG 1.2 Rendering Issues.....	825
Syntax Highlight Not Available in Eclipse Plugin.....	825

<i>TocJS</i> Transformation Does not Generate All Files for a Tree-Like TOC.....	826
Topic References Outside the Main DITA Map Folder.....	826
XSLT Debugger Is Very Slow.....	826
 Chapter 18: Glossary.....	 828
Index.....	831
 Copyright.....	 843

Chapter 1

Introduction

Welcome to the User Manual of Oxygen XML Developer plugin 18.1.

Oxygen XML Developer 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 offers developers a powerful *Integrated Development Environment* and the intuitive *Graphical User Interface* of Oxygen XML Developer plugin is easy to use and provides robust functionality for content editing, project management, and validation of structured mark-up sources. Coupled with *XSLT* and *FOP* transformation technologies, Oxygen XML Developer plugin offers support 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 Developer 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:

- [What is Oxygen XML Developer plugin](#)
- [Getting Familiar with the Layout](#)
- [Supported Document Types](#)
- [Resources to Help You Get Started Using Oxygen XML Developer plugin](#)
- [Your First Document or Project](#)
- [Getting Help](#)

This section provides a variety of resources to help you get the most out of the application. Typically, the first step of getting started with Oxygen XML Developer plugin would be to install the software. For detailed information about that process, see the [Installation chapter](#).

After installation, when you launch Oxygen XML Developer 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.

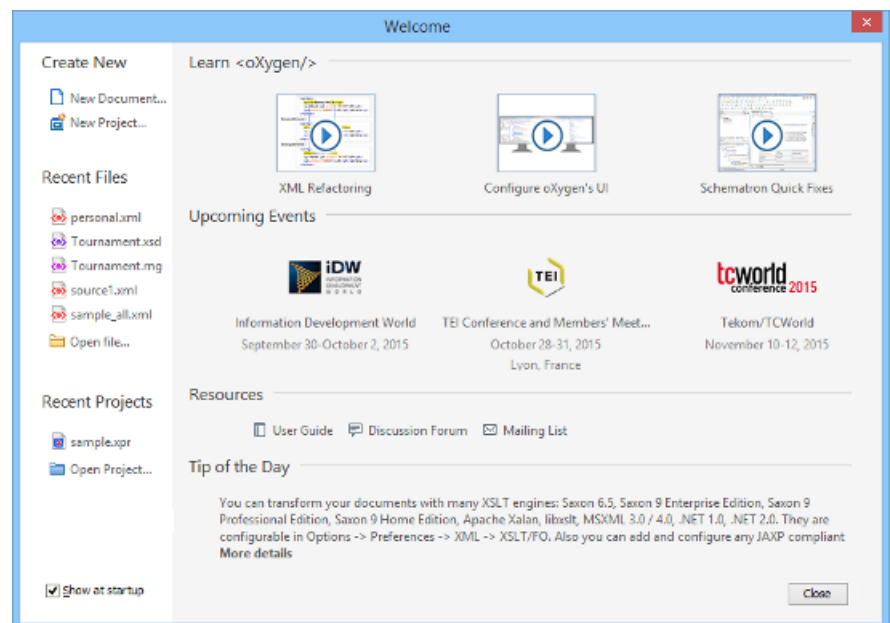


Figure 1: Welcome Dialog Box

If you do not want it to be displayed every time you launch Oxygen XML Developer plugin, disable the **Show at startup** option in the bottom-left corner of the dialog box. To display it any time, go to **Help > Welcome**.

What is Oxygen XML Developer plugin

Oxygen XML Developer 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 Developer 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 Developer plugin, see the [oXygen Website](#).

Getting Familiar with the Layout

Oxygen XML Developer 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 Developer plugin. Most of the menus are common for all types of documents, but Oxygen XML Developer 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 Developer 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 suit 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 Developer 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 Developer plugin for the first time. Oxygen XML Developer 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.

Supported Document Types

You can use the main editing pane in Oxygen XML Developer plugin to edit a large variety of document types. You can see the type of document association by the special icons displayed in the tabs of the editor title bar.

The supported document types include the following:

-  - 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
-  - XProc scripts
-  - SQL documents
-  - JSON documents
-  - Ant build scripts
-  - Markdown documents

Resources to Help You Get Started Using Oxygen XML Developer plugin

Configuring Oxygen XML Developer plugin

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

- See the [Configuring Oxygen section](#) for details on the various ways that you can configure the application and its features.

Video Tutorials

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

- Go to the [Oxygen XML Developer plugin Videos Page](#) to see the list of video tutorials and webinars.

Oxygen XML Developer plugin Documentation

The Oxygen XML Developer 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 Section](#) - Provides information about the **Text** editor.
- [Editing Documents Section](#) - Includes information about editing numerous different types of documents.
- [WebHelp System Output Section](#) - Provides information about the WebHelp system that can be used for publishing content.

- [Importing Data Section](#) - Provides information about importing data from text files, MS Excel files, database data, and HTML files.

Sample Documents

Your installation of Oxygen XML Developer 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 Developer plugin. You will find files and folders for various 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 document 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 Developer plugin:

- See the [Oxygen XML Developer 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 Developer 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 Developer 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 Developer 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 Developer 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 Developer 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 Developer plugin.

Your First Document or Project

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

Creating a New Project

Oxygen XML Developer 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 your 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 more information on managing projects and their content, see [Navigator View](#) on page 113.

Related information

[Using Projects to Group Documents](#) on page 153

Getting Help

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

- [The Oxygen XML Developer plugin Support Section of the Website](#)
- [The Oxygen XML Developer plugin Forum](#)
- [The Oxygen XML Developer 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.

Help Menu

The Oxygen XML Developer plugin **Help** menu provides various resources to assist you with your tasks.

This menu includes the following actions or options:

Welcome

This option opens the **Welcome** screen that includes some resources to assist you with using Oxygen XML Developer plugin.

Help Contents

Use this action to open a dialog box that presents Eclipse help topics and it includes a section that is specific to Oxygen XML Developer plugin. Also, you can use the **F1** key to open a **Help** view that presents a section in the User Manual that is appropriate for the context of the current cursor position.

Report <Oxygen/> problem

You can use this option to open a dialog box that allows you to write the description of a problem that was encountered while using the application. You can also select additional information to be sent to the technical support team in the five tabs:

- **General info** - You can edit your contact details in case you want to be contacted for further details or to be notified of a resolution.
- **Class Loader URLs** - You can choose whether or not to include the listed *Class Loader URLs* with your report.

- **System properties** - You can choose whether or not to include the listed system property details with your report.
-  **Tip:** You are able to change the URL where the reported problem is sent by using the `com.oxygenxml.report.problems.url` system property. The report is sent in XML format through the `report` parameter of the POST HTTP method.
- **Plugins** - You can choose whether or not to include details about your installed plugins with your report.
- **Frameworks** - You can choose whether or not to include details about your installed frameworks with your report.

Support Tools > Randomize XML text content

Use this action when you need to send samples to the oXygen support team and you want to keep the text content confidential. It opens a dialog box that allows you to select the resources for which the text content will be randomized. You can then save the resources and send them to the oXygen support team without fear of compromising sensitive or private data.

 **Warning:** Before using this action, you need to copy the XML resources and save them in a separate folder. Otherwise, you may lose your original content.

About Eclipse

Use this option and then click the Oxygen XML Developer plugin icon to open a dialog box that contains information about Oxygen XML Developer plugin and the installed version.

Related information

[Details to Submit in a Request for Technical Support Using the Online Form](#) on page 822

Chapter

3

Installation

Topics:

- [*Installation Options for Oxygen XML Developer plugin*](#)
- [*Install Oxygen XML Developer plugin on Windows*](#)
- [*Install Oxygen XML Developer plugin on Mac OS X*](#)
- [*Install Oxygen XML Developer plugin on Linux*](#)
- [*Group Deployment*](#)
- [*Obtaining and Registering a License Key for Oxygen XML Developer plugin*](#)
- [*Setting Up a Floating License Server*](#)
- [*Transferring a License Key*](#)
- [*Upgrading Oxygen XML Developer plugin*](#)
- [*Uninstalling Oxygen XML Developer plugin*](#)

This chapter includes information about installing and licensing Oxygen XML Developer plugin on various platforms. Oxygen XML Developer plugin is available on Windows, Linux, and Mac OS X and there are a variety of methods and options for installing and running Oxygen XML Developer plugin on your system or server. This section also includes information about registering, transferring, or releasing licenses, upgrading, installing add-ons, and uninstalling.

Installation Options for Oxygen XML Developer plugin

Choosing an installer

You have a choice of installers;

- The Update Site installer
- The Zip archive installer

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

Choosing a license option

You must [obtain and register a license](#) key to run Oxygen XML Developer plugin.

You can choose from two kinds of license:

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

Upgrading, transferring, and uninstalling.

You can also [upgrade](#) Oxygen XML Developer plugin, [transfer a license](#), or [uninstall](#) Oxygen XML Developer plugin.

Getting help with installation

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

Install Oxygen XML Developer plugin on Windows

Choosing an Installer

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

- [Update Site method](#)
- [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 Developer plugin uses the same Java Virtual Machine as the copy of Eclipse it is running in.

Eclipse Plugin Installation - Update Site Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.6. 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/Developer/Eclipse/site.xml` into the **Location** field of the **Add Site** dialog box.
5. Click **OK**.
6. Select the **Oxygen XML Developer plugin** checkbox.
7. Click **Next >** and continue with the rest of the installation wizard.
8. Restart Eclipse when prompted.
9. Verify that Oxygen XML Developer plugin is installed correctly by creating a new XML Project. Go to **File > New > Other** and choose **Oxygen XML Developer 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 Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Eclipse Plugin Installation - Zip Archive Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.6. 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 Developer plugin is installed correctly by creating a new XML Project. Go to **File > New > Other** and choose **Oxygen XML Developer 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 Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Install Oxygen XML Developer plugin on Mac OS X

Choosing an Installer

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

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

System Requirements

System requirements for a Mac OS X install:

Operating system

OS X version 10.8 64-bit or later

CPU

- Minimum - Intel-based Mac, 1 GHz

- Recommended - Dual Core class processor

Memory

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

Storage

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

Java

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

Eclipse Plugin Installation - Update Site Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.6. 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/Developer/Eclipse/site.xml` into the **Location** field of the **Add Site** dialog box.
5. Click **OK**.
6. Select the **Oxygen XML Developer plugin** checkbox.
7. Click **Next >** and continue with the rest of the installation wizard.
8. Restart Eclipse when prompted.
9. Verify that Oxygen XML Developer plugin is installed correctly by creating a new XML Project. Go to **File > New > Other** and choose **Oxygen XML Developer 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 Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Eclipse Plugin Installation - Zip Archive Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.6. 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 Developer plugin is installed correctly by creating a new XML Project. Go to **File > New > Other** and choose **Oxygen XML Developer 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 Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Install Oxygen XML Developer plugin on Linux

Choosing an Installer

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

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

System Requirements

System requirements for a Linux install:

Operating system

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

CPU

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

Memory

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

Storage

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

Java

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

Eclipse Plugin Installation - Update Site Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.6. 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/Developer/Eclipse/site.xml` into the **Location** field of the **Add Site** dialog box.
5. Click **OK**.
6. Select the **Oxygen XML Developer plugin** checkbox.
7. Click **Next >** and continue with the rest of the installation wizard.
8. Restart Eclipse when prompted.
9. Verify that Oxygen XML Developer plugin is installed correctly by creating a new XML Project. Go to **File > New > Other** and choose **Oxygen XML Developer 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 Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Eclipse Plugin Installation - Zip Archive Method

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.6. 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 Developer plugin is installed correctly by creating a new XML Project. Go to **File > New > Other** and choose **Oxygen XML Developer 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 Developer plugin installed on your computer, you will not be prompted for a license key for the Eclipse version. The existing license key will be used automatically.

Group Deployment

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

Creating custom default options

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

Creating default project files

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

Shared project files

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

Using floating licenses

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

Obtaining and Registering a License Key for Oxygen XML Developer plugin

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

For demonstration and evaluation purposes, a time limited license is available upon request at <https://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.

Choosing a License Type

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

1. A **Named-User License** may be used by a single *Named User* on one or more computers. Named-user licenses are not transferable to a new *Named User*. If you order multiple named-user licenses, you will receive a single license key good for a specified number of named users. It is your responsibility to keep track of the named users that each license is assigned to.
2. A **Floating License** may be used by any user on any machine. However, the total number of copies of Oxygen XML Developer plugin in use at one time must not be more than the number of floating licenses available. A user who runs two different distributions of Oxygen XML Developer plugin (for example, Standalone and Eclipse Plugin) at the same time on the same computer, consumes a single floating license.
3. A **Subscription** license that allows you to use the application for a specific period of time (either 6 months or 1 year). This type of license is user-based and is covered by a Support and Maintenance Pack, which means that during the subscription period you will get free upgrades to all major and minor releases and priority technical support.

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

Obtaining a License

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

- You can purchase one or more licenses from the Oxygen XML Developer plugin website at <https://www.oxygenxml.com/buy.html>. A license key will be sent to you by email.
- If your company or organization has already 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 <https://www.oxygenxml.com/registerCode.html>. A license key will be sent to you by email.
- If you want to evaluate the product, you can obtain a trial license key for 30 days from the Oxygen XML Developer plugin website at <https://www.oxygenxml.com/register.html>.

Register a Named-User or Subscription License

To register a *Named-User License* or *Subscription License* on a machine owned by the *Named User*, follow these steps:

1. Purchase a license from the [Oxygen XML Developer plugin website](#). You will receive an email that contains your license key.
2. Save a backup copy of your email message that contains the new license key.
3. Open an XML document in the Oxygen XML Developer plugin.
If this is a new install of Oxygen XML Developer plugin, the registration dialog box is displayed. If the registration dialog box is not displayed, go to **Window (Eclipse on Mac OSX)** and choose **Preferences > Oxygen XML Developer plugin** and click the **Register** button.

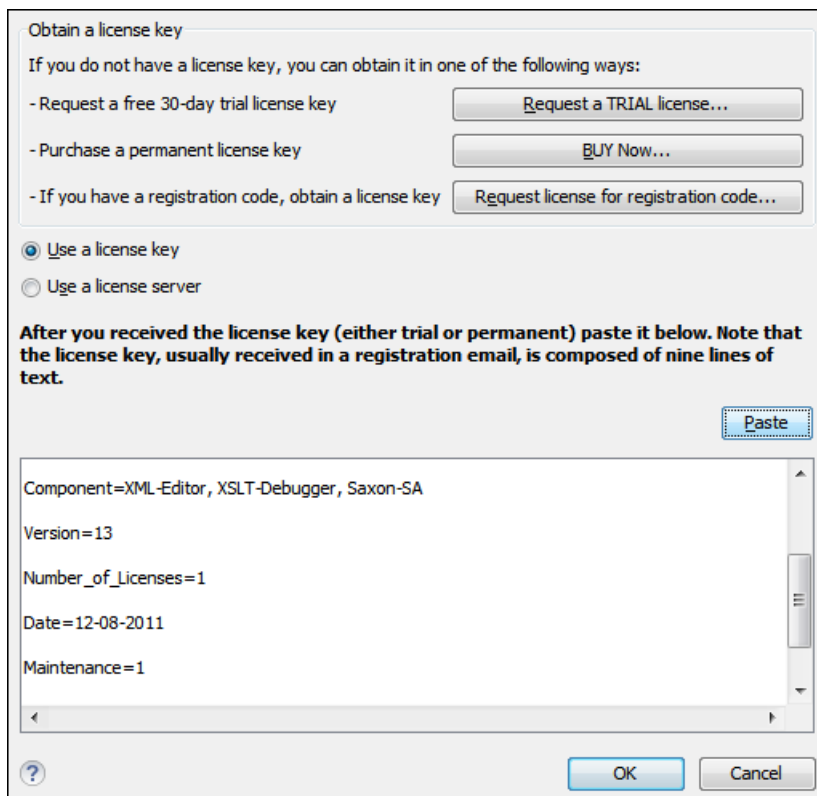


Figure 2: License Registration Dialog Box

4. Select **Use a license key** as licensing method.
5. Paste your license key into the registration dialog box. The license key is composed of nine lines of text between two text markers.
6. 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 the machines that share the pool of floating licenses are on multiple 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 [TCP](#) or [HTTP](#) server as the registration process is different for each.
- 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](#). Oxygen XML Developer plugin will use your connection to a local area

network, without additional notice, to automatically connect to other running instances of Oxygen XML Developer plugin. These connections may transmit your IP address to the local network.



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



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

Request a Floating License from a TCP License Server

Use this procedure if your company uses an Oxygen XML Developer plugin TCP license server and the license server has already been set up by your server administrator:

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 **TCP server** as server type.
6. In the *Host* field, enter the host name or IP address of the license server.
7. In the *Port* field, enter the port number used to communicate with the license server.
8. Click the **OK** button.

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

Related tasks

[Setting up a TCP Floating License Server Using a 32-bit Windows Installer](#) on page 20

Setting up the TCP floating license server as a Windows process.

[Setting up a TCP Floating License Server Using All-Platforms Distribution](#) on page 22

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

Request a Floating License from an HTTP License Server

Use this procedure if your company uses an Oxygen XML Developer plugin HTTP license server and the license server has already been set up by your server administrator:

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

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

Related information

[Setting up an HTTP Floating License Server](#) on page 17

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 Developer plugin running on your machine are using your floating license.
- You register a *Named User* license with your copy of Oxygen XML Developer plugin, and no other copies of Oxygen XML Developer plugin running on your machine are using your floating license.

To release a floating license on demand, follow these steps:

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 **Use a license key** option is selected.
4. Click **OK**.
A dialog box is displayed asking if you want to reset your license key.
5. Select between:
 - **Use the last one** - Falls back to your previous license key. Use this option if you want to release a floating license and revert to a *Named User* license.
 - **Reset** - Removes your license key from your user account on the current computer.

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

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 Developer 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 Developer plugin [preferences directory](#) to the `lib` sub-folder of the installation folder on each installation to be registered.

Setting Up a Floating License Server

Installing a License Server to Manage Floating Licenses

If you are using floating licenses for Oxygen XML Developer plugin, you must set up an Oxygen XML Developer plugin floating license server. A floating license server can be installed as one of the following:

- An [HTTP server](#). This is the recommended method.
- A [TCP server](#) (deprecated).



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

Activating Floating License Keys

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

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

Split or Combine License Keys to Work with Your License Servers

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

Setting up an HTTP Floating License Server

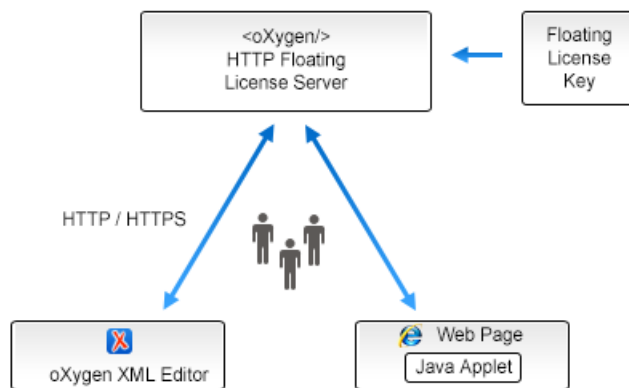


Figure 3: Floating License Server (HTTP Server)

The Oxygen XML Developer plugin license server is available in several distributions, tailored for covering a variety of deployment configurations:

- **Windows installer** - Easy-to-use Windows installation wizard. Requires elevated permissions to run it.
- **All-platform distribution** - Script-based deployment that does not require elevated permissions to run it. Provides scripts for Windows, Mac, and Linux.
- **Web Archive (WAR) distribution** - Provides more flexibility in your deployment configuration, but it requires an existing HTTP server (such as Apache Tomcat).

Installation Steps for the HTTP License Server Installer Distribution for Windows

1. Download the HTTP license server installer from the [Oxygen XML Developer plugin website](#).
2. Run the installer and follow the on-screen instructions.
3. You need to configure two sets of credentials:
 - a. **Administrator credentials** - used for accessing the Oxygen XML Developer plugin license server administrative interface. Optionally you can choose to change the standard 8080 port.
 - b. **Standard user credentials** - used by an Oxygen XML Developer plugin application to connect to the license server.
4. Optionally you can choose to install the server as a Windows service. In this case, you can choose the name of the Windows service.

Installation Steps for the HTTP License Server All-Platform Distribution

1. Download the HTTP license server all-platform archive from the [Oxygen XML Developer plugin website](#).

2. Unpack the archive.
3. Run the license server scripts suitable for your operating system (`licenseServer.bat` for Windows or `licenseServer.sh` for Linux and Mac).



Note: To specify a different port (other than the default 8080), you can pass the new port number as an argument to the scripts (for example, `licenseServer.bat 8082`).

4. On the first run, you will be prompted to set two sets of credentials:
 - a. **Administrator credentials** - used for accessing the Oxygen XML Developer plugin license server administrative interface.
 - b. **Standard user credentials** - used by an Oxygen XML Developer plugin application to connect to the license server.

Installation Steps for the HTTP License Server WAR Distribution

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 HTTP license server **Web ARchive (.war)** from the [Oxygen XML Developer plugin website](#).
3. Configure two Tomcat users:
 - a. One user with the role *user*, used by an Oxygen XML Developer plugin application to connect to the license server. In the subsequent example, this user name is **John**.
 - b. Another user with the roles *admin* and *manager-gui*, used for accessing the Oxygen XML Developer plugin license server administrative interface and the Tomcat management interface. In the subsequent example, this user name is **Mary**.

A typical way to achieve this is to edit the `tomcat-users.xml` file from your Tomcat installation (if using a Tomcat *zip/tar.gz* distribution, by default this configuration file is found in the `/TomcatInstallFolder/conf/` directory). After adding the two users, the configuration file might look like this:

```
<tomcat-users xmlns="http://tomcat.apache.org/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
  version="1.0">
  <!-- ... other user and role definitions ... -->
  <role rolename="user"/>
  <role rolename="admin"/>
  <role rolename="manager-gui"/>
  <user username="John" password="user_pass" roles="user"/>
  <user username="Mary" password="admin_pass" roles="admin,manager-gui"/>
</tomcat-users>
```

4. Go to the Tomcat Web Application Manager page and log-in with the user you configured with the *manager-gui* role (**Mary** in the example above). In the **WAR file to deploy** section, choose the WAR file and click the **Deploy** button. The `oxygenLicenseServlet` application is now up and running, but the license key is not yet registered.
5. Go to the `oxygenLicenseServlet` application page by clicking the `oxygenLicenseServlet` link in the manager page. You will need to authenticate with the user configured with the *admin* role (Mary in our example).
6. Activate the floating license key. This process involves binding your license key to your license server deployment. Once the process is completed you cannot activate the floating license with another license server. Follow these steps to activate the license:

- a. Access the HTTP license server 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 floating license key into the form and press **Submit**. The browser used in the activation process needs to have Internet access.

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



Note: If, for some reason, your browser does not take you to this activation form, refer to the [Manual Activation Procedure](#).

- c. Press **Activate**.

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

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

Manual License Activation Procedure

1. Access the HTTP license server 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 HTTP server.

Automatic Subscription Renewal

If the HTTP license server is configured with a subscription license, then the license server will automatically check to see if a new subscription license was purchased and will automatically download and install it for you.

 **Important:** This automatic checking procedure implies a connection to a web service located at [oxygenxml.com](http://www.oxygenxml.com). You can deactivate this automatic behavior by disabling the **Automatically check for subscription renewal** option from the main management page of the license server.

If the automatic renewal process fails, you can try either of the following possible solutions:

- If your server uses a proxy to connect to the Internet, go to the main management page of the license server and configure the proxy settings by clicking the **Proxy settings** link in the **Management tasks** section.
- [Manually replace the floating license key](#).

Upgrading Your HTTP Floating License Server

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

Follow this procedure:

1. Access the license server 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 server.
4. Go to Oxygen XML Developer plugin website and [download the HTTP license server](#).
5. Deploy the downloaded license server.
6. Access the license server 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 an HTTP Floating License Server

The following procedure assumes that your Oxygen XML Developer plugin HTTP floating license server contains a previously [activated license key](#) and provides instructions for replacing it with another one. The goal of the procedure is to allow you to activate and configure the new license key without any downtime.

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 HTTP floating license server with a new one, follow these steps:

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

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



Note: If for some reason your browser does not take you to this activation form, refer to the [Manual Activation Procedure](#).

5. Press **Activate**.

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

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 Developer plugin runs
 - MAC address of the computer where Oxygen XML Developer plugin runs



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

Setting up a TCP Floating License Server Using a 32-bit Windows Installer

Setting up the TCP floating license server as a Windows process.

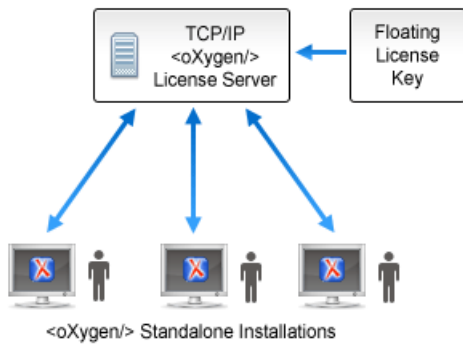


Figure 4: TCP Floating License Server (Process in Windows)

Installation Steps

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

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

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

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

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

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

3. Manually add the `oXygenLicenseServer.exe` file in the Windows Firewall list of exceptions. Go to **Control Panel > System and Security > Windows Firewall > Allow a program or feature through Windows Firewall > Allow another program** and browse for `oXygenLicenseServer.exe` from the Oxygen XML Developer 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 Developer plugin version and you want to have all of them managed by the same server, or if you have a multiple-user floating license and you want to split it between two or more license servers, please contact support@oxygenxml.com and ask for a new license key.

Upgrading Your TCP Floating License Server

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

Follow this procedure:

1. Go to the Oxygen XML Developer 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 TCP Floating License Server

The following procedure assumes that your Oxygen XML Developer plugin TCP 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 server 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 TCP floating license server.

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>.

Windows Service Reports 'Error 1067: 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 TCP Floating License Server Using All-Platforms Distribution

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

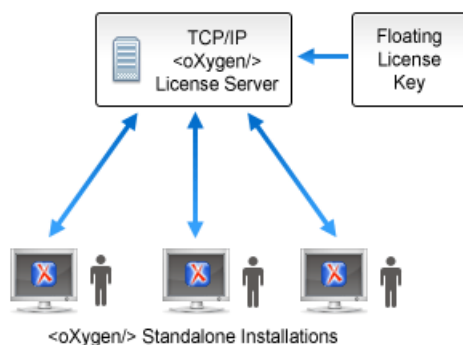


Figure 5: TCP Floating License Server (All-Platforms Distribution)

Installation Steps

1. Ensure that a Java runtime version 6 or later is installed on the server machine.

2. Download the license server installation kit for your platform from the [Oxygen XML Developer plugin website](#).

3. Unzip the installation kit into a new folder.

4. Start the server using the startup script from a command line console.

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

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

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

```
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 TCP Floating License Server

The goal of the following procedure is to help you minimize the downtime generated when you upgrade the Oxygen XML Developer plugin TCP 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 this:

```
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 Developer 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 TCP Floating License Server

The following procedure assumes that your Oxygen XML Developer plugin TCP 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 HTTP license server 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 a License Key

If you want to transfer your Oxygen XML Developer plugin license key to another computer (for example, if you are disposing of your old computer or transferring it to another person), 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 **Use a license key** option is selected.
4. Click **OK**.
A dialog box is displayed asking if you want to reset your license key.
5. Select between:
 - **Use the last one** - Falls back to your previous license key, if applicable.
 - **Reset** - Removes your license key from your user account on the current computer.

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

Upgrading Oxygen XML Developer plugin

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

What is Preserved During an Upgrade?

When you install a new version of Oxygen XML Developer plugin, some data is preserved and some is overwritten. If there is a previous version of Oxygen XML Developer plugin already installed on your computer, it can coexist with the new one, which means you 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 Developer plugin](#).
2. Download and install the new version using the appropriate instructions for your platform and the installation method you chose.
3. Restart the Eclipse platform.
4. Start the Oxygen XML Developer plugin to ensure that the application can start and that your license is recognized by the upgrade installation.
5. If you are upgrading from a minor version to a major version (for example, from 16.1 to 17.0) and you did not purchase a Maintenance Pack that covers the new major version (17.0), you will need to enter a new license for the new version (17) into the registration dialog box that is displayed when the plugin is started.

Uninstalling Oxygen XML Developer plugin

Uninstalling the Eclipse plugin



CAUTION:

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

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

Chapter

4

Configuring Oxygen XML Developer plugin

Topics:

- [*Preferences*](#)
- [*Configuring Options*](#)
- [*Associating a File Extension with Oxygen XML Developer plugin*](#)
- [*Scenarios Management*](#)
- [*Editor Variables*](#)
- [*Localizing of the User Interface*](#)

This chapter presents all the user preferences and options that allow you to configure various features and aspects of the application itself. It also includes information about storing and sharing options, importing and exporting options or scenarios, customizing system properties, setting startup parameters, and the editor variables that are available for customizing user-defined commands..

Preferences

You can configure Oxygen XML Developer plugin options using the **Preferences** dialog box.

To open the preferences dialog box, go to **Window (Eclipse on Mac OSX)** and choose **Preferences > Oxygen XML Developer plugin**.

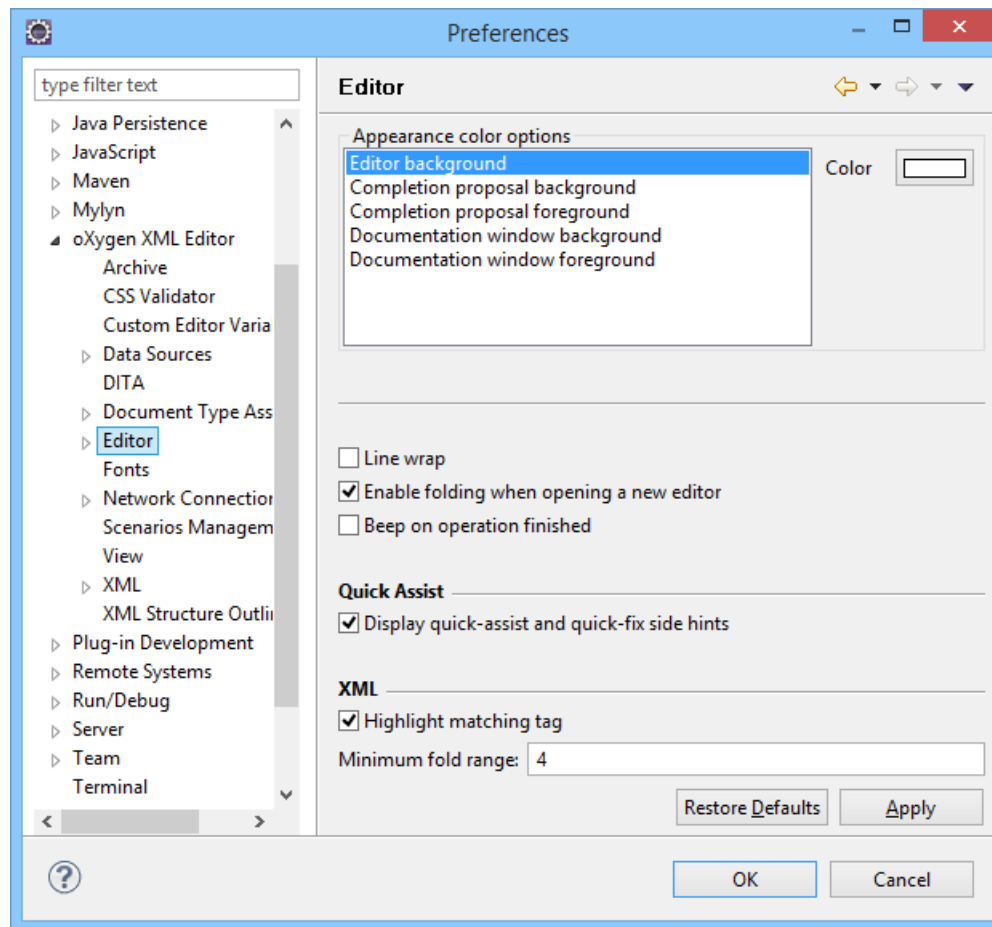


Figure 6: Eclipse Preferences Dialog Box

Press  or **F1** for help on any preferences page.

You can restore options to their default values by pressing the **Restore Defaults** button, available in each preferences page.

A filtered version of the **Preferences** dialog box is available by selecting **Options** from the contextual menu in the editor. It displays an appropriate preferences page according to the context where the action was invoked and filters the tree on the left according to where the preference page is located in the hierarchy.

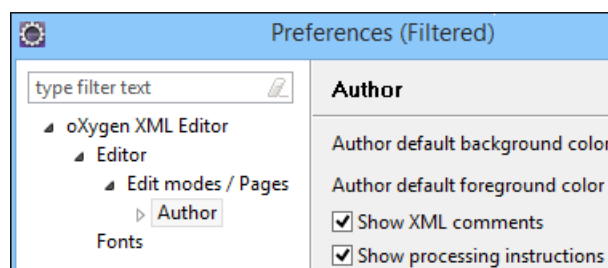


Figure 7: Eclipse Preferences Dialog Box - Filtered Version

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 Developer plugin License

To configure the license options, [open the Preferences dialog box](#). This preferences page presents the details of the license key that enables the Oxygen XML Developer plugin, such as registration name, category and number of purchased licenses, encrypted signature of the license key. Clicking the **Register** button opens the Oxygen XML Developer plugin **License** dialog box that allows you to insert a new license key.

Archive Preferences

To configure *Archive* preferences, [open the Preferences dialog box](#) and go to **Archive**.

The following options are available in the **Archive** preferences page:

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 Developer plugin. You can use the **Edit** button at the bottom of the table to edit an existing mapping or the **New** button to create a new one and associate your own list of extensions to an archive format.

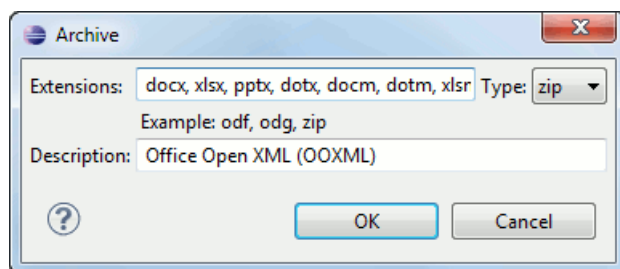


Figure 8: Edit Archive Extension Mappings



Important: You have to restart Oxygen XML Developer plugin after removing an extension from the table for that extension to not be recognized as an archive extension.

Store Unicode file names in Zip archives

Use this option when you archive files that contain international (non-English) characters in file names or file comments. If this option is selected and an archive is modified in any way, UTF-8 characters are used in the names of all files in the archive.

CSS Validator Preferences

To configure the **CSS Validator** preferences, [open the Preferences dialog box](#) and go to **CSS Validator**.

You can configure the following options for the built-in **CSS Validator** of Oxygen XML Developer plugin:

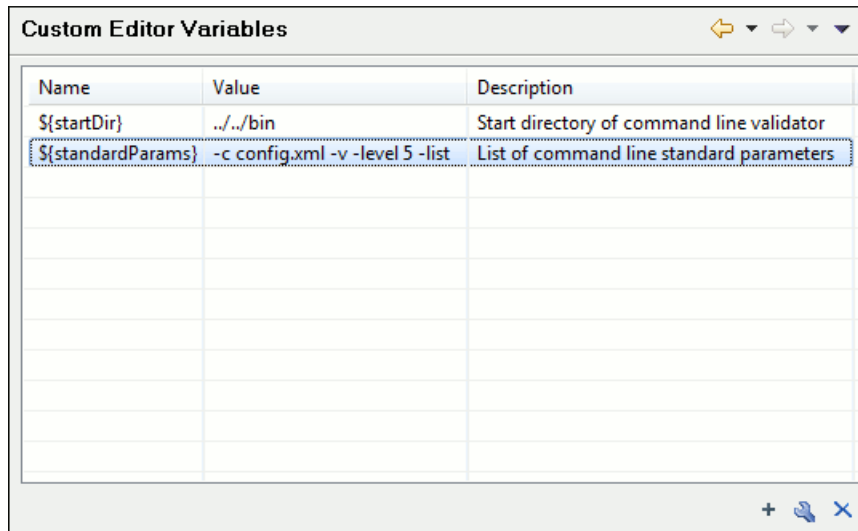
- **Profile** - Selects one of the available validation profiles: **CSS 1, CSS 2, CSS 2.1, CSS 3, CSS 3 with Oxygen extensions, SVG, SVG Basic, SVG Tiny, Mobile, TV Profile, ATSC TV Profile**. The **CSS 3 with Oxygen extensions** profile includes all the CSS 3 standard properties and the CSS extensions specific for Oxygen. That means all Oxygen specific extensions are accepted in a CSS stylesheet by [the built-in CSS validator](#) when this profile is selected.
- **Media type** - Selects one of the available mediums: **all, aural, braille, embossed, handheld, print, projection, screen, tty, tv, presentation, oxygen**.
- **Warning level** - Sets the minimum severity level for reported validation warnings. Can be one of: **All, Normal, Most Important, No Warnings**.
- **Ignore properties** - You can type comma separated patterns that match the names of CSS properties that will be ignored at validation. As wildcards you can use:
 - * to match any string.
 - ? to match any character.
- **Recognize browser CSS extensions (also applies to content completion)** - If checked, Oxygen XML Developer plugin recognizes (no validation is performed) browser-specific CSS properties. The **Content Completion Assistant** lists these properties at the end of its list, prefixed with the following particles:
 - -moz - for Mozilla.
 - -ms - for Internet Explorer or Edge.
 - -o - for Opera.
 - -webkit - for Safari/Webkit.

Custom Editor Variables Preferences

An editor variable is useful for making a transformation scenario, validation scenario, or other tool independent of its file path. 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 editor variables](#).

Custom editor variables are created and configured in the **Custom Editor Variables** preferences page. To access this page, [open the Preferences dialog box](#) and go to **Custom Editor Variables**.

This preferences page displays a table of all the custom editor variables that have been defined. The table includes three columns for the editor variable **Name**, its **Value**, and its **Description**. To create a new variable, click the  **New** button at the bottom of the table and define your custom editor variable in the subsequent dialog box. To edit an existing custom editor variable, click the  **Edit** button and configure the variable in the subsequent dialog box. You can also use the  **Delete** button to remove custom editor variables that are no longer needed.



The image shows a dialog box titled "Custom Editor Variables". It contains a table with three columns: Name, Value, and Description. The first row has the name "\${startDir}", the value ".././bin", and the description "Start directory of command line validator". The second row has the name "\${standardParams}", the value "-c config.xml -v -level 5 -list", and the description "List of command line standard parameters". The table has several empty rows below. At the bottom right of the dialog are icons for adding, deleting, and applying changes.

Name	Value	Description
\${startDir}	.././bin	Start directory of command line validator
\${standardParams}	-c config.xml -v -level 5 -list	List of command line standard parameters

Figure 9: Custom Editor Variables Table

Data Sources Preferences

To configure the **Data Sources** preferences, [open the Preferences dialog box](#) and go to **Data Sources**. This preferences page allows you to configure data sources and connections to relational and native XML databases. For a list of drivers that are available for the major database servers, see [Download Links for Database Drivers](#) on page 33.

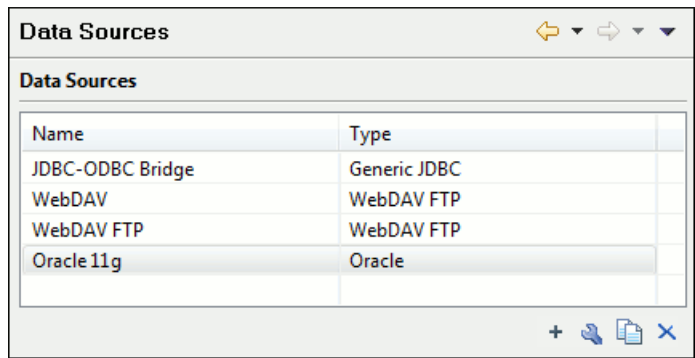
Connection Wizards Section

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.



The image shows a dialog box titled "Data Sources". It contains a table with two columns: Name and Type. The first row has the name "JDBC-ODBC Bridge" and the type "Generic JDBC". The second row has the name "WebDAV" and the type "WebDAV FTP". The third row has the name "WebDAV FTP" and the type "WebDAV FTP". The fourth row has the name "Oracle11g" and the type "Oracle". The table has several empty rows below. At the bottom right of the dialog are icons for adding, deleting, and applying changes.

Name	Type
JDBC-ODBC Bridge	Generic JDBC
WebDAV	WebDAV FTP
WebDAV FTP	WebDAV FTP
Oracle11g	Oracle

Figure 10: 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.

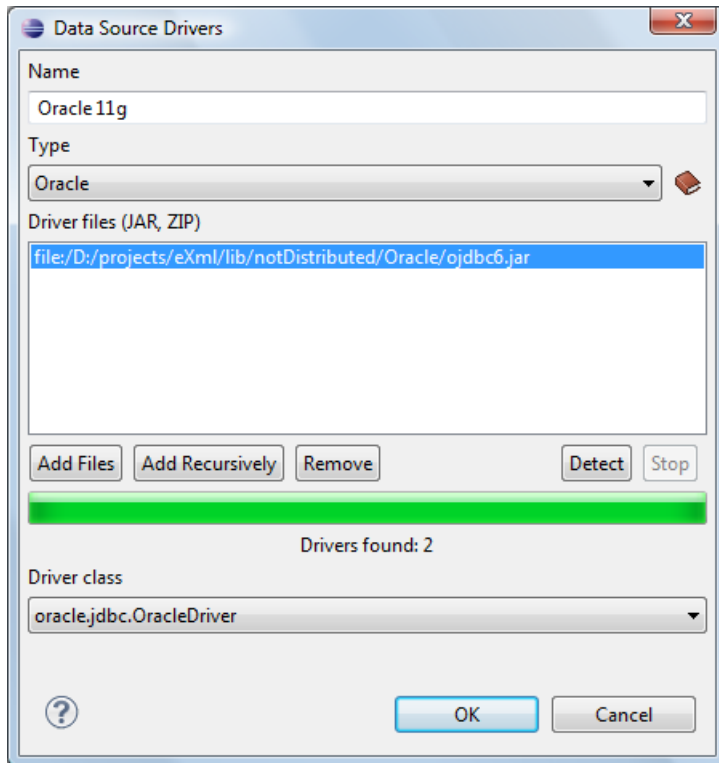


Figure 11: Data Sources Drivers Dialog Box

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 Developer 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. 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. 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.

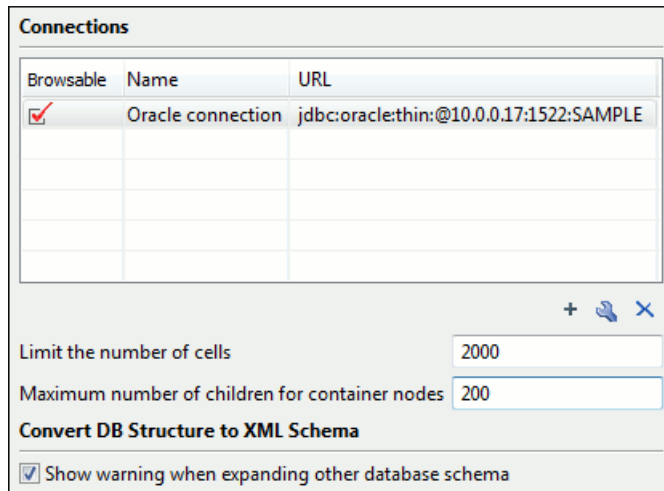


Figure 12: Connections Preferences Panel

The following buttons and options are available at the bottom of the **Connections** panel:

+ New

Opens the **Connection** dialog box that allows you to configure a new database connection.

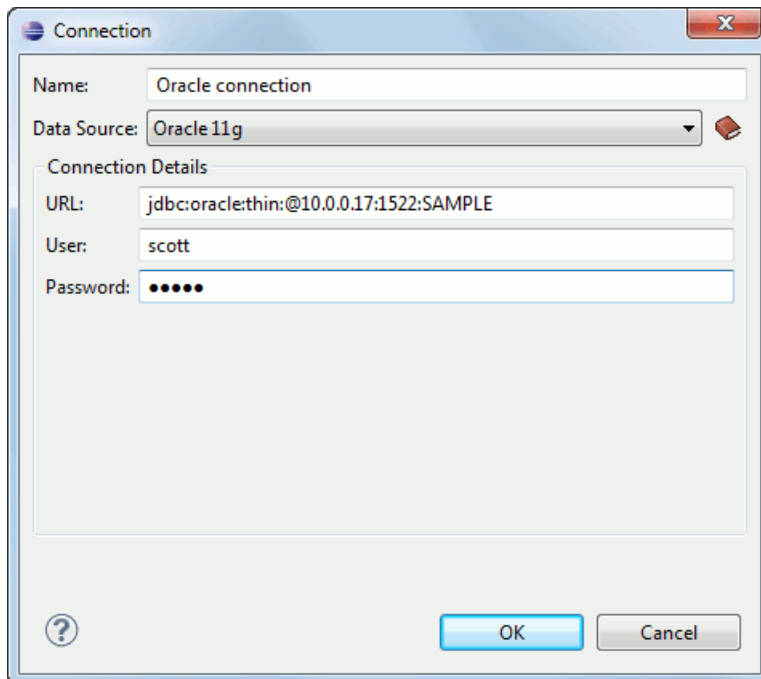


Figure 13: Connection Dialog Box

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*.

Table Filters Preferences

The **Table Filters** preferences page allows you to choose the types of tables to be shown in the **Data Source Explorer** view. To open this preferences page, *open the Preferences dialog box* and go to **Data Sources > Table Filters**.

You can choose to display the following types of tables:

- **Alias**
- **Global Temporary**
- **Local Temporary**
- **Synonym**
- **System Table**
- **Table**
- **View**

Download Links for Database Drivers

For a list of major relational databases and the drivers that are available for them, see https://www.oxygenxml.com/database_drivers.html.

In addition, the following is a list of other popular databases along with instructions for getting the drivers that are necessary to access the databases in Oxygen XML Developer plugin:

- **Berkeley DB XML database** - Copy the *jar* files from the Berkeley database install directory into the Oxygen XML Developer plugin install directory as described in [the procedure for configuring a Berkeley DB data source](#).
- **IBM DB2 Pure XML database** - Go to the [IBM website](#) and in the *DB2 Clients and Development Tools* category select the *DB2 Driver for JDBC and SQLJ* download link. Fill out the download form and download the zip file. Unzip the zip file and use the `db2jcc.jar` and `db2jcc_license_cu.jar` files in Oxygen XML Developer plugin for [configuring a DB2 data source](#).
- **eXist database** - Copy the *jar* files from the eXist database install directory to the Oxygen XML Developer plugin install directory as described in [the procedure for configuring an eXist data source](#).
- **MarkLogic database** - Download the MarkLogic driver from [MarkLogic Community site](#).
- **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 <http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html> and download the Oracle 11g JDBC driver called `ojdbc6.jar`.
- **PostgreSQL 8.3 database** - Go to <http://jdbc.postgresql.org/download.html> and download the PostgreSQL 8.3 JDBC3 driver.
- **Documentum xDB (X-Hive/DB) 10 XML database** - Copy the *jar* files from the Documentum xDB (X-Hive/DB) 10 database install directory to the Oxygen XML Developer plugin install directory as described in [the procedure](#) for configuring a Documentum xDB (X-Hive/DB) 10 data source.

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 Developer plugin installation) to be used for validating and publishing DITA content. You can select from the following:

Built-in DITA-OT 1.8

If this is set, all defined DITA transformation scenarios will run with DITA-OT 1.8.5. The built-in DITA OT 1.8.5 directory is: `[OXYGEN_INSTALL_DIR]/frameworks/dita/DITA-OT`.

Built-in DITA-OT 2.x (with support for DITA 1.3 and Lightweight DITA)

Starting with Oxygen 18.0, this is the default setting. All defined DITA transformation scenarios will run with DITA-OT 2.3.3. This also gives you access to DITA 1.3 file templates when you [create new documents from templates](#). The default DITA OT 2.3.3 directory is: `[OXYGEN_INSTALL_DIR]/frameworks/dita/DITA-OT2.x`.

Custom

Allows you to specify a custom directory for your DITA OT distribution.

Location

You can either provide a new file path for the specific DITA OT that you want to use or select a previously used one from the drop-down list. You can specify the path by using the text field, the  [Insert Editor Variables](#) button, or the  **Browse** button.

Prefer navigation title for topicref rendering

If enabled and there is a `navtitle` attribute set on a `topicref`, then the `navtitle` is used to render the title of the topic in the **DITA Maps Manager**.

Insert topic reference section

Allows you to specify that when inserting a topic reference, the values for certain attributes will always be automatically populated with a detected value (based on the specifications), even if it is the same as the default value. You can choose to always populate the values for the following attributes:

- **Format** - If enabled, the attribute will always be automatically populated with a detected value.
- **Scope** - If enabled, the `sformatcope` attribute will always be automatically populated with a detected value.

- **Type** - If enabled, the `type` attribute will always be automatically populated with a detected value.
- **Navigation title** - If enabled, the `navtitle` attribute will always be automatically populated with a detected value.

Insert link section

Allows you to specify that when a link reference is inserted (using actions in the  **Link** drop-down menu), the values for certain attributes will always be automatically populated with a detected value (based on the specifications), even if it is the same as the default value. You can choose to always populate the values for the following attributes:

- **Format** - If enabled, the `format` attribute will always be automatically populated with a detected value.
- **Scope** - If enabled, the `scope` attribute will always be automatically populated with a detected value.
- **Type** - If enabled, the `type` attribute will always be automatically populated with a detected value.

Use '.' instead of the ID of the parent topic (DITA 1.3)

When addressing a non-topic element within the topic that contains the URI reference, the URI reference can use an abbreviated fragment-identifier syntax that replaces the topic ID with ". " (`#./elementId`). For more information, see <https://www.oxygenxml.com/dita/1.3/specs/index.html#archSpec/base/uri-based-addressing.html>.

Show console output

Allows you to specify when to display the console output log. The following options are available:

- **When build fails** - displays the console output log if the build fails.
- **Always** - displays the console output log, regardless of whether or not the build fails.

Document Type Association Preferences

Oxygen XML Developer plugin uses document type associations to associate a *document type* with a set of functionality provided by a framework. To configure the **Document Type Association** options, [open the Preferences dialog box](#) and go to **Document Type Association**.

The following actions are available in 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. This Java class should implement the `ro.sync.ecss.extensions.api.DocumentTypeCustomRuleMatcher` interface.

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. This will create a snapshot of the framework in its current form. It is merely a copy of the document type and will not *evolve* along with the base document type like the **Extend** action does.

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 Developer 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 Developer 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_INSTALL_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 that matches the specific document.

All loaded document type configurations are first sorted by priority, then by document type.

Document Type Configuration Dialog Box

The **Document type** configuration dialog box allows you to create or edit a *Document Type Association* (framework). It is displayed when you use the **New**, **Edit**, **Duplicate**, or **Extend** buttons in the **Document Type Association** preferences page ([open the Preferences dialog box](#) and go to **Document Type Association**).

The configuration dialog box includes the following fields and sections:

- **Name** - The name of the *Document Type Association*.
- **Priority** - Depending on the priority level, Oxygen XML Developer plugin establishes the order in which the existing document type associations are evaluated to determine the type of a document you are opening. It can be one of the following: Lowest, Low, Normal, High, or Highest. You can set a higher priority to *Document Type Associations* you want to be evaluated first.
- **Description** - A detailed description of the framework.
- **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.

- **Initial edit mode** - Sets the default edit mode when you open a document for the first time.
- **Configuration Tabs** - The bottom section of the dialog box includes various tabs where you can configure numerous options for the *framework*.

Association Rules Tab

By combining multiple association rules you can instruct Oxygen XML Developer plugin to identify the type of a document. An Oxygen XML Developer plugin *association rule* holds information about *Namespace*, *Root local name*, *File name*, *Public ID*, *Attribute*, and *Java class*. Oxygen XML Developer plugin identifies the type of a document when the document matches at least one of the *association rules*. This tab give you access to a **Document type rule** dialog box that you can use to create *association rules* that activate on any document matching all the criteria defined in the dialog box.

To open the **Association Rules** tab of the **Document type** configuration dialog box, [open the Preferences dialog box](#), go to **Document Type Association**, use the [New, Edit, Duplicate, or Extend button](#), and click on the **Association Rules** tab.

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* from the list.

⬆ Move Up

Moves the selected *association rule* up one spot in the list.

⬇ Move Down

Moves the selected *association rule* down one spot in the list.

Schema Tab

In the **Schema** tab, you can specify a schema that Oxygen XML Developer plugin uses if an XML document does not contain a schema declaration and no default validation scenario is associated with it.

To open the **Schema** tab of the **Document type** configuration dialog box, [open the Preferences dialog box](#), go to **Document Type Association**, use the [New, Edit, Duplicate, or Extend button](#), and click on the **Schema** tab.

This tab includes the following options for defining a schema to be used if no schema is detected in the XML file:

Schema type

Use this drop-down list to select the type of schema.

Schema URI

You can specify the URI of the schema file. You can specify the path by using the text field, its history drop-down, the



[Insert Editor Variables](#) button, or the browsing tools in the **Browse** drop-down list.



Tip: 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 multiple users.

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, various resources (such as stylesheets), and framework translation files. Oxygen XML Developer plugin loads the resources looking in the folders in the order they appear in the list.

To open the **Classpath** tab of the **Document type** configuration dialog box, [open the Preferences dialog box](#), go to **Document Type Association**, use the [New, Edit, Duplicate, or Extend button](#), and click on the **Classpath** tab.

The **Classpath** tab includes the following actions:

New

Opens a dialog box that allows you to add a resource to the table in the **Classpath** tab. You can specify the path by using the text field, its history drop-down, the  **Insert Editor Variables** button, or the browsing tools in the  ▾ **Browse** drop-down list.



Tip: The path can also contain wildcards (for example, `${framework}/lib/*.jar`).

Edit

Opens a dialog box that allows you to edit a resource in the **Classpath** tab. You can specify the path by using the text field, its history drop-down, the  **Insert Editor Variables** button, or the browsing tools in the  ▾ **Browse** drop-down list.



Tip: The path can also contain wildcards (for example, `${framework}/lib/*.jar`).

Delete

Deletes the currently selected resource from the list.

Move Up

Moves the selected resource up one spot in the list.

Move Down

Moves the selected resource down one spot in the list.

Related information

[Extensions Tab](#) on page 51

[Author Tab](#) on page 38

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.

To open the **Author** tab of the **Document type** configuration dialog box, [open the Preferences dialog box](#), go to **Document Type Association**, use the [New, Edit, Duplicate, or Extend button](#), and click on the **Author** tab.

The options that you configure in the **Author** tab are grouped in subtabs.

CSS Subtab

The **CSS** subtab contains the CSS files that Oxygen XML Developer 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.

To open the **CSS** subtab, [open the Preferences dialog box](#), go to **Document Type Association**, use the [New, Edit, Duplicate, or Extend button](#), click on the **Author** tab, and then the **CSS** subtab.

The following actions are available in the **CSS** subtab:

New

Opens a dialog box that allows you to add a CSS file. You can specify the path by using the text field, its history drop-down, the  **Insert Editor Variables** button, or the browsing tools in the  ▾ **Browse** drop-down list.

Edit

Opens a dialog box that allows you to edit the current selection.

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.

If there are CSSs specified in the document then

You can choose between the following options for controlling how the CSS files that are set in this subtab will be handled if a CSS is specified in the document itself:

- **Ignore CSSs from the associated document type** - The CSS files set in this CSS subtab 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 this CSS subtab are merged with the CSS files specified in the document itself.

Actions Subtab

The **Actions** subtab contains a sortable table that includes all the framework-specific actions. Each action has a unique ID, a name, a description, and a shortcut key.

To open the **Actions** subtab, *open the Preferences dialog box*, go to **Document Type Association**, use the *New, Edit, Duplicate, or Extend button*, click on the **Author** tab, and then the **Actions** subtab.

The following actions are available in this subtab:

New

Opens the *Action dialog box* that allows you to add an action.

Duplicate

Duplicates the currently selected action.

Edit

Opens the *Action dialog box* that allows you to edit the existing action.

Delete

Deletes the currently selected action.

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**, use the *New, Edit, Duplicate, or Extend button*, click on the **Author** tab, and then the **Actions** subtab. At the bottom of this subtab, click  **New** to create a new action, or  **Edit** to modify an existing one.

Action

ID: Description:

Name:

Menu access key:

[How to translate frameworks](#)

Large icon (24x24):

Small icon (16x16):

Shortcut key:

☒ Enable platform-independent shortcut keys

Operations

1

Activation XPath:

This XPath expression applies only to elements and attributes. [More details](#)

Operation:

Arguments:

Name	Description	Type	Value
fragment	The fragment to be inserted	Fragment	
insertLocation	An XPath expression indicating the insert location ...	XPathExpres...	
insertPosition	The insert position relative to the node determine...	ConstantList	Inside as first child
schemaAware	Controlling if the insertion is schema aware or not...	ConstantList	true
goToNextEditablePo	After inserting the fragment, the first editable node...	ConstantList	true

Operation priority:

☐ Evaluate activation XPath expressions even in read-only contexts

Figure 14: 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.

 **Tip:** You can use the `${i18n('key')}` editor variable to allow for multiple translations of the name.

Menu access key

In Windows, you can access menus by holding down **Alt** and pressing the keyboard key that corresponds to the *Letter* that is underlined in the name of the menu. Then, while still holding down **Alt**, you can select submenus and menu action the same way by pressing subsequent corresponding keys. You can use this option to specify the *Letter* in the name of the action that can be used to access the action.

Description

A description of the action. This description is displayed as a tooltip when hovering over the action.

 **Tip:** You can use the `${i18n('key')}` editor variable to allow for multiple translations of the description.

How to translate frameworks link

Use this link to see more information about [Localizing Frameworks](#).

Large icon

Allows you to select an image for the icon that Oxygen XML Developer plugin uses for the toolbar action.



Tip: A good practice is to store the image files inside the framework directory and use the `${frameworks}` editor variable to make the image relative to the framework location. If the images are bundled in a *jar* archive (for instance, along with some Java operations implementation), it is convenient to reference the images by their relative path location in the *class-path*.

Small icon

Allows you to select an image for the icon that Oxygen XML Developer plugin uses for the contextual menu action.

Shortcut key

This field allows you to configure a shortcut key for the action that you are editing. The `+` character separates the keys.

Enable platform-independent shortcut keys

If this 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 section

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: [Activate Multiple Functions for Actions using XPath Expressions](#) on page 41.

The following options are available in this section:

Activation XPath

An XPath 2.0 expression that applies to elements and attributes. For more details see: [Activate Multiple Functions for Actions using XPath Expressions](#) on page 41.

Operation

Specifies the invoked operation.

Arguments

Specifies the arguments of the invoked operation. The  **Edit** at the bottom of the table 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 multiple XPath expressions are true, the operation with the highest priority is invoked.

-  **Add** - Adds an operation.
-  **Remove** - Removes an operation.
-  **Duplicate** - Duplicates an operation.

Evaluate activation XPath expressions even in read-only contexts

If this checkbox is enabled, the action can be invoked even when the cursor is placed in a read-only location.

Activate 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.

Three special XPath extension functions are provided:

- `oxy:allows-child-element()` - You can use this function to check whether or not an element is valid child element in the current context, according to the associated schema.

- `oxy:allows-global-element()` - You can use this function to check whether or not an element is a valid global element for the current framework, according to the associated schema.
- `oxy:current-selected-element()` - You can use this function to get the currently selected element.

`oxy:allows-child-element()` Function

The `oxy:allows-child-element()` function allows you to check whether or not an element that matches the arguments of the function is valid as a child of the element at the current cursor position, according to the associated schema. It 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.

oxy:allows-global-element() Function

The `oxy:allows-global-element()` function allows you to check whether or not an element that matches the arguments of the function is valid for the current framework, according to the associated schema. It has the following signature:

oxy:allows-global-element(\$elementName, (\$attributeName, \$defaultAttributeValue, \$contains?))?

The following parameters are supported:

elementName

The name of the element that you want to check if it is valid in the current framework. Its value is a string that supports the following forms:

- The element with the specified local name that belongs to the default namespace.

```
oxy:allows-global-element("para")
```

The above example verifies if the `para` element (of the default namespace) is allowed in the current framework.

- The element with the local name specified by any namespace.

```
oxy:allows-global-element("*:para")
```

The above example verifies if the `para` element (of any namespace) is allowed in the current framework.

- A prefix-qualified name of an element.

```
oxy:allows-global-element("prefix:para")
```

The prefix is resolved in the context of the framework. 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-global-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 framework.

- Any element.

```
oxy:allows-global-element("**")
```

The above function verifies if any element is allowed in the current framework.

attributeName

The attribute of an element that you want to check if it is valid in the current framework. Its value is a string that supports the following forms:

- The attribute with the specified name from no namespace.

```
oxy:allows-global-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 framework.

- The attribute with the local name specified by any namespace.

```
oxy:allows-global-element( "*", " *:localname", " topic/topic ")
```

- A qualified name of an attribute.

```
oxy:allows-global-element( "*", " prefix:localname", " topic/topic ")
```

The prefix is resolved in the context of the framework. 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.

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.

Localizing Frameworks

Oxygen XML Developer plugin supports framework localization (translating framework actions, buttons, and menu entries to various languages). This lets you develop and distribute a framework to users that speak other languages without changing the distributed framework.

To localize the content of a framework, create a `translation.xml` file that 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 Developer 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 can 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 `${i18n(translation_key)}`. Actions with a name format other than `${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 multiple 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 `language list` tag and any number of keys.

The `translation.xml` file for the DocBook framework is located here: `[OXYGEN_INSTALL_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 various keys.

Menu Subtab

In the **Menu** subtab, you can configure which actions will appear in the framework-specific menu. The subtab is divided in two sections: **Available actions** and **Current actions**.

To open the **Menu** subtab, [open the Preferences dialog box](#), go to **Document Type Association**, use the [New, Edit, Duplicate, or Extend button](#), click on the **Author** tab, and then the **Menu** subtab.

The **Available actions** section presents a table that displays the actions defined in the **Actions** subtab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the Oxygen XML Developer 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 Subtab

In the **Contextual menu** subtab you configure what framework-specific action the **Content Completion Assistant** proposes. The subtab is divided in two sections: **Available actions** and **Current actions**.

To open the **Contextual Menu** subtab, *open the [Preferences dialog box](#)*, go to **Document Type Association**, use the *[New](#), [Edit](#), [Duplicate](#), or [Extend](#) button*, click on the **Author** tab, and then the **Contextual Menu** subtab.

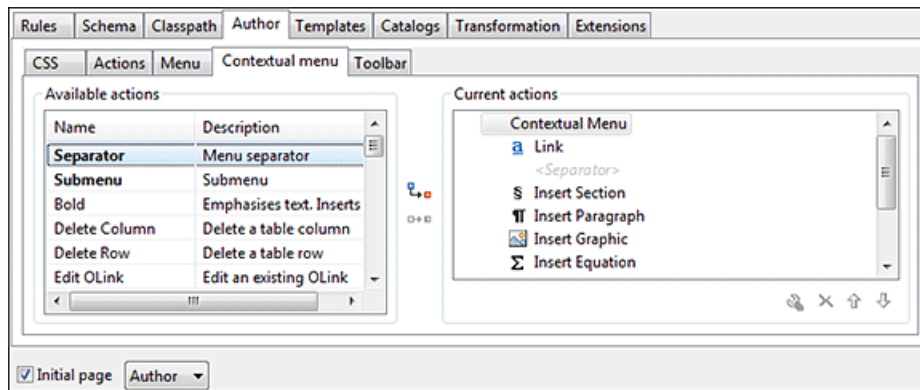


Figure 15: Contextual Menu Subtab

The **Available actions** section presents a table that displays the actions defined in the **Actions** subtab, along with their icon, ID, and name. The **Current actions** section contains the actions that are displayed in the contextual menu for documents that belong to the edited framework.

The following actions are available in this subtab:



Add as sibling

Adds the selected action or submenu from the **Available actions** section to the **Current actions** section as a sibling of the selected action.



Add as child

Adds the selected action or submenu from the **Available actions** section to the **Current actions** section as a child of the selected action.



Edit

This option is available for container (submenu) items that are listed in the **Current actions** section. It opens a configuration dialog box that allows you to edit the selected container (submenu).

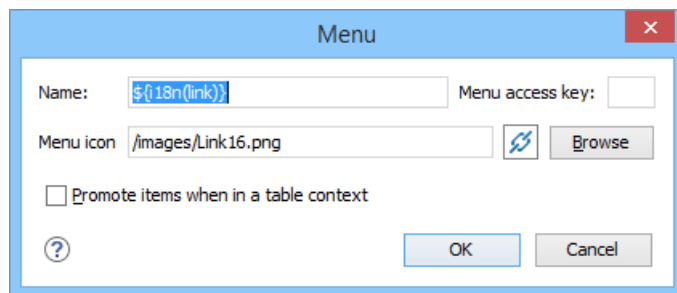


Figure 16: Menu Action Configuration Dialog Box

The following options are available in this dialog box:

Name

Specifies the name of the action. This name is displayed as a tooltip or as a menu item.



Tip: You can use the *[\\${i18n\('key'\)}](#) editor variable* to allow for multiple translations of the name.

Menu access key

In Windows, you can access menus by holding down **Alt** and pressing the keyboard key that corresponds to the *Letter* that is underlined in the name of the menu. Then, while still holding down **Alt**, you can select submenus and menu action the same way by pressing subsequent corresponding keys. You can use this option to specify the *Letter* in the name of the action that can be used to access the action.

Menu icon

Allows you to select an image for the icon that Oxygen XML Developer plugin uses for the container (submenu).

Promote items when in a table context

If this option is enabled, when invoking the contextual menu from within a table, all the actions in this container (submenu) will be promoted to the main level in the contextual menu. Actions and submenus that are not promoted are still available in the **Other actions** submenu when invoking the contextual menu within a table.

Remove

Removes the selected action or submenu from the **Current actions** section.

Move Up

Moves the selected item up in the list.

Move Down

Moves the selected item down in the list.

Toolbar Subtab

In the **Toolbar** subtab you configure what framework-specific action the Oxygen XML Developer plugin toolbar holds. The subtab is divided in two sections: **Available actions** and **Current actions**.

To open the **Toolbar** subtab, *open the Preferences dialog box*, go to **Document Type Association**, use the *New, Edit, Duplicate, or Extend button*, click on the **Author** tab, and then the **Toolbar** subtab.

The **Available actions** section presents a table that displays the actions defined in the **Actions** subtab, along with their icon, ID, and name. The **Current actions** section holds the actions that are displayed in the Oxygen XML Developer plugin toolbar when you work with a document belonging to the edited framework. To add an action in this section as a sibling of the currently selected action, use the  **Add as sibling** button. To add an action in this section as a child of the currently selected action use the  **Add as child** button.

The following actions are available in the **Current actions** section:

Edit

Edits an item.

Remove

Removes an item.

Move Up

Moves an item up.

Move Down

Moves an item down.

Content Completion Subtab

In the **Content Completion** subtab you configure what framework-specific the **Content Completion Assistant** proposes. The subtab is divided in two sections: **Available actions** and **Current actions**.

To open the **Content Completion** subtab, *open the Preferences dialog box*, go to **Document Type Association**, use the *New, Edit, Duplicate, or Extend button*, click on the **Author** tab, and then the **Content Completion** subtab.

Available and Current Actions

The **Available actions** section presents a table that displays the actions defined in the **Actions** subtab, 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.

Filter Table

The **Filter** section presents a table that allows you to add elements to be filtered from the **Content Completion Assistant** or from some specific helper views or menus. Use the  **Add** button to add more filters to the table, the  **Edit** button to modify an existing item in the table, or the  **Remove** button to remove a filtered item. The  **Add** and  **Edit** buttons open a **Remove item** dialog box.

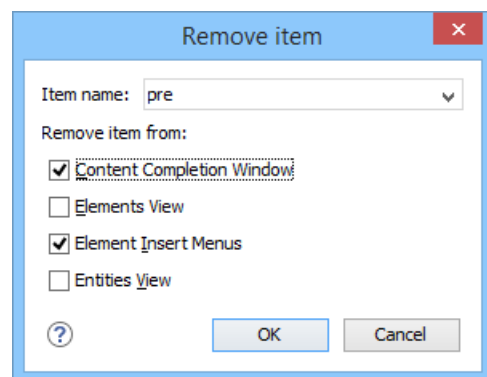


Figure 17: Remove Item Dialog Box

Use this dialog box to add or configure the elements that will be filtered:

Item name

Use this text field to enter the name of the element to be filtered. The drop-down list also includes a few special content completion actions that can be filtered (<SPLIT> and <ENTER>).

Remove item from

You can choose to filter the element from any of the following:

- **Content Completion Window** - The element will not appear in the **Content Completion Assistant**.
- **Elements View** - The element will not appear in the **Elements** view.
- **Element Insert Menus** - The element will not appear in the **Append Child**, **Insert Before**, or **Insert After** menus that are available in certain contextual menus (for example, the contextual menu of the **Outline** view).
- **Entities View** - The element will not appear in the **Entities** view.

Templates Tab

The **Templates** tab specifies a list of directories where new file templates are located. These file templates are gathered from all the document types and presented in the various folders inside the **Framework templates** folder in the [New from templates wizard](#).

To open the **Templates** tab of the **Document type** configuration dialog box, [open the Preferences dialog box](#), go to **Document Type Association**, use the [New, Edit, Duplicate, or Extend button](#), and click on the **Templates** tab.

The **Templates** tab includes the following actions:

New

Opens a dialog box that allows you to specify the path to the directory of the template. You can specify the path by using the text field, its history drop-down, the  **Insert Editor Variables** button, or the browsing tools in the  ▾ **Browse** drop-down list.



Tip: The path can also contain wildcards. For example, using `${frameworkDir}/templates/*` would add all the template folders found inside the `templates` directory.

Edit

Opens a dialog box that allows you to edit the path of the selected template. You can specify the path by using the text field, its history drop-down, the  **Insert Editor Variables** button, or the browsing tools in the  ▾ **Browse** drop-down list.



Tip: The path can also contain wildcards. For example, using `${frameworkDir}/templates/*` would add all the template folders found inside the `templates` directory.

Delete

Deletes the currently selected template from the list.

Move Up

Moves the selected template up one spot in the list.

Move Down

Moves the selected template down one spot in the list.

Catalogs Tab

The **Catalogs** tab specifies a list of *XML catalogs*, specifically for the edited framework, that are added to list of catalogs that Oxygen XML Developer plugin uses to resolve resources.

To open the **Catalogs** tab of the **Document type** configuration dialog box, *open the Preferences dialog box*, go to **Document Type Association**, use the *New, Edit, Duplicate, or Extend button*, and click on the **Catalogs** tab.

You can perform the following actions:

Add

Opens a dialog box that allows you to add a catalog to the list. You can specify the path by using the text field, its history drop-down, the  **Insert Editor Variables** button, or the browsing tools in the  ▾ **Browse** drop-down list.

Edit

Opens a dialog box that allows you to edit the path of an existing catalog.

Delete

Deletes the currently selected catalog from the list.

Move Up

Moves the selected catalog up one spot in the list.

Move Down

Moves the selected catalog down one spot in the list.

Transformation Tab

In the **Transformation** tab, you can configure the transformation scenarios associated with the particular framework you are editing. These transformation scenarios are presented in the **Configure Transformation Scenarios** dialog box when transforming a document and you can specify which scenarios will be used by default for a particular document type.

To open the **Transformation** tab of the **Document type** configuration dialog box, *open the Preferences dialog box*, go to **Document Type Association**, use the *New, Edit, Duplicate, or Extend button*, and click on the **Transformation** tab.

The **Transformation** tab offers the following options:

Default checkbox

You can set one or more of the scenarios listed in this tab to be used as the default transformation scenario when another specific scenario is not specified. The scenarios that are set as default are rendered bold in the **Configure Transformation Scenarios** dialog box.

New

Opens the **New scenario** dialog box allowing you to *create a new transformation scenario for the particular document type*.

Duplicate

Allows you to duplicate the configuration of an existing transformation scenario. It opens the **Edit scenario** dialog box where you can *configure the properties of the duplicated 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.

Validation Tab

In the **Validation** tab, you can configure the validation scenarios associated with the particular framework you are editing. These validation scenarios are presented in the **Configure Validation Scenarios** dialog box when validating a document and you can specify which scenarios will be used by default for a particular document type.

 **Note:** If a master file is associated with the current file, the validation scenarios defined in the master file are used and take precedence over the default scenarios defined for the particular framework. For more information on master files, see *Master Files Support* on page 159 or *Working with Modular XML Files in the Master Files Context* on page 218.

To open the **Validation** tab of the **Document type** configuration dialog box, *open the Preferences dialog box*, go to **Document Type Association**, use the *New, Edit, Duplicate, or Extend button*, and click on the **Validation** tab.

The **Validation** tab offers the following options:

Default checkbox

You can set one or more of the scenarios listed in this tab to be used as the default validation scenario when another specific scenario is not specified in the validation process. The scenarios that are set as default are rendered bold in the **Configure Validation Scenarios** dialog box.

New

Opens the **New scenario** dialog box allowing you to *create a new validation scenario*.

Duplicate

Allows you to duplicate the configuration of an existing validation scenario. It opens the **Edit scenario** dialog box where you can *configure the properties of the duplicated 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 validation scenarios.



Export selected scenarios

Export validation scenarios.



Move Up

Moves the selected scenario up one spot in the list.



Move Down

Moves the selected scenario down one spot in the list.

Extensions Tab

The **Extensions** tab specifies implementations of Java interfaces used to provide advanced functionality to the document type.

To open the **Extensions** tab of the **Document type** configuration dialog box, *open the Preferences dialog box*, go to **Document Type Association**, use the *New, Edit, Duplicate, or Extend button*, and click on the **Extensions** tab.

Libraries containing the implementations must be present in the *classpath* of your document type. The Javadoc available at <https://www.oxygenxml.com/InstData/Editor/SDK/javadoc/> contains details about how each API implementation functions.

Editor Preferences

Oxygen XML Developer plugin lets you configure the appearance of various components and features of the main editor. To access these options, *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

Allows you to set the background color for text editors.

Completion proposal background

Allows you to set the background color of the **Content Completion Assistant**.

Completion proposal foreground

Allows you to set the color of the text in the **Content Completion Assistant**.

Documentation window background

Allows you to set the background color of the documentation of elements suggested by the **Content Completion Assistant**.

Documentation window foreground

Allows you to set the color of the text for the documentation of elements suggested by the **Content Completion Assistant**.

Line wrap

If enabled, long lines are automatically wrapped in edited documents. The line wrap does not alter the document content since the application does not use *newline* characters to break long lines.

Enable folding when opening a new editor

If enabled (default value), the vertical stripe that holds the *folding markers* is displayed in **Text** mode.

Beep on operation finished

Oxygen XML Developer plugin emits a short beep when a validation, check well-formedness, or transformation 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.

Display quick-assist and quick-fix side hints

Displays the *Quick Assist* icon (💡) and *Quick Fix* icon (🔧) in the line number stripe on the left side of the editor.

Highlight matching tag

If you place the cursor on a start or end tag, Oxygen XML Developer plugin highlights the corresponding member of the pair.

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 the editor.

Content Completion Preferences

Oxygen XML Developer plugin provides a **Content Completion Assistant** that provides a list of available options at any point in a document and can auto-complete structures, elements, and attributes. To configure the **Content Completion** preferences, [open the Preferences dialog box](#) and go to **Editor > Content Completion**. These options control how the **Content Completion Assistant** works.

The following options are available:

Auto close the last opened tag

When enabled, Oxygen XML Developer plugin automatically closes the last open tag when you type `</`.

Automatically rename/delete/comment matching tags

If you rename, delete, or comment out a start tag, Oxygen XML Developer plugin automatically renames, deletes, or comments out the matching end tag.



Note: If you select **Toggle comment** for multiple starting tags and the matching end tags are on the same line as other start tags, the end tags are not commented.

Enable content completion

Toggles the content completion feature on or off.

Close the inserted element

When you choose an entry from the **Content Completion Assistant** list of proposals, Oxygen XML Developer plugin inserts both start and end tags. The following additional options are available in regards to closing the element:

- **If it has no matching tag** - The end tag of the inserted element is automatically added only if it is not already present in the document.
- **Add element content** - Oxygen XML Developer plugin inserts the required elements specified in the DTD, XML Schema, or RELAX NG schema that is [associated with the edited XML document](#).
 - **Add optional content** - If enabled, Oxygen XML Developer plugin inserts the optional elements specified in the DTD, XML Schema, or RELAX NG schema.
 - **Add first Choice particle** - If enabled, Oxygen XML Developer plugin inserts the first **choice** particle specified in the DTD, XML Schema, or RELAX NG schema.

Case sensitive search

When enabled, the search in the **Content Completion Assistant** is case-sensitive when you type a character ('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.

Position cursor between tags

When enabled, Oxygen XML Developer plugin automatically moves the cursor between the start and end tag after inserting the element. This only applies to:

- Elements with only optional attributes or no attributes at all.
- Elements with required attributes, but only when the [Insert the required attributes option](#) is disabled.

Show all entities

Oxygen XML Developer plugin displays a list with all the internal and external entities declared in the current document when you type the start character of an entity reference (for example, `&`).

Insert the required attributes

Oxygen XML Developer plugin inserts automatically the required attributes taken from the DTD or XML Schema.

Insert the fixed attributes

If enabled, Oxygen XML Developer plugin automatically inserts any **FIXED** attributes from the DTD or XML Schema for an element inserted with the help of the **Content Completion Assistant**.

Show recently used items

When enabled, Oxygen XML Developer plugin remembers the last inserted items from the **Content Completion Assistant** window. The number of items to be remembered is limited by the **Maximum number of recent items shown** list box. These most frequently used items are displayed on the top of the content completion window and their icons are decorated with a small red square..

Maximum number of recent items shown

Specifies the limit for the number of recently used items presented at the top of the **Content Completion Assistant** window.

Learn attributes values

When enabled, Oxygen XML Developer plugin learns the attribute values used in a document.

Learn on open document

Oxygen XML Developer plugin automatically learns the document structure when the document is opened.

Learn words (Dynamic Abbreviations, available on CTRL-SPACE (COMMAND-SPACE on OS X))

When enabled, Oxygen XML Developer plugin learns the typed words and makes them available in a content completion fashion by pressing **Ctrl + Space (Command + Space on OS X)** on your keyboard;



Note: In order to be learned, the words need to be separated by space characters.

Activation delay of the proposals window (ms)

Delay in milliseconds from last key press until the **Content Completion Assistant** is displayed.

Annotations Preferences

Certain types of schemas (XML Schema, DTDs, Relax NG) can include annotations that document the various elements and attributes that they define. Oxygen XML Developer plugin can display these annotations when offering content completion suggestions. To configure the **Annotations** preferences, [open the Preferences dialog box](#) and go to **Editor > Content Completion > Annotations**.

The following options are available:

Show annotations in Content Completion Assistant

Oxygen XML Developer plugin displays the schema annotations of an element, attribute, or attribute value currently selected in the **Content Completion Assistant** proposals list.

Show annotations in tooltip

Oxygen XML Developer plugin displays the annotation of elements and attributes as a tooltip when you hover over them with the cursor in the editing area or in the **Elements** view.

Show annotation in HTML format, if possible

This option allows you to view the annotations associated with an element or attribute in HTML format. It is available when editing XML documents that have associated an XML Schema or Relax NG schema. When this option is disabled the annotations are converted and displayed as plain text.

Prefer DTD comments that start with "doc:" as annotations

To address the lack of dedicated annotation support in DTD documents, Oxygen XML Developer plugin recommends prefixing with the **doc :** particle all comments intended to be shown to the developer who writes an XML validated against a DTD schema.

When this option is enabled, Oxygen XML Developer plugin uses the following mechanism to collect annotations:

- If at least one **doc :** comment is found in the entire DTD, only comments of this type are displayed as annotations.
- If no **doc :** comment is found in the entire DTD, all comments are considered annotations and displayed as such.

When the option is disabled, all comments, regardless of their type, are considered annotations and displayed as such.

Use all Relax NG annotations as documentation

When this option is selected, any element outside the Relax NG namespace, that is `http://relaxng.org/ns/structure/1.0`, is considered annotation and is displayed in the annotation window next to the **Content Completion Assistant** window and in the **Model** view. When this option is not selected, only elements from the Relax NG annotations namespace, that is `http://relaxng.org/ns/compatibility/annotations/1.0` are considered annotations.

Related information

[Schema Annotations in Text Mode](#) on page 171

XPath Preferences

Oxygen XML Developer plugin provides content-completion support for XPath expressions. To configure the options for the content completion in XPath expressions, [open the Preferences dialog box](#) and go to **Editor > Content Completion > XPath**.

The following options are available:

- **Enable content completion for XPath expressions** - Enables the [Content Completion Assistant in XPath expressions](#) that you enter in the `match`, `select`, and `test` XSL attributes.
 - **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 Developer plugin provides content completion assistance when you are writing XML Schema (XSD). To configure XSD preferences, [open the Preferences dialog box](#) and go to **Editor > Content Completion > XSD**. The option in this preferences page allows you to define additional elements to be suggested by the **Content Completion Assistant** in `xs:appinfo` elements (in addition to the elements defined in the XML Schema).

The following option is available:

When in "xs:appinfo" context, also include elements declared in the schema

You can choose between the following:

- **None** - The **Content Completion Assistant** offers only the XML Schema schema information.
- **ISO Schematron** - The **Content Completion Assistant** also includes ISO Schematron elements in `xs:appinfo`.
- **Schematron 1.5** - The **Content Completion Assistant** also includes Schematron 1.5 elements in `xs:appinfo`.
- **Other** - The **Content Completion Assistant** also includes elements from an XML Schema identified by a URL in `xs:appinfo` elements.

XSLT Preferences

XSLT stylesheets are often used to create output in XHTML or XSL-FO. In addition to suggesting content completion options for XSLT stylesheet elements, Oxygen XML Developer plugin can suggest elements from these vocabularies. To configure the XSLT content completion options, [open the Preferences dialog box](#) and go to **Editor > Content Completion > XSLT**.

The following options are available:

Include elements declared in the schema section

This section includes options in regards to detecting elements from the declared schema.

Automatically detect HTML or Formatting Objects

Detects if the output being generated is HTML or FO and provides content completion for those vocabularies. Oxygen XML Developer plugin analyzes the namespaces declared in the root element to find an appropriate schema.

If the detection fails, Oxygen XML Developer plugin uses one of the following options:

- **None** - The **Content Completion Assistant** suggests only XSLT elements.

- **HTML** - The **Content Completion Assistant** includes HTML elements, including HTML5 elements (such as `video`, `canvas`, etc.).
- **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 another output vocabulary, you can use this option to specify the path to the schema for that vocabulary. 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 section

This section specifies an additional schema that will be used for documenting XSL stylesheets. You can 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.

Custom Validation Engines Preferences

As the name implies, the **Custom Validation Engines** preferences page displays the list of custom validation engines than can be associated to a particular editor and used for validating documents. To access this page, [open the Preferences dialog box](#) and go to **Editor > Custom Validation Engines**.

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** or **Edit** button.

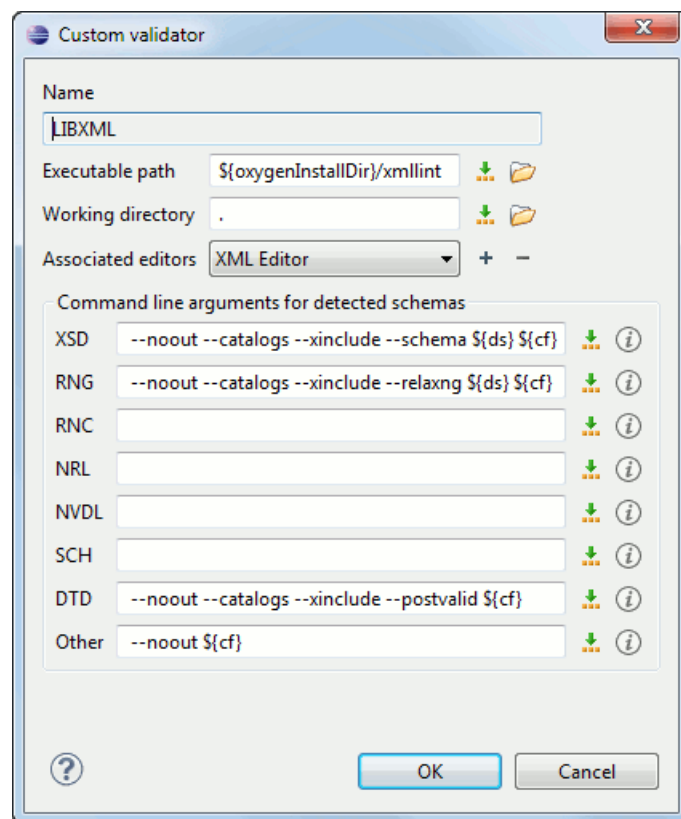


Figure 18: Custom Validator Dialog Box

The **Custom Validator** dialog box allows you to configure the following parameters:

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 specify the path by using the text field, the  [Insert Editor Variables](#) button, or the  **Browse** button.

Working directory

The working directory of the custom validation tool. You can specify the path by using the text field, the  [Insert Editor Variables](#) button, or the  **Browse** button.

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 currently edited file against various 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 (such as -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 a URL for the current validated XML document.

Related information

[Editor Variables](#) on page 99

Increasing the Stack Size for Validation Engines

To prevent the appearance of a *StackOverflowException* error, 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.

Document Checking Preferences

To configure the **Document Checking** (validation) options, [open the Preferences dialog box](#) and go to **Editor > Document Checking**. This page contains preferences for configuring how a document is checked for both well-formedness and validation errors.

The following options are available:

Maximum number of validation highlights

If a validation generates more errors than the number specified in this option, only the errors up to this number are highlighted in the editor panel and on the stripe that is displayed at the right side of the editor panel. This option applies to both [automatic validation](#) and [manual validation](#).

Clear validation markers on close

If this option is enabled, all the error markers added in the **Problems** view for that document are removed when the Oxygen XML Developer plugin is closed.

Enable automatic validation

This causes the validation to be automatically executed in the background as the document is modified in Oxygen XML Developer plugin.

Delay after the last key event (s)

The period of keyboard inactivity before starting a new validation (in seconds).

Edit Modes Preferences

Oxygen XML Developer plugin lets you configure which [edit mode](#) a file is opened in the first time it is opened. This setting only applies 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 options for editing modes, [open the Preferences dialog box](#) and go to **Editor > Edit Modes**.

Allow Document Type specific edit mode setting to override the general mode setting

If selected, the initial edit mode setting set in the [Document Type configuration dialog box](#) overrides the general edit mode setting from the table below.

Select the initial edit mode (page) for each editor

This table specifies the default editing mode that will be opened for each type of document when the **Allow Document Type specific edit mode setting to override the general mode setting** option is not selected. Use the  **Edit** button to change the initial edit mode for each type of document (editor). The initial edit mode can be one of the following:

- [Text](#)
- [Grid](#)
- **Design** (available only for the XSD editor).

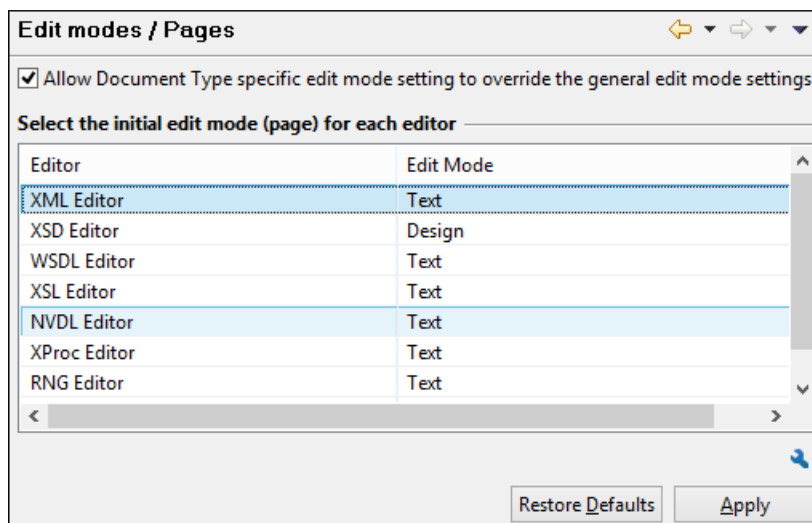


Figure 19: Edit Modes Preferences Page

Grid Preferences

Oxygen XML Developer plugin provides a [Grid view](#) of an XML document. To configure the **Grid** options, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Grid**.

The following options are available:

Compact representation

If selected, the *compact representation* of the grid is used: a child element is displayed beside the parent element. In the *non-compact representation*, a child element is nested below the parent.

Format and indent when passing from grid to text or on save

If selected, the content of the document is *formatted and indented* each time you switch from the **Grid** view to the **Text** view.

Default column width (characters)

Sets the default width (in characters) of a table column of the grid. A column may contain the following:

- 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

Allows you to set 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

Allows you to set the background color for the selected cells of the grid except the active cell.

Border color

Allows you to set the color used for the lines that separate the grid cells.

Background color

Allows you to set the background color of grid cells that are not selected.

Foreground color

Allows you to set the text color of the information displayed in the grid cells.

Row header colors**Background color**

Allows you to set the background color of row headers that are not selected.

Active cell color

Allows you to set the background color of the row header cell that is currently active.

Selection color

Allows you to set the background color of the header cells corresponding to the currently selected rows.

Column header colors

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.

Background color

Allows you to set the background color of column headers that are not selected.

Active cell color

Allows you to set the background color of the column header cell that is currently active.

Selection color

Allows you to set the background color of the header cells corresponding to the currently selected columns.

Schema Design Preferences

Oxygen XML Developer plugin provides a [graphical schema design editor](#) to make editing XML Schema easier. To configure the **Schema Design** options, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Schema Design**.

The following options are available in the **Schema Design** preferences page:

Show annotation in the diagram

When selected, Oxygen XML Developer plugin displays the content of `xs:documentation` elements in schema diagrams.

When trying to edit components from another schema

The schema diagram editor will combine schemas imported by the current schema file into a single schema diagram. You can choose what happens if you try to edit a component from an imported schema. The options are:

- **Always go to its definition** - Oxygen XML Developer plugin opens the imported schema file so that you can edit it.
- **Never go to its definition** - The imported schema file is not opened and the component cannot be edited in place.
- **Always ask** - Oxygen XML Developer plugin asks if you want to open the imported schema file.

Properties Preferences

Oxygen XML Developer plugin lets you control which properties to display for XML Schema components in the [XML Schema Design view](#). To configure the schema design properties displayed, [open the Preferences dialog box](#) and go to **Editor > Edit modes > Schema Design > Properties**.

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 Developer plugin provides a diagram view as part of the text mode editor. To configure the **Diagram** preferences, [open the Preferences dialog box](#) and go to **Editor > Edit modes / Pages > Text Diagram**.

The following options are available in this preference page:

Show Full Model XML Schema diagram

When this option is selected, the **Text** mode editor for XML Schemas includes a split screen view that shows a diagram of the schema structure. This is useful for seeing the effects of schema changes you make. For editing a the schema using a diagram instead of 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 the split screen views (**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 the split screen views (**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.

Show/Hide Annotations link

Use this link to navigate to the [Schema Design preferences page](#) where you can choose to show or hide annotations in schema diagrams.

Format Preferences

This preferences page contains various formatting options that influence editing and formatting in the **Text** mode.



Note: These settings apply to the formatting of source documents. The formatting of output documents is determined by the [transformation scenarios that create them](#).

To configure the **Format** options, [open the Preferences dialog box](#) and go to **Editor > Format**.

The following options are available:

Detect indent on open

If selected, Oxygen XML Developer plugin detects how a document is indented when it is opened. Oxygen XML Developer plugin uses a heuristic method of detection by computing a weighted average indent value from the initial document content. You can deselect 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. If this option is selected, Oxygen XML Developer plugin detects the following:

- When TAB characters are used to indent content, the size of the TAB characters is used for the indent size.
- Otherwise, the detected size of SPACE characters is used for the indent size.



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 selected.

Use zero-indent, if detected

By default, if no indent was detected in the document, the fixed-size indent is used. Select 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. Selecting this option automatically disables the [Detect indent on open](#) option.

Indent size

The meaning of this setting depends on the following:

- If the [Detect indent on open option](#) is selected and TAB characters are detected at the beginning of the line, the *indent size* is the width of a TAB character. Otherwise, the *indent size* value is ignored and Oxygen XML Developer plugin uses the number of detected SPACE characters.
- If the [Indent with tabs option](#) is selected, the *indent size* is the width of a TAB character.
- If neither of these options are selected, the *indent size* is the number of SPACE characters used for indenting the text lines.

For additional information about changing the *indent size*, see [Setting an Indent Size to Zero](#) on page 181.

For information about when this setting is used, see [Where Indent Size and Line Width Settings are Used in Oxygen XML Developer plugin](#) on page 61.

Indent on enter

If selected, when you press **Enter** to insert a line break in the **Text** editing mode, an 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.

Format and indent the document on open

If selected, an XML document is formatted and indented before opening it in Oxygen XML Developer plugin.



Note: Some specialized types of XML documents do not benefit from this feature, including Relax NG, XSD, XSL, and Ant. However, the feature is available for some non-XML types of documents, such as CSS and JSON.

Detect line width on open

If selected, Oxygen XML Developer plugin automatically detects the line width when the document is opened.

Hard line wrap (Limit to "Line width - Format and Indent")

If selected, when typing content in the **Text** editing mode and the maximum line width is reached, a line break is automatically inserted.

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 a maximum of 100 characters. This setting is also used when saving XML content edited in the **Author** editing mode.



Note: 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.

For information about when this setting is used, see [Where Indent Size and Line Width Settings are Used in Oxygen XML Editor](#).

Clear undo buffer before Format and Indent

The **Format and Indent** operation can be *undone*, but if used intensively, a considerable amount of the memory allocated for Oxygen XML Developer plugin will be used for storing the undo states. If this option is selected, Oxygen XML

Developer plugin empties the undo buffer before doing a **Format and Indent** operation. This means you will not be able to undo any changes you made before the format and indent operation. Select this option if you encounter out of memory problems (**OutOfMemoryError**) when performing the **Format and Indent** operation.

Where Indent Size and Line Width Settings are Used in Oxygen XML Developer plugin

The values set in the **Indent Size** and **Line Width - Format and Indent** options are used in various places in the application, including the following:

- When the **Format and Indent** action is used in the **Text** editing mode.
- When you press ENTER to break a line in the **Text** editing mode.
- When the **Hard line wrap (Limit to "Line width - Format and Indent")** option is selected and the maximum line width is reached while editing in the **Text** mode.

To watch our video demonstration about the formatting options offered by Oxygen XML Developer plugin, go to https://www.oxygenxml.com/demo/Autodetect_Formatting.html.

CSS Preferences

Oxygen XML Developer 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 Preferences

To configure the **JavaScript** format options, [open the Preferences dialog box](#) and go to **Editor > Format > JavaScript**.

The following options control the behavior of the **Format and Indent** action:

- **Start curly brace on new line** - Opening curly braces start on a new line.
- **Preserve empty lines** - Empty lines in the JavaScript code are preserved. This option is enabled by default.
- **Allow formatting embedded JavaScript** - Applied only to XHTML documents, this option allows Oxygen XML Developer plugin to format embedded JavaScript code, taking precedence over the [Schema aware format and indent](#) option. This option is enabled by default.

XML Preferences

To configure the **XML** Formatting options, [open the Preferences dialog box](#) and go to **Editor > Format > XML**.

The following options are available:

Format and Indent 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.

The XPath expressions can accept multiple attribute conditions and inside each condition you can use *AND/OR* boolean operators and parentheses to override the priority.

You can use one or more of 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.

Example: The following is an example of how you could use multiple boolean operators and parentheses inside an attribute condition:

```
*[@a and @b or @c and @d]
*[@a and (@b or @c) and @d]
```

The following are just examples of how simplified XPath expressions might look like:

- `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.
- `//xs:documentation[@lang="en"]`

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).

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 tab of the Element Spacing 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 fewer 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 Developer plugin formats and indents XML documents, a whitespace normalization process is applied, thus replacing whitespace sequences with single space characters. Oxygen XML Developer 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.



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.

XPath 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 an 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*.

XQuery 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.

Mark Occurrences Preferences

This preferences page specifies which types of files will have the [Highlight IDs Occurrences](#) feature activated. To configure these options, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**:

The following options are available in this preferences page:

Highlight component occurrences in the current file for:

- **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** - Activates the [Highlight Component Occurrences](#) feature in XSD files.
- **WSDL files** - Activates the [Highlight Component Occurrences](#) feature in WSDL files.
- **RNG files** - Activates the highlight component occurrences feature in RNG files.
- **Schematron files** - Activates the [Highlight Component Occurrences](#) feature in Schematron files.

Open/Save Preferences

Oxygen XML Developer plugin lets you control how files are opened and saved. To configure the options for opening and saving documents, [open the Preferences dialog box](#) and go to **Editor > Open/Save**.

The following options are available:

Open section

Format document when longest line exceeds

Oxygen XML Developer plugin will create line breaks if the characters in a line exceed the specified value. You can choose one of the following:

- **Always format**
- **Never format**
- **Always ask**

Save section

Save all files before transformation or validation

Saves all opened files before validating or transforming an XML document. This ensures that any dependencies are resolved when modifying the XML document and its XML Schema.

Check errors on save

If enabled, Oxygen XML Developer plugin runs a validation that checks your document for errors before saving it.

Performance section

Clear undo buffer on save

If selected, Oxygen XML Developer plugin clears its undo buffer when you save a document. Thus, 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.

Spell Check Preferences

Oxygen XML Developer plugin provides support for spell checking in the [Text](#) editing mode. To configure the **Spell Check** options, [open the Preferences dialog box](#) and go to **Editor > Spell Check**.

The following options are available:

Automatic spell check

This option is disabled by default. When enabled, Oxygen XML Developer 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 (such as CSS and DTD), in which automatic spell checking is not usually helpful, are excluded by default.

Language options section

This section includes the following language options:

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. 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

This section includes the following other options:

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, such as 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 recognized as 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 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** and change the color in the **Spell Check Annotation** option.

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 (`.dic` files) and term lists (`.tdi` files) that Oxygen XML Developer plugin uses and to choose where to save new learned words.

The following options are valid when Oxygen XML Developer plugin uses the Hunspell spell checking engine:

Dictionaries and term lists default folder

Displays the default location where the dictionaries and term lists that Oxygen XML Developer plugin uses are stored.

Include dictionaries and term list from

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.



Important: Consider the following notes in regards to this option:

- 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 (for example, `en_UK.dic` from the default location is merged with `en_US.dic` from a custom location).
- If you have a generic dictionary file (one that just has a two letter language code for its file name, such as `en.dic`) saved in either the default or custom location, the other more specific dictionaries (for example, `en_UK.dic` and `en_US.dic`) will not be merged and the existing generic dictionary will simply be used instead.
- 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.

How to add more dictionaries and term lists link

Use this link to open a topic in the Oxygen XML Developer plugin User Guide that explains how to [add more dictionaries and term lists](#).

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.

Related information

[Adding Spell Check Dictionaries](#) on page 420

[Adding Spell Check Term Lists](#) on page 422

Syntax Highlight Preferences

Oxygen XML Developer plugin supports syntax highlighting in the **Text** mode editors for numerous types of documents, including XML, XHTML, JavaScript, XQuery, XPath, PHP, CSS, LESS, Markdown, Text, DTD, RNC, Java, JSON, and more.

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 type of document. 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 that 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 also used in XSD, XSL, RNG documents and the **Preview** area has a separate tab for each of them when **XML** is selected in the top pane.

The **Enable nested syntax highlight** option controls whether or not content types that are nested in the same file (such as PHP, JS, or CSS scripts inside an HTML file) are highlighted according to the color schemes defined for each content type.

Elements/Attributes by Prefix Preferences

Oxygen XML Developer plugin allows you to specify syntax highlighting 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 only want the prefix, and not the whole element or attribute name, to be styled with a particular color, enable the **Draw only the prefix with a separate color** option.

Templates Preferences

This page simply allows you to navigate to the preference pages for code templates or document templates.

Code Templates Preferences

[Code templates](#) are code fragments that can be inserted at the current editing position. Oxygen XML Developer plugin includes a set of built-in templates for CSS, 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:



Note: The title of the dialog box will be determined by the type of parameter and as follows:

- For `url` and `url_relative` parameters, the title will be the name of the parameter and the value of the 'message'.
- For the other parameters listed below, the title will be the name of that respective parameter.
- If no parameter is used, the title will be "Input".

Parameter	
url	Format: <code>\${ask('message', url, 'default_value')}</code>
	Description: Input is considered a URL. Oxygen XML Developer plugin checks that the provided URL is valid.
	Example: • <code>\${ask('Input URL', url)}</code> - The displayed dialog box has the name <code>Input URL</code>. The expected input type is URL. • <code>\${ask('Input URL', url, 'http://www.example.com')}</code> - The displayed dialog box has the name <code>Input URL</code>. The expected input type is URL. The input field displays the default value <code>http://www.example.com</code>.
password	Format: <code>\${ask('message', password, 'default')}</code>
	Description: The input is hidden with bullet characters.
	Example: • <code>\${ask('Input password', password)}</code> - The displayed dialog box has the name <code>'Input password'</code> and the input is hidden with bullet symbols. • <code>\${ask('Input password', password, 'abcd')}</code> - The displayed dialog box has the name <code>'Input password'</code> and the input hidden with bullet symbols. The input field already contains the default <code>abcd</code> value.
generic	Format: <code>\${ask('message', generic, 'default')}</code>
	Description: The input is considered to be generic text that requires no special handling.
	Example: • <code>\${ask('Hello world!')}</code> - The dialog box has a <code>Hello world!</code> message displayed. • <code>\${ask('Hello world!', generic, 'Hello again!')}</code> - The dialog box has a <code>Hello world!</code> message displayed and the value displayed in the input box is <code>'Hello again!'</code>.
relative_url	Format: <code>\${ask('message', relative_url, 'default')}</code>
	Description: Input is considered a URL. Oxygen XML Developer plugin tries to make the URL relative to that of the document you are editing.  Note: If the <code>\$ask</code> editor variable is expanded in content that is not yet saved (such as an <i>untitled</i> file, whose path cannot be determined), then Oxygen XML Developer plugin will transform it into an absolute URL.
	Example: • <code>\${ask('File location', relative_url, 'C:/example.txt')}</code> - The dialog box has the name <code>'File location'</code>. The URL inserted in the input box is made relative to the current edited document location.

Parameter	
combobox	Format: <code>\${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</code>
	Description: Displays a dialog box that offers a drop-down menu. The drop-down menu is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated value (<code>real_value</code>).  Note: The 'default' parameter specifies the default selected value and can match either a key or a value.
	Example: <code>\${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}</code> - 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 <code>osx</code> key. However, the same result could be obtained if the default value is indicated by <code>Mac OS X</code>, as in the following example: <code>\${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'Mac OS X')}</code> <code>\${ask('Mobile OS', combobox, ('win':'Windows Mobile';'ios':'iOS';'and':'Android'), 'Android')}</code>
editable_combobox	Format: <code>\${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</code>
	Description: Displays a dialog box that offers a drop-down menu with editable elements. The drop-down menu is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated real value (<code>real_value</code>) 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: <code>\${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}</code> - 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.

Parameter	
radio	Format: <code>\${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</code> 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: <code>\${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}</code> - 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.
 - `${timeStamp}` - Time stamp, that is the current time in Unix format. For example, it can be used to save transformation results in multiple output files on each transformation.
 - `${uuid}` - Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java [UUID](#) class.
 - `${id}` - Application-level unique identifier. It is a short sequence of 10-12 letters and digits that is not guaranteed to be universally unique.
 - `${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_INSTALL_DIR]/frameworks` directory.
 - `${pd}` - The file path for the parent folder of the current project selected in the **Navigator** view.
 - `${oxygenInstallDir}` - Oxygen XML Developer plugin installation folder as file path.
 - `${homeDir}` - The path (as file path) of the user home folder.
 - `${pn}` - Current project name.
 - `${env(VAR_NAME)}` - Value of the VAR_NAME environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the `${system(var.name)}` editor variable.
 - `${system(var.name)}` - Value of the var.name Java System Property. The Java system properties can be specified in the command line arguments of the Java runtime as `-Dvar.name=value`. If you are looking for operating system environment variables, use the `${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>.

Related information

[Code Templates](#) on page 176

Document Templates Preferences

Oxygen XML Developer plugin provides a selection of document templates that make it easier to create new documents in a variety of formats. The list of available templates is presented when you create a new document. You can also [create your own templates](#) and share them with others. You can store your custom file templates in the existing `templates` folder in the Oxygen XML Developer plugin installation directory or store them in a custom directory. If you store them in a custom direction, you need to add that directory to the list of template directories that Oxygen XML Developer plugin uses.

To add a template directory, follow these steps:

1. [open the Preferences dialog box](#) and go to **Editor > Templates > Document Templates**.
2. Use the  **New** button to select a location of the new document template folder.

This will add the folder to the list in this preferences page and it will now appear in the [New from templates wizard](#).



Note: For DITA templates, they will also appear in the dialog box for creating new DITA topics, but if you create a [corresponding properties file](#), you need to set the `type` property to `dita`.

You can also use the **Edit** or **Delete** buttons to manage folders in the list, and you can alter the order in which Oxygen XML Developer plugin looks in these directories by using the **Up** and **Down** buttons.

Fonts Preferences

Oxygen XML Developer plugin allows you to choose the fonts to be used in the **Text**, **Design**, and **Grid** editor modes. To configure the font options, [open the Preferences dialog box](#) and go to **Fonts**.

The following options are available:

Text mode default font

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.

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.

(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:

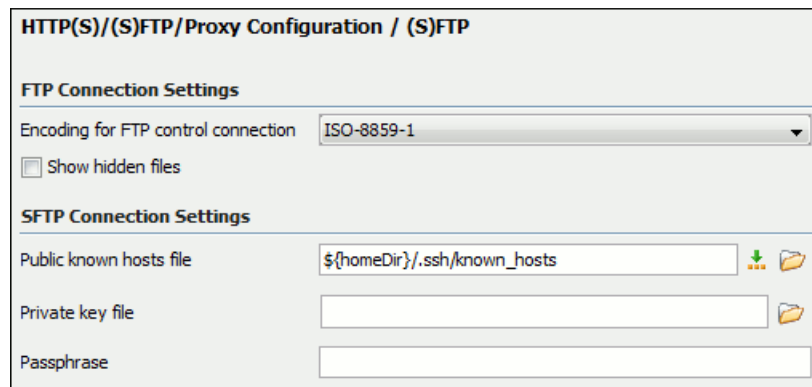


Figure 20: (S)FTP Configuration Preferences Panel

- **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 Developer plugin will use it for communication. Otherwise, it will use ISO-8859-1.
- **Public known hosts file** - File containing the list of all SSH server host keys that you have determined are accurate. The default value is `${homeDir}/.ssh/known_hosts`.
- **Private key file** - The path to the file containing the private key used for the private key method of authentication of the secure FTP (SFTP) protocol.
- **Passphrase** - The passphrase used for the private key method of authentication of the secure FTP (SFTP) protocol.
- **Show SFTP certificate warning dialog** - If checked, a warning dialog box will be displayed each time when the authenticity of the host cannot be established.

HTTP(S)/WebDAV Preferences

To set the **HTTP(S)/WebDAV** preferences, [open the Preferences dialog box](#) and go to **Network Connection Settings > HTTP(S)/WebDAV**. The following options are available:

Enable the HTTP(S)/WebDAV Protocols

Activates the HTTP(S)/WebDAV protocols bundled with Oxygen XML Developer plugin. Any adjustment to this option requires a restart of the application.

Internal Apache HttpClient Version

Oxygen XML Developer plugin uses the Apache HttpClient to establish connections to HTTP servers. For Oxygen XML Developer plugin to benefit from particular sets of features provided by different versions, you may choose between v3 and v4.



Note: For a full list of features, go to <http://hc.apache.org/httpclient-3.x/> and <http://hc.apache.org/httpcomponents-client-ga/>.

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 from that server.

Automatically accept a security certificate, even if invalid

When enabled, the HTTPS connections that Oxygen XML Developer plugin attempts to establish with 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.

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.

Trusted Hosts Preferences

This preferences page contains a list of domains that have been identified as trusted. You can add or remove domains from the list and Oxygen XML Developer plugin will allow connections to the listed 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 to the list of trusted hosts.
 **Tip:** You can specify a specific port at the end of the URL (for instance, `www.example.com:8080`). Otherwise, if no port is specified, connections will be allowed on all ports for the particular host.
- **Delete** - Allows you to remove an entry from the list of trusted hosts.

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 21: 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

The **View** preferences page allows you to configure some options in regards to certain views. To edit these options, [open the Preferences dialog box](#) and go to **View**.

The following options are available:

Console section

Enable <oxyen/> consoles

If enabled, various messages will be contributed to the [Console view](#) when certain events are triggered (such as *schema detection*, *validation*, or *transformation* events).

Fixed width console

If enabled, a line in the [Console view](#) will be hard wrapped after the specified maximum numbers of characters allowed on a line is reached.

Limit console output

If enabled, the content of the [Console view](#) will be limited to a configurable number of characters.

Console buffer

If the **Limit console output** option is enabled, this 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.

Import Preferences

To configure importing options, [open the Preferences dialog box](#) and go to **XML > Import**. This page allows you to configure how empty values and null values are handled when they are encountered in imported database tables or Excel sheets. Also you can configure the format of date / time values recognized in the imported database tables or Excel sheets.

The following options are available:

Create empty elements for empty values

If checked, an empty value from a database column or from a text file is imported as an empty element.

Create empty elements for null values

If checked, null values from a database column are imported as empty elements.

Escape XML content

Enabled by default, this option instructs Oxygen XML Developer plugin to escape the imported content to an XML-safe form.

Add annotations for generated XML Schema

If checked, the generated XML Schema contains an annotation for each of the imported table columns. The documentation inside the annotation tag contains the remarks of the database columns (if available) and also information about the conversion between the column type and the generated XML Schema type.

Date / Time Format section

Specifies the format used for importing date and time values from Excel spreadsheets or database tables, and in the generated XML schemas. You can choose from the following format types:

- **Unformatted** - 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** - 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 selected, 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`.

Table 1: Pattern Letters

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am / pm marker	Text	PM
H	Hour in day (0-23)	Number	0

Letter	Date or Time Component	Presentation	Examples
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*
 - *TwoDigitHours* (must be between 00 and 23)

Sample XML Files Generator Preferences

The [Generate Sample XML Files tool](#) (available on the **XML Tools** menu) allows you to generate XML instance documents based on a W3C XML Schema. There are various options that can be configured within the tool and these options are also available in the **Sample XML Files Generator** preferences page. This allows you to set default values for these options. To configure the options for generating the XML files, [open the Preferences dialog box](#) and go to **XML > Sample XML Files Generator**.

The following options are available:

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. The following 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 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.

Preferred number of repetitions

Allows you to set the preferred number of repeating elements related to **minOccurs** and **maxOccurs** facets defined in the 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.

Type alternative strategy

Used for the **xs:alternative** element from XML Schema 1.1. The possible strategies are:

- **First** - The first valid alternative type is always used.
- **Random** - A random alternative type is used.

Choice strategy

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

If enabled, generates 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.

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.

Related information

[Generating Sample XML Files](#) on page 315

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

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). You can choose between the following:

- **system** - If selected, the system identifier in the document is used.
- **public** - If selected, the URI supplied in the matching public catalog entry is used. Generally, the purpose of catalogs is to override the system identifiers in XML documents, so **public** should usually be used for your catalogs.



Note: If the catalog contains 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 this setting would be irrelevant.

Verbosity

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. This option selects the detail level of such logging messages of the XML catalog resolver that will be displayed in the **Catalogs** table at the bottom of the window. You can choose between the following:

- **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.

Resolve schema locations also through system mappings

If enabled, Oxygen XML Developer plugin analyzes both *uri* and *system* mappings 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.

Process "schemaLocation" namespaces through URI mappings for XML Schema

If selected, the target namespace of the imported XML Schema 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 disabled, the system IDs are used to resolve the schema location.

Use default catalog

If this option is enabled and Oxygen XML Developer plugin cannot resolve the catalog mapping with any other means, the default global catalog (listed below this checkbox) is used. For more information, see [How Oxygen XML Developer plugin Determines which Catalog to Use](#) on page 222.

Catalogs table

You can use this table to add or manage global user-defined catalogs. The following actions are available at the bottom of the table:



Add

Opens a dialog box that allows you to add a catalog to the list. You can specify the path by using the text field, its history drop-down, the [Insert Editor Variables](#) button, or the browsing tools in the ▾ **Browse** drop-down list.



Edit

Opens a dialog box that allows you to edit an existing catalog. You can specify the path by using the text field, its history drop-down, the [Insert Editor Variables](#) button, or the browsing tools in the ▾ **Browse** drop-down list.



Delete

Deletes the currently selected catalog from the list.



Up

Moves the selection to the previous resource.



Down

Moves the selection to the following resource.



Note: When you add, delete, or edit a catalog in this table, you need to reopen the currently edited files that use the modified catalog or run [a manual Validate action](#) so that the XML catalog changes take full effect.

You can also add or configure catalogs at framework level from the [Catalogs tab](#) in the [Document Type configuration dialog box](#).

Related information

[Controlling the Catalog Resolver](#)

[Working with XML Catalogs](#) on page 221

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 that reference the same schemas (grammars) such as DTD, XML Schema, or RelaxNG.

Ignore the DTD for validation if a schema is specified

Forces validation against a referenced schema (W3C XML Schema, Relax NG schema) even if the document includes also a DTD DOCTYPE declaration. This option is useful when the DTD declaration is used only to declare DTD entities and the schema reference is used for validation against a W3C XML Schema or a Relax NG schema.



Note: Schematron schemas are treated as additional schemas. The validation of a document associated with a DTD and referencing a Schematron schema is executed against both the DTD and the Schematron schema, regardless of the value of the **Ignore the DTD for validation if a schema is specified** option.

Enable XInclude processing

Enables XInclude processing. If checked, the XInclude support in Oxygen XML Developer plugin is turned on for validation and transformation of XML documents.

Base URI fix-up

According to the specification for XInclude, processors must add an `xml:base` attribute to elements included from locations with a different base URI. Without these attributes, the resulting info:et information would be incorrect.

Unfortunately, these attributes make XInclude processing to not be 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 not desired by your application, you can disable this option.

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.

Relax NG Preferences

To configure options in regards to Relax NG, [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 if 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 options in regards to Schematron, [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.

Message language

This option allows you to specify the language to be used in Schematron validation messages. You can choose between the following:

- **Use the language defined in the application** - The language that is specified in [the application](#) will be used and only the validation messages that match that language will be presented. You can use the **Change application language** link to navigate to the preferences page where you can specify the language to be used in the application.
- **Use the "xml:lang" attribute set on the Schematron root** - The language specified in the `xml:lang` attribute from the Schematron root will be used and only the validation message that match that language will be presented.
- **Ignore the language and show all message** - All messages are displayed in whatever language they are defined within the Schematron schema.
- **Custom** - Use this option to specify a custom language to be used and only the messages that match the specified language will be presented.



Note: In all cases, if the selected language is not available for a validation error or warning, all messages will be displayed in whatever language they are defined with in the Schematron schema.

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 Schema Preferences

To configure options in regards to XML Schema, [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 to be used as the default. You can choose 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 the [New document wizard](#).

Default XML Schema validation engine

Allows you to select the default validation engine to be used for XML Schema. You can choose **Xerces** or **Saxon EE**.

Xerces validation features section

Enable full schema constraint checking

Sets the *schema-full-checking* feature to `true`. This enables a validation of the parsed XML document against a schema (W3C XML Schema or DTD) while the document is parsed.

Enable honour all schema location feature

Sets the *honour-all-schema-location* feature to `true`. All the files that declare W3C XML Schema components from the same namespace are used to compose the validation model. In case this option is disabled, only the first W3C XML Schema file that is encountered in the XML Schema import tree is taken into account.

Enable full XPath 2.0 in assertions and alternative types

When enabled (default value), you can use the full XPath support in assertions and alternative types. Otherwise, only the XPath support offered by the Xerces engine is available.

Assertions can see comments and processing instructions

Controls whether or not comments and processing instructions are visible to the XPath expression used for defining an assertion in XSD 1.1.

Saxon EE validation features section

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 or not comments and processing instructions are visible to the XPath expression used to define an assertion. By default, they are not made visible (unlike Saxon 9.3).

XML Refactoring Preferences

To specify a folder for loading the custom XML refactoring operations, [open the Preferences dialog box](#) and go to **XML > XML Refactoring**. The following option is available in this preferences page:

Load additional refactoring operations from

Use this text box to specify a folder for loading custom XML refactoring operations. You can specify the path by using the text field, the  [Insert Editor Variables](#) button, or the  **Browse** button.

XML Signing Certificates Preferences

Oxygen XML Developer plugin provides two types of *keystores* for certificates that are used for digital signatures of XML documents: Java KeyStore (**JKS**) and Public-Key Cryptography Standards version 12 (**PKCS-12**). A *keystore* file is protected by a password. To configure a certificate *keystore*, [open the Preferences dialog box](#) and go to **XML > XML Signing Certificates**. You can customize the following parameters of a *keystore*:

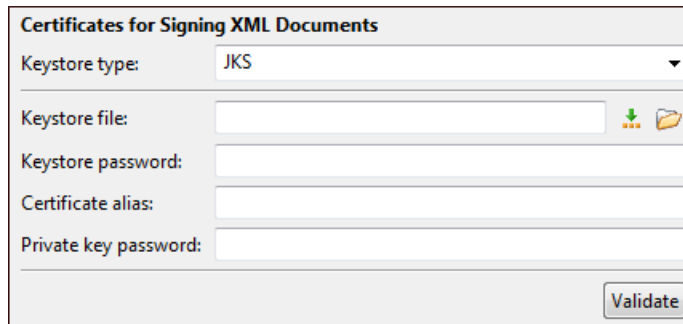


Figure 22: Certificates Preferences Panel

- **Keystore type** - The type of *keystore* that Oxygen XML Developer plugin uses (**JKS** or **PKCS-12**).
- **Keystore file** - The location of the imported file.
- **Keystore password** - The password that is used for protecting the privacy of the stored keys.
- **Certificate alias** - The alias used for storing the key entry (the certificate or the private key) inside the *keystore*.
- **Private key password** - The private key password of the certificate (required only for JKS *kestores*).
- **Validate** - Press this button to verify the configured *keystore* and the validity of the certificate.

XProc Preferences

Oxygen XML Developer plugin includes a built-in XProc engine called *Calabash*. You can add or configure external XProc engines by using the **XProc** 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.

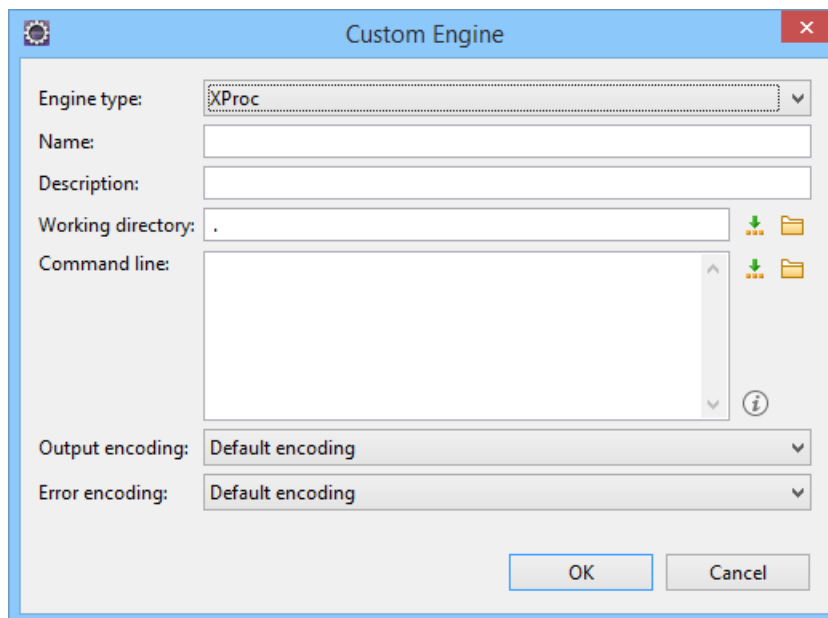


Figure 23: Creating an XProc external engine

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 specify the path by using the text field, the [Insert Editor Variables](#) button, or the **Browse** button.

- **Command line** - The command line that will run the XProc engine as an external process. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.
- **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_INSTALL_DIR]\lib\xproc\calabash\lib`. If they do not exist, you have to create them.

XSLT-FO-XQuery Preferences

To configure options in regards to XSLT, XQuery, and FO processors, *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 (including the temporary files) and the result is incorrect or the transformation fails because of this.

Custom Engines Preferences

Oxygen XML Developer plugin allows you to configure custom processors to be used for running XSLT and XQuery transformations.

 **Note:**

To configure the **Custom Engines** preferences, *open the **Preferences dialog box*** and go to **XML > XSLT-FO-XQuery > Custom Engines**.

The table in this preferences page displays the custom engines that have been defined. Use the  **New** or  **Edit** button at the bottom of the table to open a dialog box that allows you to add or configure a custom engine.

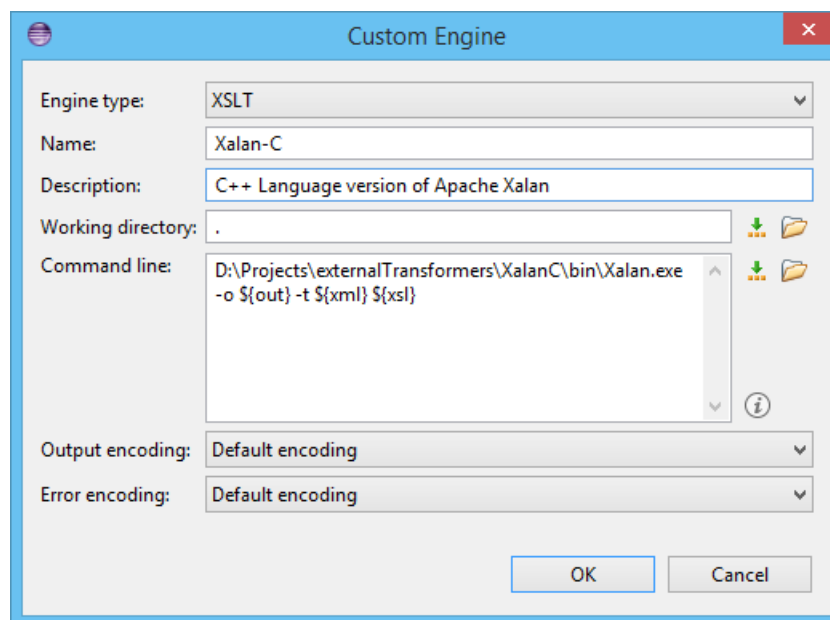


Figure 24: Parameters of a Custom Engine

The following parameters can be configured for a custom engine:

Engine type

Specifies the transformer type. You can choose between XSLT 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 executable program for the transformer. The following editor variables are available for making the path to the working directory independent of the location of the input files:

- **\${homeDir}** - The user home directory in the operating system.
- **\${cfd}** - The path to the directory of the current file.
- **\${pd}** - The path to the directory of the current project.
- **\${oxygenInstallDir}** - The Oxygen XML Developer plugin install directory.

Command line

The command line that must be executed by Oxygen XML Developer plugin to perform a transformation with the engine. The following editor variables are available for making the parameters in the command line (the transformer executable, the input files) independent of the location of the input files:

- **\${xml}** - The XML input document as a file path.
- **\${xmlu}** - The XML input document as a URL.
- **\${xsl}** - The XSL / XQuery input document as a file path.
- **\${xslu}** - The XSL / XQuery input document as a URL.
- **\${out}** - The output document as a file path.
- **\${outu}** - The output document as a URL.
- **\${ps}** - The platform separator that 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.

Debugger Preferences

To configure the **Debugger** preferences, [open the Preferences dialog box](#) and go to **XML > XSLT-FO-XQuery > Debugger**.

The following options 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

Enable this option to receive notifications when an infinite loop occurs during transformation.

Enable Saxon optimizations

This option is disabled by default and this means that the optimization level for the debugging process is set to zero. If it is enabled, the debugging process will use the optimization level from one of the following:

- The value specified in the [Optimization level option in an associated XSLT transformation](#) (or the same [option in an associated XQuery transformation](#)), if a transformation scenario is used in the debugging process.
- Otherwise, the value specified in the [Optimization level option in the XSLT Saxon-HE/PE/EE preferences page](#) (or the same [option in the XQuery Saxon HE/PE/EE preferences page](#)).

Maximum depth in templates stack

Allows you to set how many `xsl:template` instructions can appear on the current stack. This setting is used by the infinite loop detection.

Debugger layout

If you select the **Horizontal** layout, the stack of XML editors is presented on the left half of the editing area while the stack of XSL editors is on the right half. If you select the **Vertical** layout, the stack of XML editors is presented on the upper half of the editing area while the stack of XSL editors is on the lower half.

XWatch evaluation timeout (seconds)

Allows you to specify the maximum time that Oxygen XML Developer plugin allocates to the evaluation of XPath expressions while debugging.

FO Processors Preferences

Oxygen XML Developer 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 Developer 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 Developer 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.

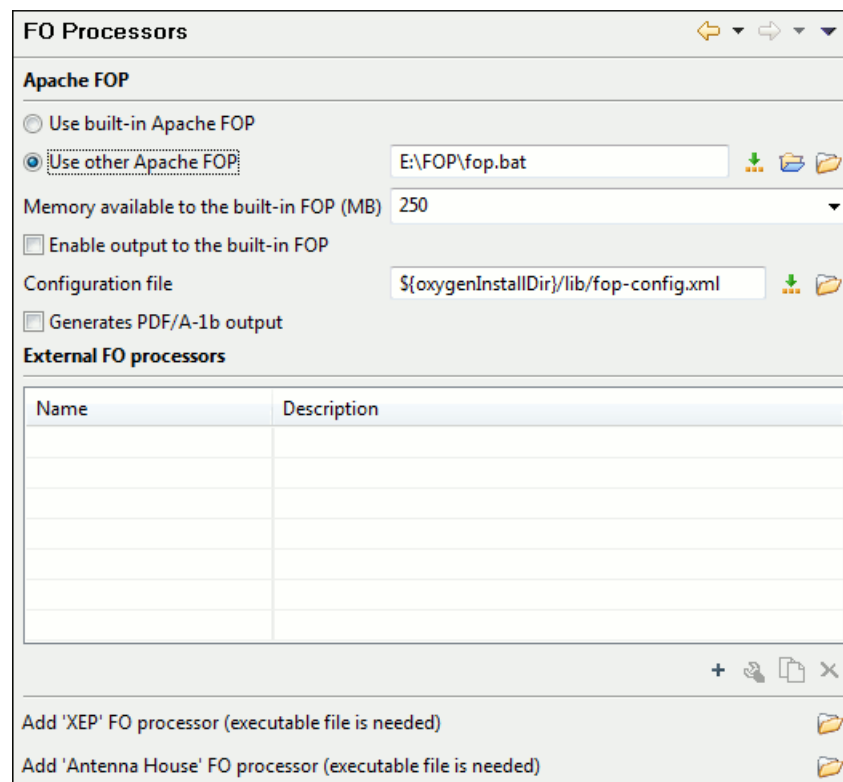


Figure 25: FO Processors Preferences Page

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 Developer plugin to use the built-in Apache FO processor.

Use other Apache FOP

Instructs Oxygen XML Developer plugin to use another Apache FO processor that is installed on your computer. You can specify the path by using the text field, the [Insert Editor Variables](#) button, or the **Browse** button.

Enable the output of the built-in FOP

All Apache FOP output is displayed in a results pane at the bottom of the Oxygen XML Developer plugin window, including warning messages about FO instructions not supported by Apache FOP.

Memory available to the built-in FOP

If your Apache FOP transformations fail with an Out of Memory error (**OutOfMemoryError**), 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). You can specify the path by using the text field, the  **Insert Editor** **Variables** button, or the  **Browse** button.

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(AHF63_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.

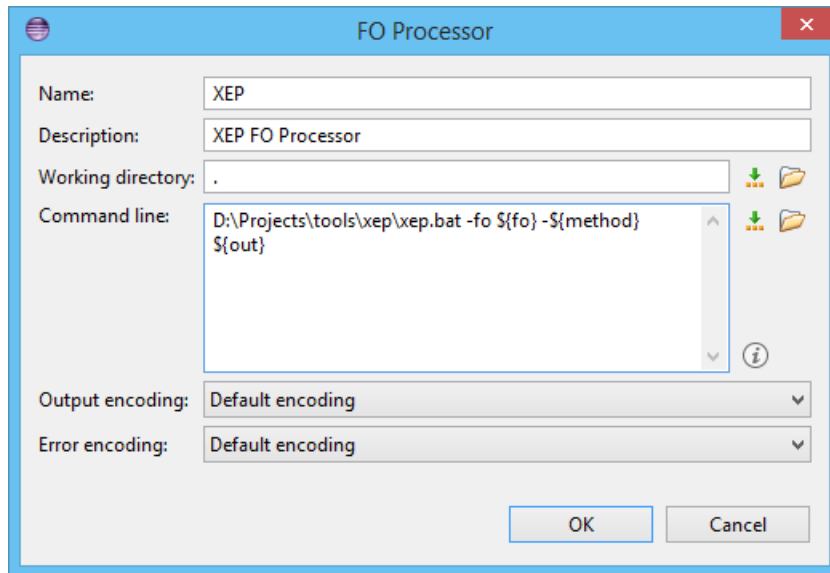


Figure 26: External FO Processor Configuration Dialog Box

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 specify the path by using the text field, the **Insert Editor Variables** button, or the **Browse** button. 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 Developer plugin installation directory.

Command line

The command line that starts the FO processor, specific to each processor. You can specify the path by using the text field, the **Insert Editor Variables** button, or the **Browse** button. You can use one of the following editor variables:

- **\${method}** - The FOP transformation method: **pdf**, **ps**, or **txt**.
- **\${fo}** - The input FO file.
- **\${out}** - The output file.
- **\${pd}** - The project directory.
- **\${frameworksDir}** - The path of the **frameworks** subdirectory of the Oxygen XML Developer plugin installation directory.
- **\${oxygenInstallDir}** - The Oxygen XML Developer plugin installation directory.
- **\${ps}** - The platform-specific path separator. It is used between the library files specified in the class path of the command line.

Output Encoding

The encoding of the FO processor output stream that is displayed in a results panel at the bottom of the Oxygen XML Developer plugin window.

Error Encoding

The encoding of the FO processor error stream that is displayed in a results panel at the bottom of the Oxygen XML Developer plugin window.

Profiler Preferences

This section explains the settings available for the XSLT/XQuery 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 84).

The following profiler 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

Determines the unit of time measurement. You can choose between milliseconds or microseconds.

Hotspot threshold

Hotspots are ignored below this specified amount (in milliseconds). For more information, see [Hotspots View](#) on page 775.

Ignore invocation less than

Invocations are ignored below this specified amount (in microseconds). For more information, see [Invocation Tree View](#) on page 774.

Percentage calculation

The percentage base that determines what time span percentages are calculated against. You can choose between the following:

- **Absolute** - Percentage values show the contribution to the total time.
- **Relative** - Percentage values show the contribution to the calling call.

XPath Preferences

To configure XPath options, [open the Preferences dialog box](#) and go to **XML > XSLT-FO-XQuery > XPath**.

Oxygen XML Developer plugin allows you to customize the following options:

Unescape XPath expression

When enabled, 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')]
```

XPath Default Namespace (only for XPath version 2.0)

Specifies the default namespace to be used for unprefixed element names. You can choose between the following four options:

- **No namespace** - If selected, Oxygen XML Developer plugin considers unprefixed element names of the evaluated XPath expressions as belonging to no namespace.
- **Use the default namespace from the root element** (default selection) - Oxygen XML Developer plugin considers unprefixed element names of the evaluated XPath expressions as belonging to the default namespace declared on the root element of the XML document you are querying.
- **Use the namespace of the root** - If selected, Oxygen XML Developer plugin considers unprefixed element names of the evaluated XPath expressions as belonging to the same namespace as the root element of the XML document you are querying.
- **This namespace** - If selected, you can use the corresponding text field to enter the namespace of the unprefixed elements.

Default prefix-namespace mappings

You can use this table to associate prefixes with namespaces. Use these mappings when you want to define them globally (not for each document). Use the **New** button to add mappings to the list and the **Delete** button to remove mappings.

XQuery Preferences

To configure the XQuery options, [open the Preferences dialog box](#) and go to **XML > XSLT-FO-XQuery > XQuery**.

The following generic XQuery preferences are available:

Validation engine

Allows you to select the processor that will be used to validate XQuery documents. If you are validating an XQuery file that has an associated validation scenario, Oxygen XML Developer plugin uses the processor specified in the scenario. If no validation scenario is associated, but the file has an associated transformation scenario, the processor specified in the scenario is used. If the processor does not support validation or if no scenario is associated, then the value from this combo box will be used as validation processor.

Size limit of Sequence view (MB)

When the result of an XQuery transformation is set as a sequence ([Present as a sequence option](#)) in the transformation scenario, the size of one chunk of the result that is fetched from the database in lazy mode in one step is set in this option. If this limit is exceeded, go to the [Sequence view](#) and click **More results available** to extract more data from the database.

Format transformer output

Specifies whether or not the output of the transformer is formatted and indented (pretty print).



Note: This option is ignored if you choose [Present as a sequence](#) (lazy extract data from a database) from the associated transformation scenario.

Create structure indicating the type nodes

If enabled, Oxygen XML Developer plugin takes the results of a query and creates an XML document containing copies of all items in the sequence, suitably wrapped.



Note: This option is ignored if you choose [Present as a sequence](#) (lazy extract data from a database) from the associated transformation scenario.

Saxon-HE/PE/EE Preferences

To configure global options for XQuery transformation and validation scenarios that use the **Saxon HE/PE/EE** engine, [open the Preferences dialog box](#) and go to **XML > XSLT-FO-XQuery > XQuery > Saxon-HE/PE/EE**.

Oxygen XML Developer plugin allows you to configure the following XQuery 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 used for XQuery transformation and validation scenarios.

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")

Allows you to choose how the *strip whitespaces* operation is handled. You can choose 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")

Allows you to set the optimization level. It is the value is an integer in the range of 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 enabled (default value), Saxon runs the XQuery transformation with the XQuery 3.0 support.

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 `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 result 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 for non-fatal errors are written (wherever 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>.

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 an 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**.

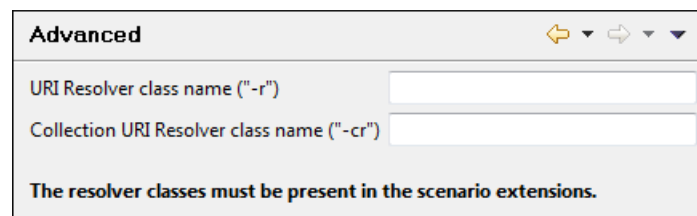


Figure 27: Saxon HE/PE/EE XQuery Advanced Preferences Panel

The advanced XQuery options that can be configured for the Saxon 9.6.0.7 XQuery transformer (all editions: Home Edition, Professional Edition, Enterprise Edition) are as follows:

- **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.

XSLT Preferences

To configure the XSLT options, [open the Preferences dialog box](#) and go to **XML > XSLT-FO-XQuery > XSLT**.

Oxygen XML Developer plugin offers the possibility of using an XSLT transformer implemented in Java (other than the XSLT transformers that come bundled with Oxygen XML Developer plugin). To use another XSLT transformer, specify the name of the transformer factory class. Oxygen XML Developer plugin sets this transformer factory class as the value of the Java property: `javax.xml.transform.TransformerFactory`.

The XSLT preferences page allows you to customize the following options:

JAXP XSLT Transformer - Value

Allows you to set the value of the `TransformerFactory` Java class.

Validation engine - XSLT 1.0

Allows you to select the XSLT engine to be used for validation of XSLT 1.0 documents.

Validation engine - XSLT 2.0

Allows you to select the XSLT engine to be used for validation of XSLT 2.0 documents.

Validation engine - XSLT 3.0

Allows you to select the XSLT engine to be 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>.

XSLT Editor Content Completion Options link

Use this link to switch to the [XSLT Content Completion preferences page](#), where you can configure the XSLT content completion options.

MSXML Preferences

To configure the MSXML options, [open the Preferences dialog box](#) and go to **XML > XSLT-FO-XQuery > XSLT > MSXML**.

The options in this preferences page for the MSXML 3.0 and 4.0 processors are as follows:

Validate documents during parse phase

If checked, and either the source or stylesheet document has a DTD or schema that its content can be checked against, validation is performed.

Do not resolve external definitions during parse phase

By default, MSXML 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, including:

- 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 behavior 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 in this preferences page for the MSXML.NET processor are as follows:

Enable XInclude processing

If checked, XInclude references will be resolved when MSXML.NET is used as the 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 that its content can be checked against, validation is performed.

Do not resolve external definitions during parse phase

By default, MSXML 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, including:

- 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 the .NET 1.X XSLT processor (`System.Xml.Xsl.XslTransform` class). It does not support escaping of characters as XML character references when they cannot be represented in the output encoding. This means that it will be outputted as ' '? '. Usually this happens when output encoding is set to ASCII. If 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 you to create multiple result documents using the [`exsl:document extension element`](#).

Use named URI resolver class

This option allows you 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 ( )` functions (during XSL transformation phase).

Assembly file name for URI resolver class

This option specifies a file name of the assembly where the specified resolver class can be found. The [Use named URI resolver class option](#) specifies a partially or fully qualified URI resolver class name (for example, `Acme.Resolvers.CacheResolver`). Such a name requires additional assembly specification using this option or the [Assembly GAC name for URI resolver class 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](#).

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](#).

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. Use this option if you want to use an external EXSLT.NET implementation instead of a built-in one.

Credential loading source xml

This option allows you 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 you 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).

Saxon-HE/PE/EE Preferences

To configure global options for XSLT transformation and validation scenarios that use the **Saxon HE/PE/EE** engine, [open the Preferences dialog box](#) and go to **XML > XSLT-FO-XQuery > XSLT > Saxon > Saxon-HE/PE/EE**.

Oxygen XML Developer plugin allows you to configure the following XSLT options for the Saxon 9.6.0.7 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")

Line numbers where errors occur are included in the output messages.

Debugger trace into XPath expressions (applies to debugging sessions)

Instructs the [XSLT Debugger](#) to *step into* XPath expressions.

Expand attributes defaults ("-expand")

Specifies whether or not the attributes defined in the associated DTD or XML Schema are expanded in the output of the transformation you are executing.

DTD validation of the source ("-dtd")

Specifies whether or not the source document will be validated against their associated DTD. You can choose from the following:

- **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")

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")

Allows you to choose how the *strip whitespaces* operation is handled. You can choose 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")

Allows you to set the optimization level. It is the value is an integer in the range of 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 following options are available for Saxon 9.6.0.7 Professional Edition (PE) and Enterprise Edition (EE) only:

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 `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 Developer plugin only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about [executing such a stylesheet on Saxonica's website](#).

The 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 `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 result 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 for non-fatal errors are written (wherever 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**.

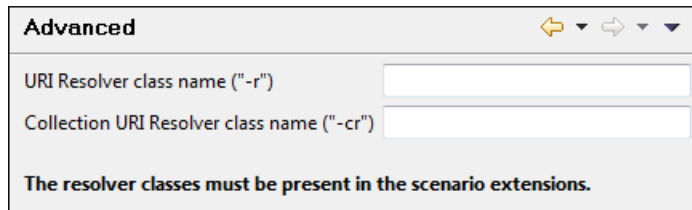


Figure 28: Saxon HE/PE/EE XSLT Advanced Preferences Panel

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.

Saxon6 Preferences

To configure the **Saxon 6** options, [open the Preferences dialog box](#) and go to **XML > XSLT-FO-XQuery > XSLT > Saxon > Saxon6**.

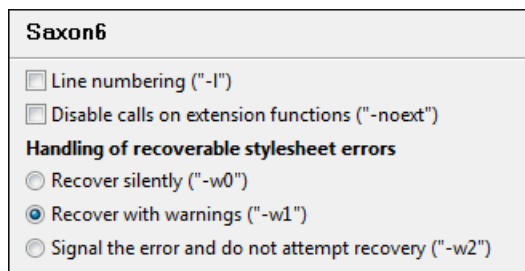


Figure 29: Saxon 6 XSLT Preferences Panel

The built-in Saxon 6 XSLT processor can be configured with the following options:

- **Line numbering** - Specifies whether or not line numbers are maintained and reported in error messages for the XML source document.
- **Disable calls on extension functions** - If enabled, external function calls are not allowed. Checking this 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, since they 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.

XSLTProc Preferences

To configure **XSLTProc** options, [open the Preferences dialog box](#) and go to **XML > XSLT-FO-XQuery > XSLT > XSLTProc**.

The following options are available in this preferences page:

- **Enable XInclude processing** - If checked, XInclude references will be resolved when XSLTProc is used as transformer in [XSLT transformation scenarios](#).
- **Skip loading the document's DTD** - If checked, the DTD specified in the DOCTYPE declaration will not be loaded.
- **Do not apply default attributes from document's DTD** - If checked, the default attributes declared in the DTD and not specified in the document are not included in the transformed document.

- **Do not use Internet to fetch DTD's, entities or docs** - If checked, the remote references to DTD's and entities are not followed.
- **Maximum depth in templates stack** - If this limit of maximum templates depth is reached the transformation ends with an error.
- **Verbosity** - If checked, the transformation will output detailed status messages about the transformation process in the **Warnings** view.
- **Show version of libxml and libxslt used** - If checked, Oxygen XML Developer plugin will display in the **Warnings** view the version of the **libxml** and **libxslt** libraries invoked by XSLTProc.
- **Show time information** - If checked, the **Warnings** view will display the time necessary for running the transformation.
- **Show debug information** - If checked, the **Warnings** view will display debug information about what templates are matched, parameter values, and so on.
- **Show all documents loaded during processing** - If checked, Oxygen XML Developer plugin will display in the **Warnings** view the URL of all the files loaded during transformation.
- **Show profile information** - If checked, Oxygen XML Developer plugin will display in the **Warnings** view a table with all the matched templates, and for each template will display: the match XPath expression, the template name, the number of template modes, the number of calls, the execution time.
- **Show the list of registered extensions** - If checked, Oxygen XML Developer plugin will display in the **Warnings** view a list with all the registered extension functions, extension elements and extension modules.
- **Refuses to write to any file or resource** - If checked, the XSLTProc processor will not write any part of the transformation result to an external file on disk. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.
- **Refuses to create directories** - If checked, the XSLTProc processor will not create any directory during the transformation process. If such an operation is requested by the processed XSLT stylesheet the transformation ends with a runtime error.

XML Structure Outline Preferences

To configure options in regards to the **Outline** view, [open the Preferences dialog box](#) and go to **XML Structure Outline**. It contains the following options:

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 Developer 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 Developer plugin includes 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 Developer 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 Developer 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 Developer plugin behaves optimally in most working environments.



Important: 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 Developer plugin has an extensive set of options that you can configure. When Oxygen XML Developer plugin is installed, these options are set to default values. You can provide a different set of default values for an installation using an 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 Developer plugin and [open the Preferences dialog box](#).
3. Go through the options and set them to the desired defaults.
4. Go back to the main preferences page and click **Export Global Options** to create an XML options file.

Using Customized Default Options

There are two methods that you can use to configure an Oxygen XML Developer plugin installation to use the customized default options from the created XML options file:

- **Copy the XML Options File to the Installation Directory**

In the `[OXYGEN_INSTALL_DIR]`, create a folder called `preferences` and copy the created XML options file into it (for example:

`[ECLIPSE-INSTALL-DIR]/plugins/com.oxygenxml.developer/preferences/default.xml`, or if the plugin was installed as a drop-in:

`[ECLIPSE-INSTALL-DIR]/dropins/com.oxygenxml.developer/preferences/default.xml`).

- **Specify a Path to the XML Options File in a Startup Parameter**

Set the path to the XML options file as the value of the `com.oxygenxml.default.options` system property in the Eclipse configuration file (`[ECLIPSE-INSTALL-DIR]/configuration/config.ini`). The path can be specified with any of the following:

- A URL or file path relative to the application installation folder. For example:

```
com.oxygenxml.default.options=file\default.xml
```

This will make Oxygen XML Developer plugin look for `default.xml` inside the installation folder (for example:

`[ECLIPSE-INSTALL-DIR]/plugins/com.oxygenxml.developer/preferences/default.xml`, or if the plugin was installed as a drop-in:

`[ECLIPSE-INSTALL-DIR]/dropins/com.oxygenxml.developer/preferences/default.xml`).

- A system variable that specifies the file path. For example:

```
com.oxygenxml.default.options=file\: ${system(CONFIG)}/default.xml
```

- An environmental variable that specifies the file path. For example:

```
com.oxygenxml.default.options=file\: ${env(CONFIG)}/default.xml
```



Note: In the Eclipse configuration file, the backslash (\) is considered a special character. Therefore, use forward slashes for separators inside the file path.

Importing/Exporting/Resetting Global Options

Actions for importing, exporting, and resetting global options are available in the preferences page of the Oxygen XML Developer plugin. To open this page, [open the Preferences dialog box](#). The export operation allows 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](#). This action also resets the transformation and validation scenarios to the default scenarios and clears recently used file templates.

Import Global Options

Allows you to import a set of *Global Options* from an exported XML properties 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 properties 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.

Associating a File Extension with Oxygen XML Developer plugin

To associate a file extension with Oxygen XML Developer plugin on Windows:

1. Go to the Windows **Start** menu and open **Control Panel**.
2. Go to **Default Programs**.
3. Click **Associate a file type or protocol with a program**.
4. Click the file extension you want to associate with Oxygen XML Developer plugin, then click the **Change program** button.
5. In the subsequent dialog box, browse for and choose Oxygen XML Developer plugin.

To associate a file extension with Oxygen XML Developer plugin on Mac OS:

1. In **Finder**, select a file and from the contextual menu select **Get Info**.
2. In the **Open With** subsection, select **Other** from the application combo box.
3. Browse to and select Oxygen XML Developer plugin.
4. Select the **Always Open With** option, then click **Add**.

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

Loads a set of transformation scenarios from a project file, framework options file, or exported scenarios file.

Export Global Transformation Scenarios

Stores a set of global (not project-level) transformation scenarios in a specialized *scenarios* file.

Import Global Validation Scenarios

Loads a set of validation scenarios from a project file, framework options file, or exported scenarios file.

Export Global Validation Scenarios

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 multiple files, the notation is expanded at the execution of the command so that the same command has different effects depending on the actual file.

Oxygen XML Developer plugin includes a variety of built-in editor variables. You can also create your own [custom editor variables](#) by using the [Custom Editor Variables preferences page](#).

You can use the following editor variables in Oxygen XML Developer plugin commands of external engines or other external tools, in transformation scenarios, and in validation scenarios:

- `${af}` - The local file path of the ZIP archive that includes the current edited document.
- `${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.
- `${afne}` - The file name (with file extension, for example `.zip` or `.epub`, but without parent directory) of the zip archive that includes the current edited file.
- `${afu}` - The URL path of the ZIP archive that includes the current edited document.
- `${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value')}` - To prompt for values at runtime, use the `ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value')` editor variable. You can set the following parameters:
 - `'message'` - The displayed message. Note the quotes that enclose the message.
 - `type` - Optional parameter, with one of the following values:



Note: The title of the dialog box will be determined by the type of parameter and as follows:

- For `url` and `url_relative` parameters, the title will be the name of the parameter and the value of the `'message'`.
- For the other parameters listed below, the title will be the name of that respective parameter.
- If no parameter is used, the title will be "Input".

Parameter	
url	Format: <code>\${ask('message', url, 'default_value')}</code>
	Description: Input is considered a URL. Oxygen XML Developer plugin checks that the provided URL is valid.
	Example: • <code>\${ask('Input URL', url)}</code> - The displayed dialog box has the name Input URL. The expected input type is URL. • <code>\${ask('Input URL', url, 'http://www.example.com')}</code> - The displayed dialog box has the name Input URL. The expected input type is URL. The input field displays the default value <code>http://www.example.com</code>.

Parameter	
password	Format: <code>\${ask('message', password, 'default')}</code>
	Description: The input is hidden with bullet characters.
	Example: • <code>\${ask('Input password', password)}</code> - The displayed dialog box has the name 'Input password' and the input is hidden with bullet symbols. • <code>\${ask('Input password', password, 'abcd')}</code> - The displayed dialog box has the name 'Input password' and the input hidden with bullet symbols. The input field already contains the default <code>abcd</code> value.
generic	Format: <code>\${ask('message', generic, 'default')}</code>
	Description: The input is considered to be generic text that requires no special handling.
	Example: • <code>\${ask('Hello world!')}</code> - The dialog box has a Hello world! message displayed. • <code>\${ask('Hello world!', generic, 'Hello again!')}</code> - The dialog box has a Hello world! message displayed and the value displayed in the input box is 'Hello again!'.
relative_url	Format: <code>\${ask('message', relative_url, 'default')}</code>
	Description: Input is considered a URL. Oxygen XML Developer plugin tries to make the URL relative to that of the document you are editing.  Note: If the <code>\$ask</code> editor variable is expanded in content that is not yet saved (such as an <i>untitled</i> file, whose path cannot be determined), then Oxygen XML Developer plugin will transform it into an absolute URL.
	Example: • <code>\${ask('File location', relative_url, 'C:/example.txt')}</code> - 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: <code>\${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</code>
	Description: Displays a dialog box that offers a drop-down menu. The drop-down menu is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated value (<code>real_value</code>).  Note: The 'default' parameter specifies the default selected value and can match either a key or a value.
	Example: <code>\${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}</code> - 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 <code>osx</code> key. However, the same result could be obtained if the default value is indicated by <code>Mac OS X</code>, as in the following example: <code>\${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'Mac OS X')}</code> <code>\${ask('Mobile OS', combobox, ('win':'Windows Mobile';'ios':'iOS';'and':'Android'), 'Android')}</code>
editable_combobox	Format: <code>\${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</code>
	Description: Displays a dialog box that offers a drop-down menu with editable elements. The drop-down menu is populated with the given <code>rendered_value</code> values. Choosing such a value will return its associated real value (<code>real_value</code>) 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: <code>\${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}</code> - 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.

Parameter	
radio	Format: <code>\${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}</code> 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: • <code>\${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}</code> - 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.
- `${caret}` - The position where the cursor is located. This variable can be used in a code template, in **Author** mode operations, or in a selection plugin.
- `${cf}` - Current file as file path, that is the absolute file path of the current edited document.
- `${cfd}` - Current file folder as file path, that is the path of the current edited document up to the name of the parent folder.
- `${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.
- `${cfn}` - Current file name without extension and without parent folder. The current file is the one currently opened and selected.
- `${cfne}` - Current file name with extension. The current file is the one currently opened and selected.
- `${configured.ditaot.dir}` - The default directory of the DITA Open Toolkit distribution, as configured in the [DITA preferences page](#).
- `${cp}` - Current page number. Used to display the current page number on each printed page in the **Editor / Print** Preferences page.
- `${currentFileURL}` - Current file as URL, that is the absolute file path of the current edited document represented as URL.
- `${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>.

- `${dbgXML}` - The local file path to the XML document that is current selected in the Debugger source combo box (for tools started from the XSLT/XQuery Debugger).
- `${dbgXSL}` - The local file path to the XSL/XQuery document that is current selected in the Debugger stylesheet combo box (for tools started from the XSLT/XQuery Debugger).
- `${dita.dir.url}` - A special local contextual editor variable that gets expanded only in the **Libraries** dialog box that is accessible from the **Advanced** tab of DITA transformation scenarios. The **Libraries** dialog box allows you to specify additional libraries (jar files or additional class paths) to be used by the transformer. This `${dita.dir.url}` editor variable gets expanded to the value of the `dita.dir` parameter from the **Parameters** tab of the DITA transformation scenario.
- `${ds}` - The path of the detected schema as a local file path for the current validated XML document.
- `${dsu}` - The path of the detected schema as a URL for the current validated XML document.
- `${env(VAR_NAME)}` - Value of the VAR_NAME environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the `${system(var.name)}` editor variable.

- `${framework(fr_name)}` - The path (as URL) of the `fr_name` framework.
- `${framework}` - The path (as URL) of the current framework, as part of the `[OXYGEN_INSTALL_DIR]/frameworks` directory.
- `${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 Developer plugin employs the following algorithm when searching for a given framework name:

- All frameworks are sorted, from high to low, according to their **Priority** setting from the [Document Type 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 Developer plugin options.
 - Additional frameworks added in the [Locations preferences page](#).
 - Frameworks installed using the add-ons support.
 - Frameworks found in the [main frameworks location](#) (**Default** or **Custom**).
- `${frameworkDir}` - The path (as file path) of the current framework, as part of the `[OXYGEN_INSTALL_DIR]/frameworks` directory.
- `${frameworks}` - The path (as URL) of the `[OXYGEN_INSTALL_DIR]` directory.
- `${frameworksDir}` - The path (as file path) of the `[OXYGEN_INSTALL_DIR]/frameworks` directory.
- `${home}` - The path (as URL) of the user home folder.
- `${homeDir}` - The path (as file path) of the user home folder.
- `${i18n(key)}` - Editor variable used only at framework level to allow translating names and descriptions of **Author** mode actions in multiple actions.
- `${id}` - Application-level unique identifier. It is a short sequence of 10-12 letters and digits that is not guaranteed to be universally unique.
- `${makeRelative(base,location)}` - Takes two URL-like paths as parameters and tries to return a relative path. A use-case would be to insert content references to a certain reusable component when defining code templates.

Example:

```
${makeRelative(${currentFileURL}, ${dictionaryURL}#gogu)}
```

- `${oxygenHome}` - Oxygen XML Developer plugin installation folder as URL.
- `${oxygenInstallDir}` - Oxygen XML Developer plugin installation folder as file path.
- `${pd}` - The file path for the parent folder of the current project selected in the **Navigator** view.
- `${pdu}` - The URL for the parent folder of the current project selected in the **Navigator** view.
- `${pn}` - Current project name.
- `${ps}` - Path separator, which is the separator that can be used on the current platform (Windows, OS X, Linux) between library files specified in the class path.
- `${selection}` - The current selected text content in the current edited document. This variable can be used in a code template, in **Author** mode operations, or in a selection plugin.
- `${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=value`. If you are looking for operating system environment variables, use the `${env(VAR_NAME)}` editor variable instead.
- `${timeStamp}` - Time stamp, that is the current time in Unix format. For example, it can be used to save transformation results in multiple output files on each transformation.
- `${tp}` - Total number of pages in the document. Used to display the total number of pages on each printed page in the **Editor** / **Print** Preferences page.
- `${tsf}` - The transformation result file path. If the current opened file has an associated scenario that specifies a transformation output file, this variable expands to it.
- `${uuid}` - Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java [UUID](#) class.
- `${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)}')}
```

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, follow these steps:

1. [Open the Preferences dialog box](#) and go to **Custom Editor Variables**.
2. Click the  **New** button at the bottom of the table.
3. Use the subsequent dialog box to specify the **Name**, **Value**, and **Description** for the new editor variable.
4. Click **OK** to save your configuration.

Related information

[Editor Variables](#) on page 99

Localizing of the User Interface

To localize the Oxygen XML Developer 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 <https://www.oxygenxml.com/InstData/Developer/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 <https://www.oxygenxml.com/download.html> and download the zip archive with the plugin language pack. Unzip the downloaded zip archive in the **dropins** subdirectory of the Eclipse install directory. Restart Eclipse.

If your operating system is running in the language you want to start Eclipse in (for example, you are using Japanese version of Windows, and you want to start Eclipse in Japanese), Oxygen XML Developer plugin matches the appropriate language from the language pack. However, if your operating system is running in a language other than the one you want to start Eclipse in (for example, you are using the English version of Windows, and you want to start Eclipse in Japanese, if you have the required operating system language support including the keyboard layouts and input method editors installed), specify the `-nl <locale>` command line argument when you launch Eclipse. Oxygen XML Developer plugin uses the translation file that matches the specified `<locale>`.

You can also localize the Eclipse plugin to a different language than the initial languages in the language pack. Duplicate the `plugin.properties` file from the Oxygen XML Developer plugin installation directory, translate all the keys in the file and change its name to `plugin_<locale>.properties`.

Chapter

5

Perspectives

Topics:

- [*oXygen XML Perspective*](#)
- [*oXygen XSLT Debugger Perspective*](#)
- [*oXygen XQuery Debugger Perspective*](#)
- [*oXygen DB Perspective*](#)

An Oxygen XML Developer plugin perspective is an interface layout geared towards a specific use. The Oxygen XML Developer 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 Developer 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 Developer 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 Developer plugin. Most of the menus are common for all types of documents. However, Oxygen XML Developer 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.

Editor Pane

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

Views

Oxygen XML Developer 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.
- **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.
- **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.
- **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/XQuery Builder 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.

Related information

[oXygen XSLT Debugger Perspective](#) on page 107

[oXygen XQuery Debugger Perspective](#) on page 107

[oXygen DB Perspective](#) on page 108

oXygen 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 **Window > Open Perspective > Other > <oXygen/> XSLT Debugger**.

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 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 an 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 can 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 Developer plugin, go to https://www.oxygenxml.com/demo/XSLT_Debugger.html.

Related information

oXygen XML Perspective on page 106

oXygen XQuery Debugger Perspective on page 107

oXygen DB Perspective on page 108

oXygen 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 **Window > Open Perspective > Other > <oXygen/> XQuery Debugger**.

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 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 can 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 Developer plugin, go to https://www.oxygenxml.com/demo/XQuery_Debugger.html.

Related information

[oXygen XML Perspective](#) on page 106

[oXygen XSLT Debugger Perspective](#) on page 107

[oXygen DB Perspective](#) on page 108

oXygen DB Perspective

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

The **Database** perspective offers various helpful features, including:

- Support for browsing multiple connections at the same time.
- Support for both *Relational* and *Native XML* databases.
- Browsing the structure of databases.
- Viewing tables from databases.
- Inspecting or modifying data.
- Specifying XML Schemas for XML fields.
- SQL execution.
- XQuery execution.
- Data export to XML.

Supported Databases

Oxygen XML Developer plugin supports numerous types of databases, including:

- 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)
- SharePoint (CMS)



Note: For the databases marked with "Enterprise edition only", the XML capabilities are only available in the Enterprise edition of Oxygen XML Developer plugin. For a detailed feature matrix that compares the Academic, Professional, and Enterprise editions of Oxygen XML Developer plugin [go to the Oxygen XML Developer plugin website](#).

Supported Capabilities

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.



Note: The non-XML capabilities are available in the Enterprise, Academic, and Professional editions of Oxygen XML Developer 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 [Database Connection Support](#) on page 690.

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 Developer plugin editor panel.
- Handling the values from **XML Type** columns as XML instance documents that can be opened and edited in an Oxygen XML Developer plugin editor panel.
- Validating an XML instance document added to an XML Type (column of a table, etc.)



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

Layout of the Database Perspective

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 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.

Related information

[oXygen XML Perspective](#) on page 106

[oXygen XSLT Debugger Perspective](#) on page 107

[oXygen XQuery Debugger Perspective](#) on page 107

[Data Source Explorer View](#) on page 687

Chapter

6

Editing Modes

Topics:

- [Text Editing Mode](#)
- [Grid Editing Mode](#)
- [Design Editing Mode \(XML Schema Diagram Editor\)](#)

The main editing area in Oxygen XML Developer 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 Developer 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.
- **Design** - This mode is found in the schema editor and represents the schema as a diagram.

The default editing mode that will be initially opened for each type of document can be set in two ways:

- If the [Allow Document Type specific edit mode setting to override the general mode setting option](#) is selected in the **Edit Modes** preferences page, then the edit mode specified in the [Document Type configuration dialog box](#) is used when that particular type of document is initially opened.
- If the [Allow Document Type specific edit mode setting to override the general mode setting option](#) is not selected, then the edit mode specified in the [table in the Edit Modes preferences page](#) is used when that particular type of document is initially opened.

Text Editing Mode

The **Text** mode editor in Oxygen XML Developer plugin is designed to be a simple, yet powerful, XML source editor. It provides support to help you edit, transform, and debug XML-based documents. It is similar to other common text editors, but Oxygen XML Developer plugin also includes specialized editing actions, a powerful [Content Completion Assistant](#), a helpful [Outline view](#), and many other unique features.

Related information

[Editing XML Documents in Text Mode](#) on page 161

Navigating the Document Content in Text Mode

Oxygen XML Developer plugin includes some useful features to help you navigate XML documents in **Text** mode.

Using the Keyboard

Oxygen XML Developer plugin allows you to quickly navigate through a document using the **Ctrl + CloseBracket (Command + CloseBracket on OS X)** key to go to the next XML node and **Ctrl + OpenBracket (Command + OpenBracket on OS X)** to go to the previous one.

To navigate one word forward or backwards, use **Ctrl + RightArrow (Command + RightArrow on OS X)**, and **Ctrl + LeftArrow (Command + LeftArrow 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.

Navigation Shortcuts

Oxygen XML Developer plugin includes some keyboard shortcuts to help you quickly navigate to a particular modification. They are also available as actions in the **Navigation** menu.

- **Ctrl+Q - Last Edit Location** - Moves the cursor to the last modification in any opened document.
- **Alt+LeftArrow (Command+OpenBracket on OS X) - Back** - Moves the cursor to the previous position.
- **Alt+RightArrow (Command+CloseBracket on OS X) - Forward** - Moves the cursor to the next position.

Navigating with the Outline View

Oxygen XML Developer plugin includes a very useful [Outline view](#) that displays a hierarchical tag overview of the currently edited XML Document.

You can use this view to quickly navigate through the current document by selecting nodes in the outline tree. It is synchronized with the editor area, so when you make a selection in the **Outline** view, the corresponding nodes are highlighted in the editor area.

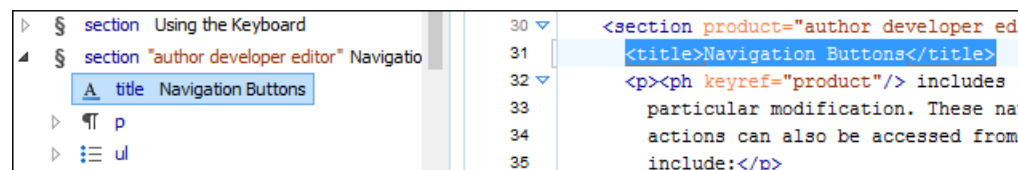


Figure 30: Outline View Navigation in Text Mode

Using the Breadcrumb to Navigate

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.

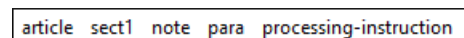


Figure 31: Breadcrumb in Text Mode

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 and navigates to it in the editor area.

Navigating with the Go To Dialog Box

In **Text** mode, you can navigate precisely to a location in the document you are editing by using the **Go To Line (Ctrl+L (Command+L on OS X))** action that is available in the **Navigation** menu.

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.

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.

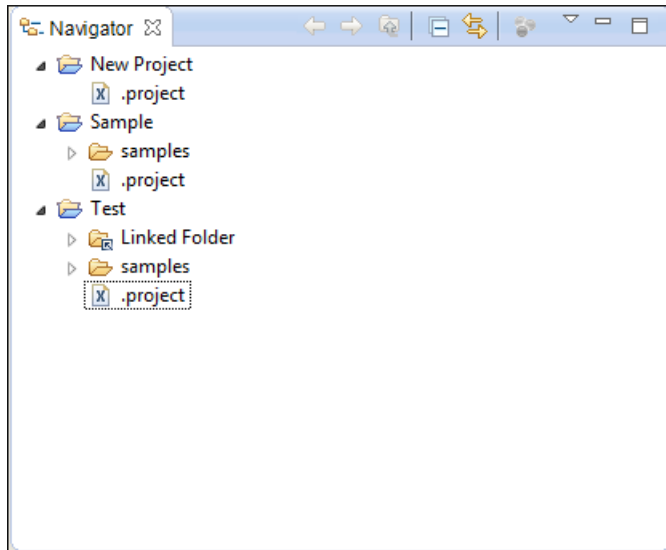


Figure 32: Navigator View

By default, the view is positioned on the left side of the Oxygen XML Developer plugin window, above the **Outline view**. If the view has been closed, it can be reopened at any time from the **Window > Show View** menu.

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).

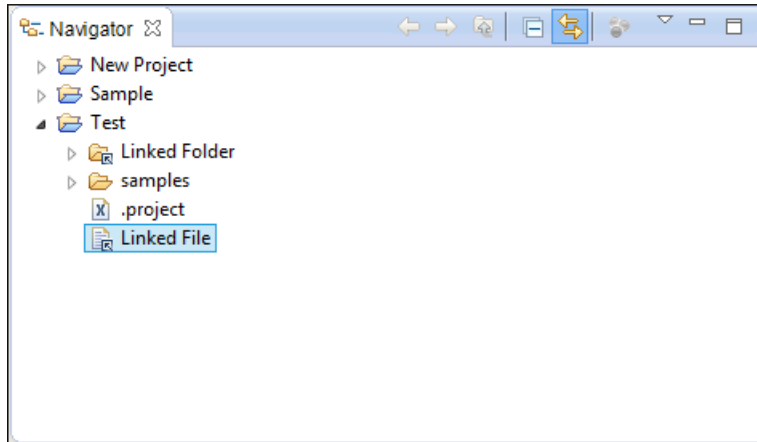


Figure 33: Navigator View with Examples of the Two Types of Resources

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 Developer plugin and the *Recycle Bin* is active) the file is moved to *Recycle Bin*.

Moving Files and Folders

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

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

Renaming Files and Folders

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

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

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

Locating and Opening Files

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

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

Saving the Project

The project file is automatically saved every time the content of the **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 Developer plugin in the **Package Explorer** view or in the **Navigator** view can be checked to be XML well-formed or validated against a schema (DTD, XML Schema, Relax NG, Schematron or NVDL) with one of the following contextual menu actions found in the **Validate** submenu:



Check Well-Formedness

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



Validate

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

Validate with Schema

Validates the selected file or files against a specified schema.



Configure Validation Scenario(s)

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

✗ 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 Developer plugin in the **Package Explorer** view or in the **Navigator** view can be transformed in one step with one of the following actions available from contextual menu in the **Transform** submenu:

Apply Transformation Scenario(s)

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

Configure Transformation Scenario(s)

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

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 Developer 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** submenu:

Rename resource

Allows you to change the name of a resource.

Move resource

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

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 files in the corresponding 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.

Related information

[Working with EPUB](#) on page 683

Outline View in Text Mode

The **Outline** view in **Text** mode displays a general tag overview of the currently edited XML Document. When you edit a document, the **Outline** view dynamically follows the changes that you make, displaying the node that you modify. This functionality gives you great insight on the location of your modifications in the current document. It also shows the correct hierarchical dependencies between elements. This makes it easy for you to be aware of the document structure and the way element tags are nested.

Outline View Features

The **Outline** view allows you to:

- Quickly navigate through the document by selecting nodes in the **Outline** tree.
- 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 editor area. It is synchronized with the editor area, so when you make a selection in the editor area, the corresponding nodes are highlighted in the **Outline** view, and vice versa.
- View document errors, as they are highlighted in the **Outline** view. A tooltip also provides more information about the nature of the error when you hover over the faulted element.

Outline View Interface

By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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.

Drop and Drop Actions in the 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 node, 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, a copy operation will be performed instead of a move.

The drag and drop actions in the **Outline** view can be [disabled and enabled from a Preferences page](#).

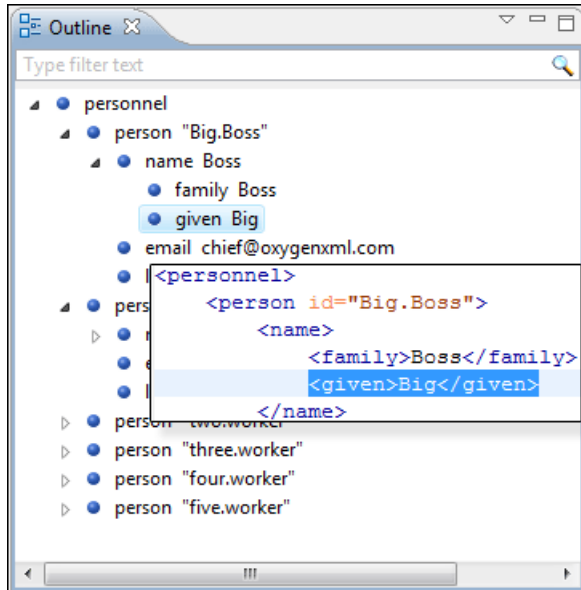


Figure 34: Outline View in Text Mode

Outline View Filters in Text Mode

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](#).

Outline View Contextual Menu Actions in Text Mode

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.

Attributes View in Text Mode

The **Attributes** view presents all the attributes of the current element determined by the schema of the document. By default, it is located on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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 are rendered with a bold font, and their values with a plain font.
- 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.

To edit the value of the corresponding attribute, double-click a cell in the **Value** column. 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 either select the value from a list or manually enter it.

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.

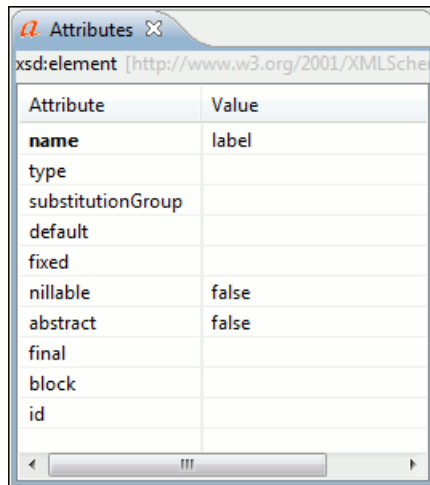


Figure 35: Attributes View

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.

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.

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. By default, it is located on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

Annotation Panel

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

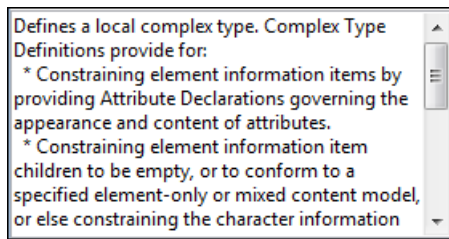


Figure 38: Annotation panel

Elements View in Text Mode

The **Elements** view presents a list of all defined elements that are valid at the current cursor position according to the schema associated to the document. By default, it is located on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

Double-clicking any of the listed elements inserts that element into the edited document, at the current cursor position.

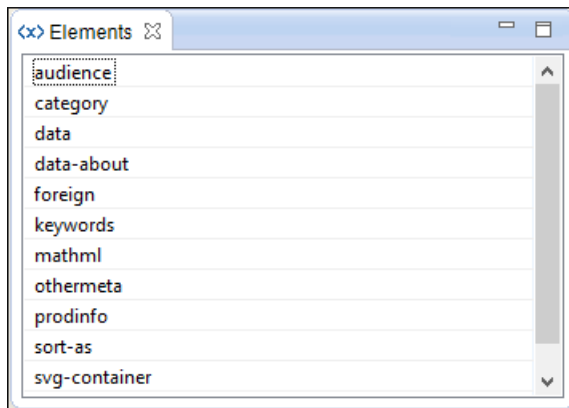


Figure 39: Elements View in Text Mode

Entities View

Entities provide abbreviated entries that can be used in XML files when there is a need of repeatedly inserting certain characters or large blocks of information. An *entity* is defined using the ENTITY statement either in the DOCTYPE declaration or in a DTD file associated with the current XML file.

There are three types of entities:

- *Built-in or Predefined* - Entities that are part of the predefined XML markup (<, >, &, ', ").
- *Internal* - Defined in the DOCTYPE declaration header of the current XML.
- *External* - Defined in an external DTD module included in the DTD referenced in the XML DOCTYPE declaration.



Note: If you want to add internal entities, you would need to switch to the Text editing mode and manually modify the DOCTYPE declaration. If you want to add external entities, you need to open the DTD module file and modify it directly.

The **Entities** view displays a list with all entities declared in the current document, as well as built-in ones. By default, it is located on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

Double-clicking one of the entities will insert it at the current cursor position in the XML document. You can also sort entities by name and value by clicking the column headers.

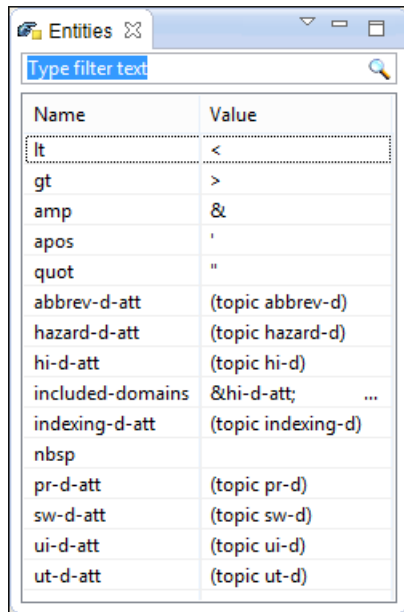


Figure 40: Entities View

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.

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 **Results** view is automatically opened when certain actions generate result messages. Those actions include the following:

- [Validation actions](#)
- [Transformation actions](#)
- [Check Spelling in Files action](#)
-
-
- [Search References action](#)
- [SQL results](#)

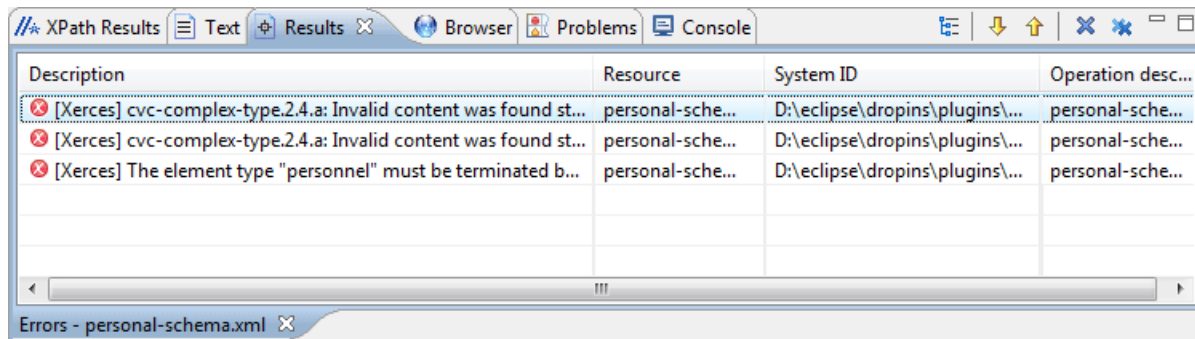


Figure 41: Results View

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 this view:

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 various 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 42: Example of Coloring XML Tags by Prefix

Related information

[Changing the colors displayed in the Text Mode Editor](#) on page 67

Presenting Validation Errors in Text Mode

Oxygen XML Developer plugin can be configured to [automatically validate documents](#) while editing in the **Text** mode, and actions are also available to [manually validate documents](#) on-request.

A line with a validation issue is marked in the editor panel by underlining the problem region with a colored line, according to the type of issue. The issues are also marked with a colored marker in the right vertical stripe.

- **Validation Errors** - By default, underlined with a red squiggly line in the editing pane and a red marker in the right vertical stripe.
- **Validation Warnings** - By default, underlined with a yellow squiggly line in the editing pane and a yellow marker in the right vertical stripe.
- **Validation Info** - By default, underlined with a blue squiggly line in the editing pane and a blue marker in the right vertical stripe.

Hovering over a validation issue presents a tooltip message with more details about the problem and [possible quick fixes](#) (if available for that issue).

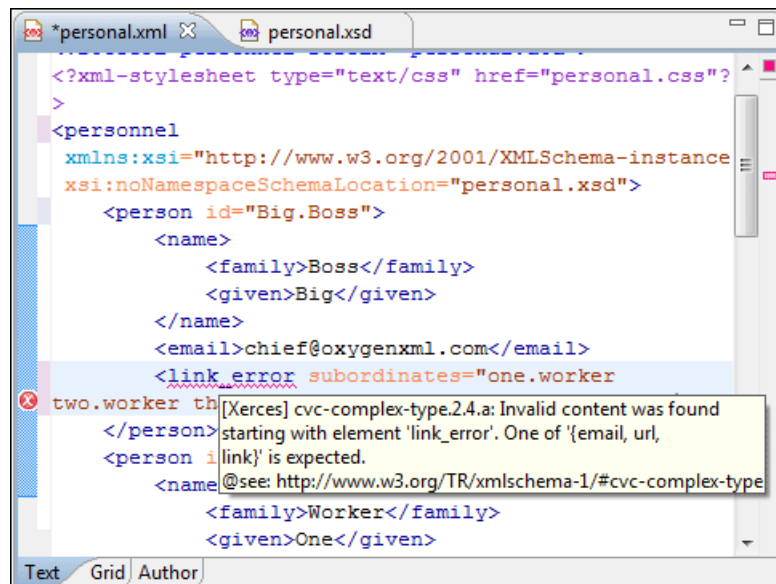


Figure 43: Presenting Validation Errors in Text Mode

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

Upper Part of the Stripe

A success indicator square will turn green if the validation is successful or only info messages are found, red if validation errors are found, or yellow if only validation warnings are found. More details about the issues are displayed in a tool tip when you hover over indicator square. If there are numerous problems, only the first three are presented in the tool tip.

Middle Part of the Stripe

Errors are depicted with red markers, warnings with yellow markers, and info message with blue markers. If you want to limit the number of markers that are displayed, [open the Preferences dialog box](#), go to **Editor > Document checking**, and specify the desired limit in the [Maximum number of validation highlights option](#).

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

Bottom Part of the Stripe

Two navigation arrows () allow you to jump to the next or previous issue. The same actions can be triggered from **Document > Automatic validation > Next error (Ctrl + Period (Command + Period on OS X))** and **Document > Automatic validation > Previous error (Ctrl + Comma (Command + 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 the **View** preferences page).

If you want to see all the validation messages grouped in the [Results view](#), you should use the  **Validate** action from the toolbar or **XML** menu. This action also collects the validation messages and displays them in the **Problems** view if the validated file is in the current workspace or in a custom **Errors** view if the validated file is outside the workspace.

Related information

[Validating XML Documents Against a Schema](#) on page 194

[Presenting Schematron Validation Issues](#)

Grid Editing Mode

The Oxygen XML Developer plugin **Grid** editing mode displays the XML document as a structured grid of nested tables where the text content can be modified without directly interacting with the XML markup. This is helpful for non-technical users who want to edit text content without modifying the XML markup. You can easily expand or collapse elements within the table and the document structure can be changed with simple drag/drop or copy/paste operations.

To switch to this mode, select **Grid** at the bottom of the editing area.

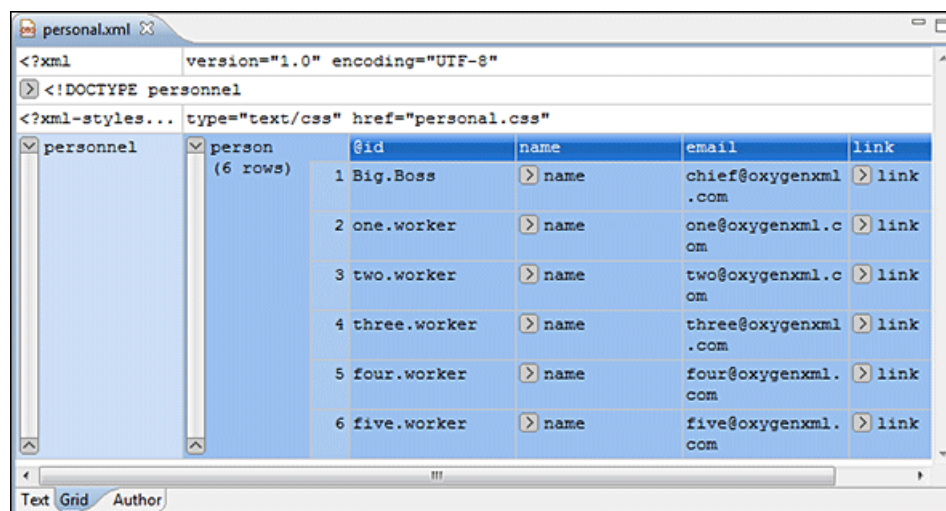


Figure 44: Grid Editing Mode

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

Related information

[Editing XML Documents in Grid Mode](#) on page 189

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.

<?xml	version="1.0" encoding="UTF-8"				
test	table	tr	@id	first	last
		(3 rows)	1	10001	Jhon Doe
			2	10002	Mark Ewing
			3	10003	Dave Flint

Figure 45: Grid Layout

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

<?xml	version="1.0" encoding="UTF-8"				
test	table	tr	@id	10001	
			first	Jhon	
			last	Doe	
		tr	@id	10002	
			first	Mark	
			last	Ewing	
		tr	@id	10003	
			first	Dave	
			last	Flint	

Figure 46: 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 2: 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.
UpArrow/PageUp	Navigates toward the beginning of the document.
DownArrow/PageDown	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 + UpArrow (Command + UpArrow on OS X)** - scrolls the grid upwards.
- **Ctrl + DownArrow (Command + DownArrow on OS X)** - scrolls the grid downwards.
- **Ctrl + LeftArrow (Command + LeftArrow on OS X)** scrolls the grid to the left.
- **Ctrl + RightArrow (Command + RightArrow 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 with a bidirectional 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 (Command + 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.

<?xml	version="1.0" encoding="UTF-8"
sample (9 rows)	#text
1	عندما يريد العالم أن يتكلم، فهو يتحدث بلغة يونيكود.
2	Quan el món vol conversar, parla Unicode
3	כאשר העולם רוצה לדבר, הוא מדבר ב-Unicode
4	Ha a világ beszélni akar, azt Unicode-ul mondja
5	Quando il mondo vuole comunicare, parla Unicode
6	世界的に話すなら、Unicodeです。
7	세계를 향한 대화, 유니코드로 하십시오
8	Når verden vil snakke, snakker den Unicode
9	Når verda ønskjer å snakke, talar ho Unicode

Figure 47: Default left to right text orientation

<?xml		"version="1.0" encoding="UTF-8"
sample		#text
(9 rows)	1	عندما يريد العالم أن يتكلم، فهو يتحدث بلغة يونيكود.
	2	Quan el món vol conversar, parla Unicode
	3	כאשר העולם רוצה לדבר, הוא מדבר ב-Unicode
	4	Ha a világ beszélni akar, azt Unicode-ul mondja
	5	Quando il mondo vuole comunicare, parla Unicode
	6	◦ 世界的に話すなら、Unicode です
	7	세계를 향한 대화, 유니코드로 하십시오
	8	Når verden vil snakke, snakker den Unicode
	9	Når verda ønskjer å snakke, talar ho Unicode

Figure 48: Right to left text orientation

Design Editing Mode (XML Schema Diagram Editor)

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

Oxygen XML Developer plugin provides a simple and expressive 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.

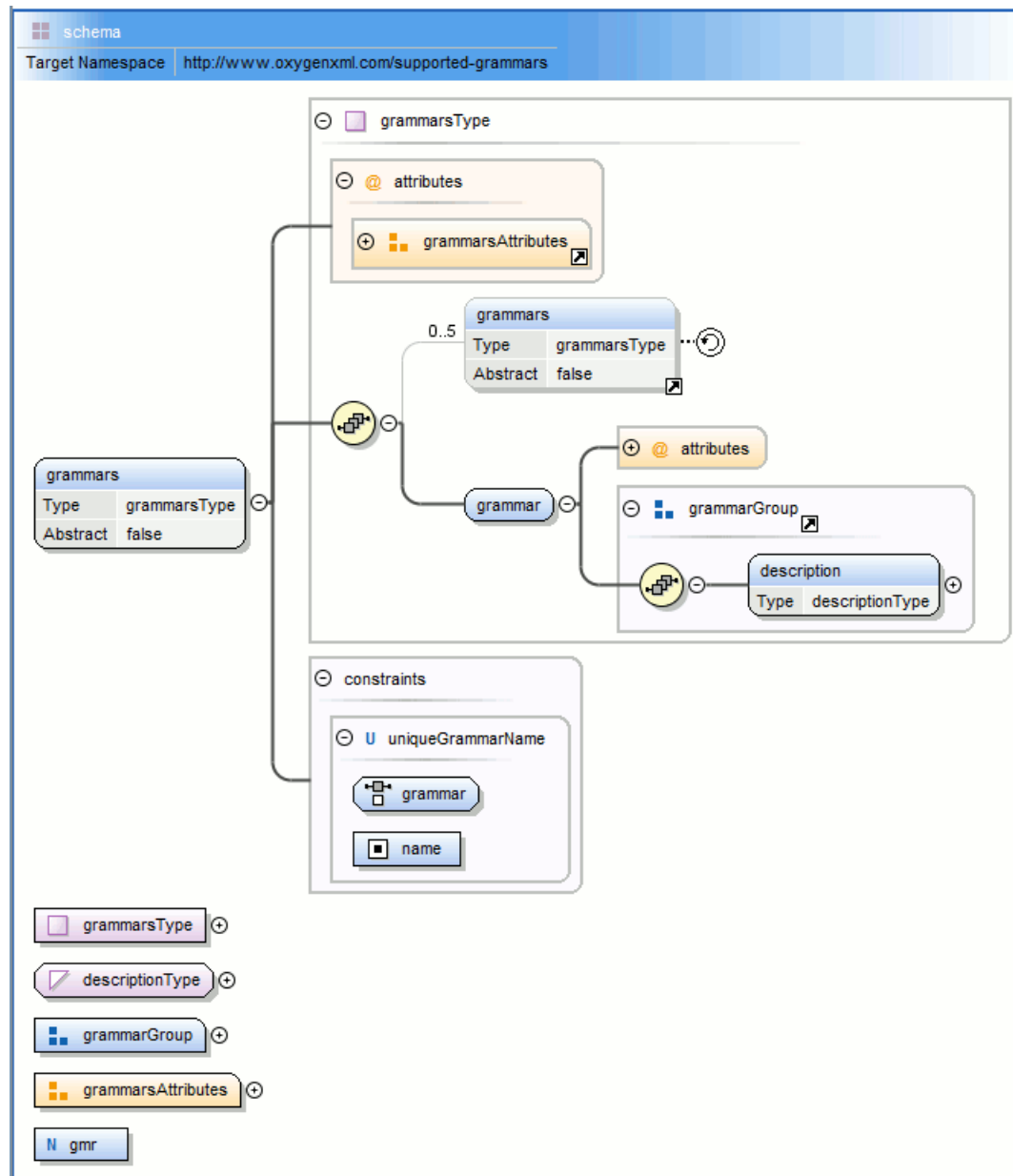


Figure 49: XML Schema Diagram

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

Navigation in the XML Schema Design Mode

The following editing and navigation features work for all types of schema components in the XML Schema **Design** mode:

- 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 jump 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 another position and from one schema to another using the **Cut/Copy** and **Paste/Paste as Reference** actions.
 - Go to the definition of an element or attribute with the **Show Definition** action.
 - You can expand and see the contents of the imports/includes/redefines in the diagram. In order to edit components from other schemas the schema for each component will be opened as a separate file in Oxygen XML Developer plugin.
-  **Tip:** If an XML Schema referenced by the current opened schema was modified on disk, the change will be detected and you will be asked to refresh the current schema contents.

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

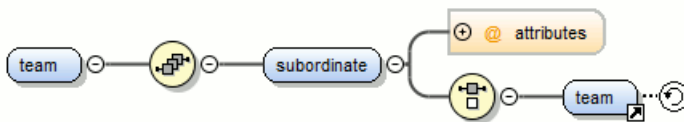


Figure 50: Recursive Reference

XML Schema Outline View

The **Outline** view for XML Schemas presents all the global components grouped by their location, namespace, or type. By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

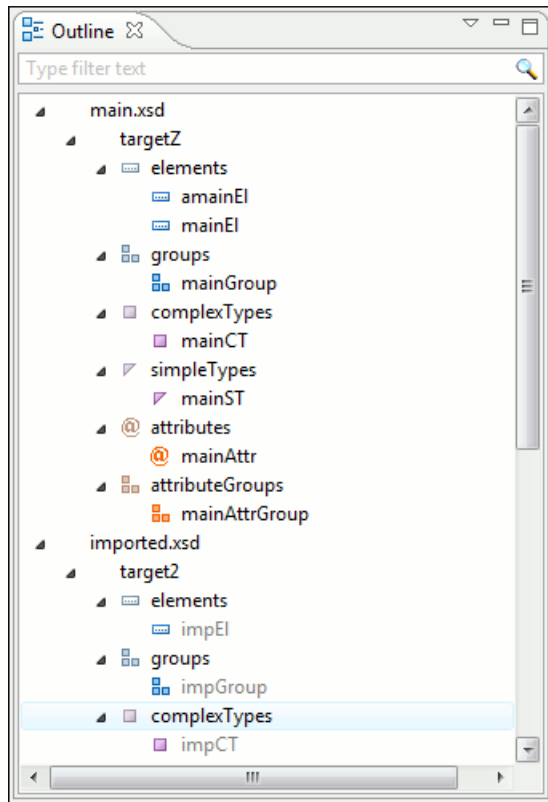


Figure 51: Outline View for XML Schema

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.

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.

Related information

[Searching and Refactoring Actions in XML Schemas](#) on page 313

[Component Dependencies View for XML Schema](#) on page 311

[XML Schema Resource Hierarchy / Dependencies View](#) on page 309

[Generating Sample XML Files](#) on page 315

[Editing Relax NG Schema in the Master Files Context](#) on page 358

XML Schema Attributes View

The **Attributes** view for XML Schemas presents the properties for the selected component in the schema diagram. By default, it is displayed on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

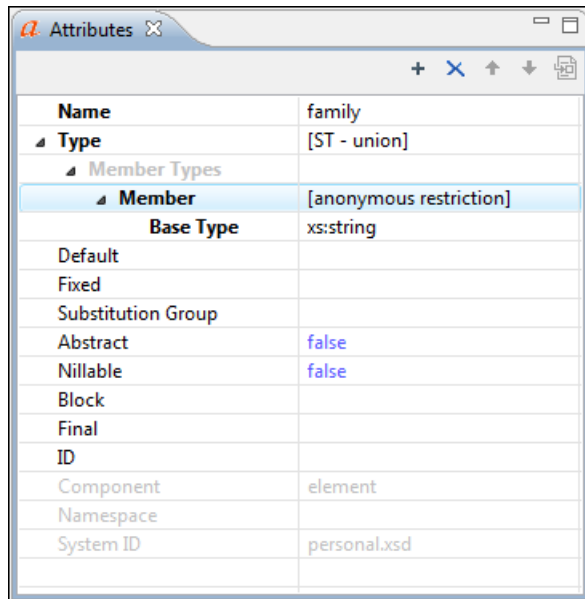


Figure 52: Attributes View

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 that contains at least one child is rendered with bold. Bold properties are properties with values set explicitly to them.

Properties for components that 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 Definition Ctrl (Meta on MAC OS) + Click

Shows the definition for the selected type.

Show Facets

Allows you to edit the facets for a simple type.

XML Schema Facets View

The **Facets** view for XML Schemas presents the facets for the selected component, if available. If the view is not displayed, it can be opened from the **Window > Show View** menu.

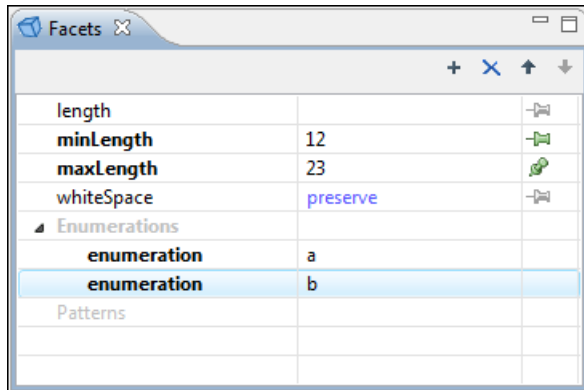


Figure 53: Facets View

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 that 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.

XML Schema 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](#). If the view is not displayed, it can be opened from the **Window > Show View** menu.

Figure 54: Palette View

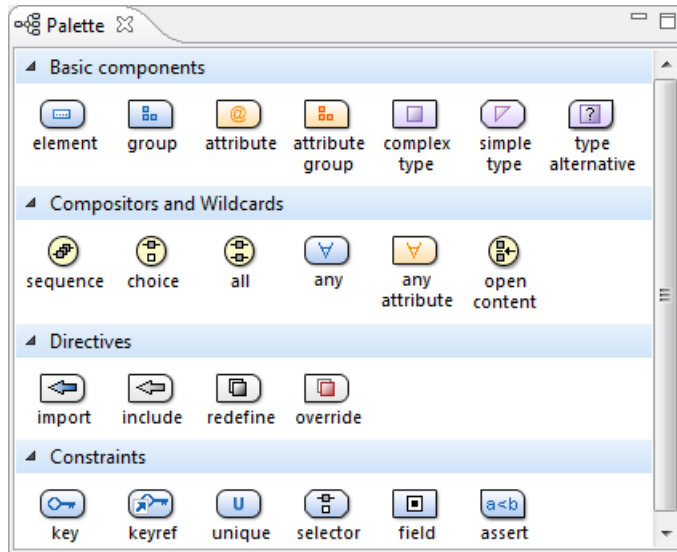


Figure 55: Palette View

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 https://www.oxygenxml.com/demo/Schema_Palette.html and https://www.oxygenxml.com/demo/Developing_XML_Schemas.html respectively.

Chapter

7

Editing Documents

Topics:

- [Working with Unicode](#)
- [Creating and Working with Documents](#)
- [Using Projects to Group Documents](#)
- [Editing XML Documents](#)
- [Editing XSLT Stylesheets](#)
- [Editing XML Schemas](#)
- [Editing XQuery Documents](#)
- [Editing WSDL Documents](#)
- [Editing CSS Stylesheets](#)
- [Editing Relax NG Schemas](#)
- [Editing NVDL Schemas](#)
- [Editing JSON Documents](#)
- [Editing StratML Documents](#)
- [Editing XLIFF Documents](#)
- [Editing JavaScript Documents](#)
- [Editing XProc Scripts](#)
- [Editing Schematron Schemas](#)
- [Editing Schematron Quick Fixes](#)
- [Editing XHTML Documents](#)
- [Editing Markdown Documents](#)
- [Spell Checking](#)
- [Handling Read-Only Files](#)
- [XML Digital Signatures](#)

This chapter includes a large amount of information about editing numerous types of documents, how to create and work with documents, working with projects, and various editing features that are provided in Oxygen XML Developer plugin. Each type of document has unique features and options when editing them in Oxygen XML Developer plugin, while some editing features are available for all document types. This chapter also includes information about some of the special tools that are included in Oxygen XML Developer plugin, such as the file and directory comparison tools.

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



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



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

Opening and Saving Documents with Unsupported Characters

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

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

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

When a document with a UTF-16 encoding is edited and saved in Oxygen XML Developer plugin, the saved document has a byte order mark (BOM) that 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 >  Insert from Character Map** action.

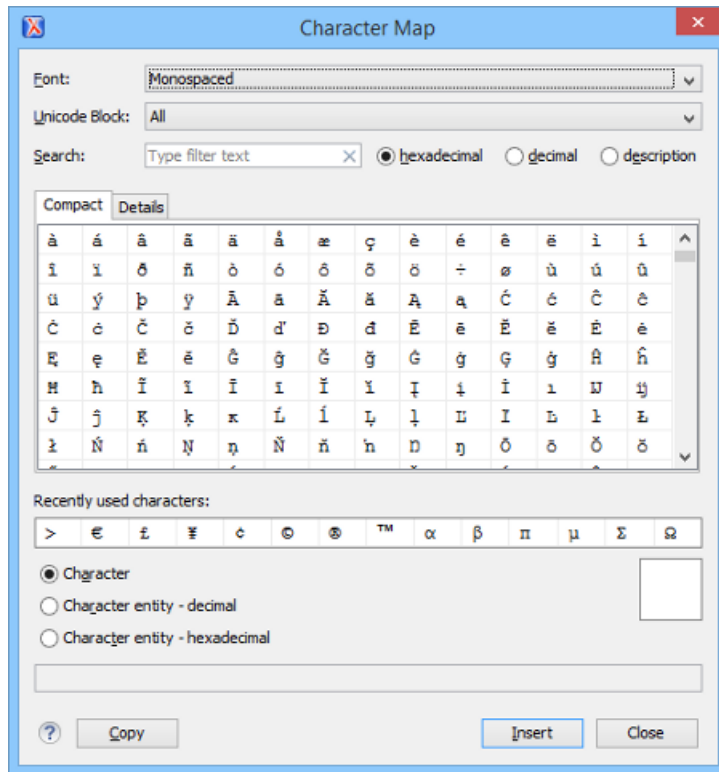


Figure 56: Character Map Dialog Box

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 **description** opens the [Details tab](#). If you enter a character description in the **Search** field, **description** 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 your document. 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 your document next to the selections in this section. 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 Developer 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 Developer 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 Developer 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 Developer plugin. To allow Oxygen XML Developer 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 Developer plugin installation, use the following procedure:

1. Start Windows Explorer and browse to the `[OXYGEN_INSTALL_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 Developer plugin for the changes to take full effect.

Result: Whenever Oxygen XML Developer 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 Developer 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 Developer plugin for the font changes to take full effect.

Creating and Working with Documents

Oxygen XML Developer 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 and Templates

Oxygen XML Developer 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.

New Document Wizard

Oxygen XML Developer plugin supports a wide range of document types. The **New Document** wizard presents the default associations between a file extension and the type of editor that opens the file. The **New Document** wizard only creates a skeleton document. It may contain a root element, the document prolog, and possibly other child elements depending on options that are specific for each schema type.

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

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

New Document Wizard

The **New Document** wizard allows you to create various types of documents and provides some options that help you to configure the new document. To use this wizard to create a new document in Oxygen XML Developer plugin, follow these steps:

1. Click the  **New** button on the toolbar or select **File > New > Other > Oxygen XML Developer plugin**.



Tip: You can also select *New from Templates* to create a document based upon predefined templates or custom templates.

Result: The **New Document** wizard is displayed with all the supported document types.

2. Select the type of document that you want to create.
3. Click the **Next** button.

Result: The next wizard page allows you to select a path where you want to store the new file and for some document types it includes some customization options. If you selected  **XML File** or  **XML Schema (XSD) File** for the type of document, you need to select the storage path and click **Next** again to reach customization options.

4. After configuring the options in this wizard, click **Finish** to create the file. If the **Open file for editing when done** option is selected, the new file will be opened in the appropriate editor.

XML File Type

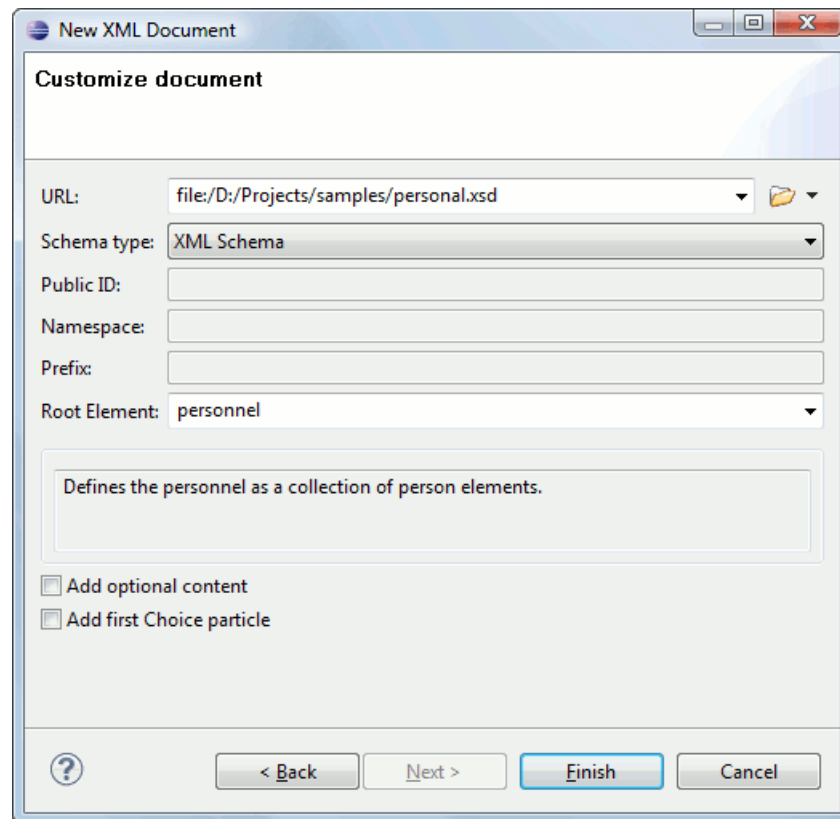


Figure 57: New XML File Configuration Options

If you selected  **XML File** for the type of document you want to create, the wizard will include the following options:

- **URL** - Specifies the path to the schema file. When you select a file, Oxygen XML Developer plugin analyzes its content and tries to fill in 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 pane** - A small description of the selected document root.
- **Add Optional Content** - If you select this option, the elements and attributes defined in the XML Schema as optional are generated in the skeleton XML document.
- **Add First Choice Particle** - If you select this option, Oxygen XML Developer plugin generates the first element of an `xs:choice` schema element in the skeleton XML document. Oxygen XML Developer plugin creates this document in a new editor panel when you click **Finish**.

XSL Stylesheet File Type

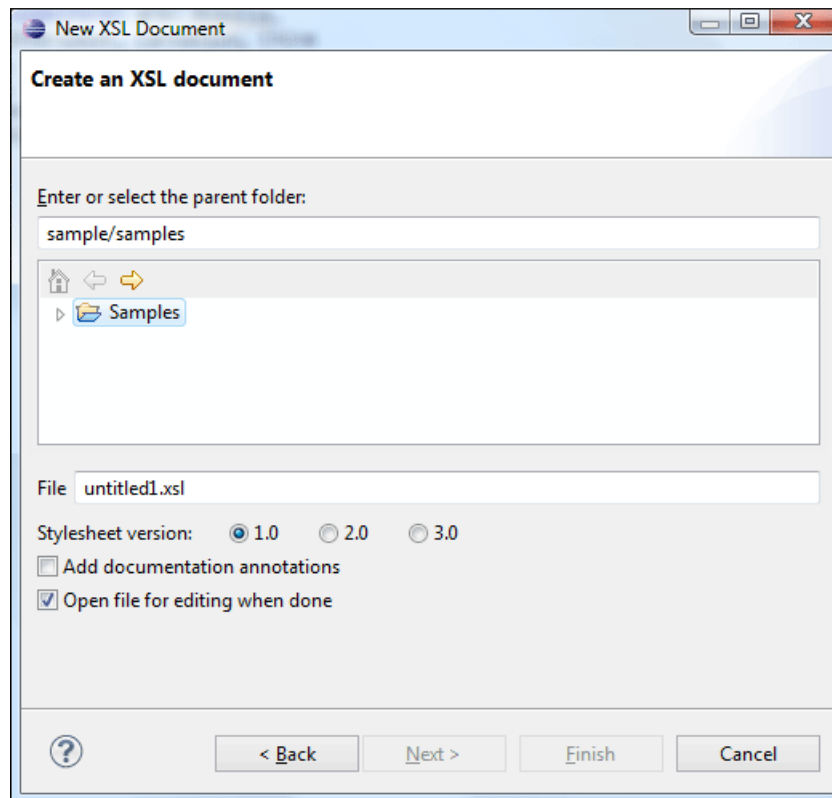


Figure 58: New XSL Document Configuration Options

If you selected  **Stylesheet (XSL) File** for the type of file you want to create, the wizard will include the following options:

- **Stylesheet version** - Allows you to select the Stylesheet version number. You can select from: 1.0, 2.0, and 3.0.
- **Add documentation annotations** - Enable this option to generate the stylesheet annotation documentation.

XML Schema (XSD) File Type

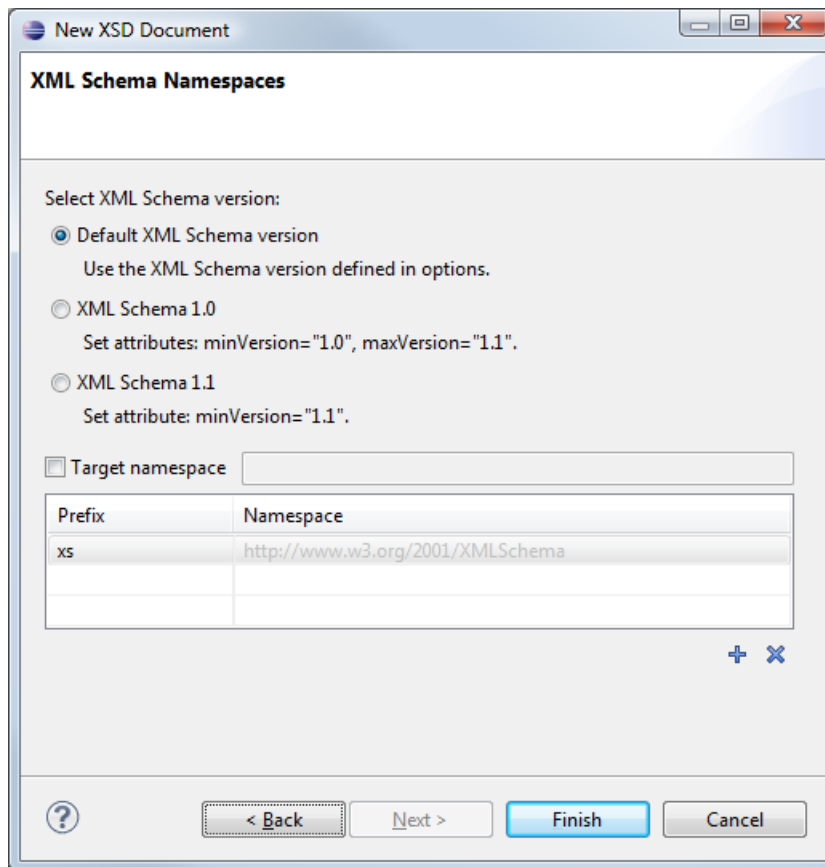


Figure 59: New XML Schema Configuration Options

If you selected  **XML Schema (XSD) File** for the type of file you want to create, the wizard will include the following options:

- **Default XML Schema version** - Select this option to use 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** - Allows you to specify the schema target namespace.
- **Namespace prefix declaration table** - This table contains namespace prefix declarations. Table information can be managed using 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](#).

Schematron File Type

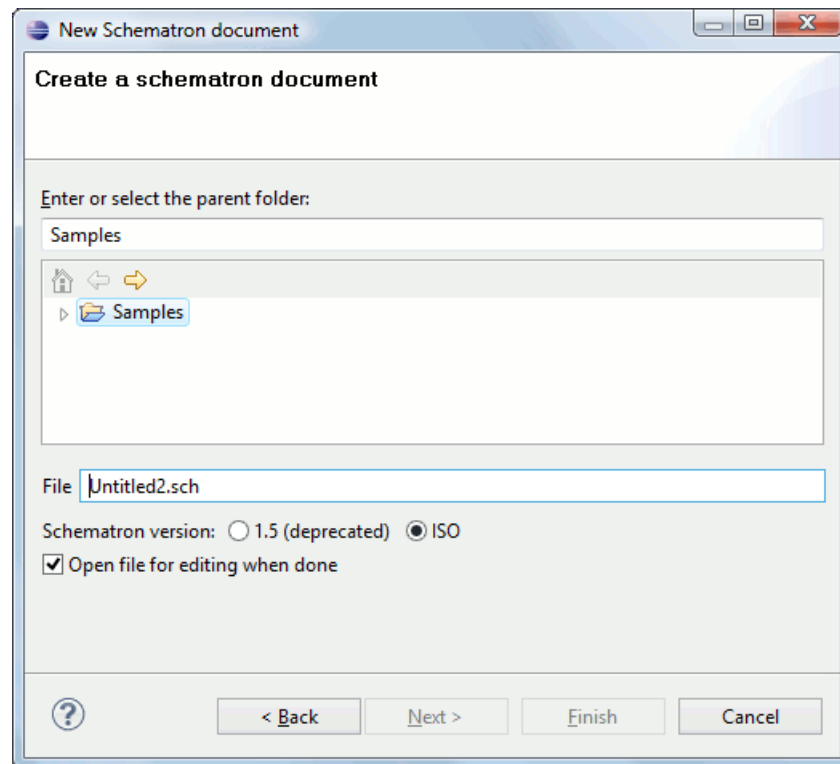


Figure 60: New Schematron Configuration Options

If you selected  **Schematron File** for the type of file you want to create and selected the **Customize** option, the configuration dialog box will include the following option:

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



Note: Starting with version 16.0 of Oxygen XML Developer plugin, the support for Schematron 1.5 is deprecated. It is recommended to use ISO Schematron instead.

Creating New Documents Based on Templates

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

The list of templates presented in the wizard includes:

- **User-defined template directory** - You can add your own custom templates by [creating template files](#) in a directory and then add that directory to the list of template directories that Oxygen XML Developer plugin uses in the [Document Templates preferences page](#). This user-defined directory will appear in the **New from templates** wizard.
- **Global templates** - Contains a list of predefined templates as well as any user-defined custom templates that are saved in the `templates` directory of the Oxygen XML Developer plugin installation folder (`[OXYGEN_INSTALL_DIR]/templates`).
- **Framework templates** - Contains the list of templates defined in the [Document Type configuration dialog box \(Templates tab\)](#) for each framework.

New from Templates Wizard

The **New from Templates** wizard allows you to create various types of documents based upon predefined or custom document templates. To use this wizard to create a new document in Oxygen XML Developer plugin, follow these steps:

1. Click the  **New** button on the toolbar and select **New from Templates** (or select **File > New > Other > Oxygen XML Developer plugin > New From Templates**).

Result: The **New from Templates** wizard is displayed that allows you to select various types of document templates.

2. Select the type of document that you want to create and click **Next**.
3. Choose the storage path and a file name for the new document.
4. Click the **Finish** button.

Result: The new file is created and if the **Open file for editing when done** option is selected, the new file will be opened in the appropriate editor.

Creating New Document Templates

Oxygen XML Developer plugin allows you to create your own custom document templates and they will appear in the **Global templates** folder (or another specified folder) within the *New from templates wizard*.

Creating a New Document Template

To create your own custom document template and have it appear in the new file wizard, follow these steps:

1. Create a new file (whatever type of document you need) and customize it to become a starting point for creating new files of this type.



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

2. Save the new file template in one of the following locations:
 - The `templates` directory of the Oxygen XML Developer plugin installation directory (`[OXYGEN_INSTALL_DIR]/templates`). File templates saved in this directory will appear in the **Global templates** category in the *New from templates wizard*.
 - You can also use any other directory of your choice, but you must add that directory to the list of templates in the *Document Templates preferences page*. This user-defined directory will appear in the *New from templates wizard* with the new file templates that you save in it.



Attention: The name that you use to save the template will be the name that appears in the new file wizard, including capitalization, space, and characters (for example, `My Custom Template1.xml` will appear in the new file wizard as **My Custom Template1**).

3. Open the new file wizard ( **New** toolbar button or **File > New > New from Templates**) and you should see your custom template in the appropriate folder. For DITA templates, they will also appear in the dialog box for creating new DITA topics, but if you create a corresponding properties file (see the procedure below), you need to set the `type` property to `dita`.

Related information

Customizing Document Templates on page 146

Customizing Document Templates

Oxygen XML Developer plugin allows you to customize certain aspects of predefined or custom document templates. For example, you can customize the icons or specify a prefix/suffix that will be used for the proposed file name in the *New from templates wizard*.

Customizing the Icons for a Document Template

If you want to customize the icons to be used for document templates, use a properties file to specify the icons using the following procedure:

1. Create a new properties file or edit an existing one.
 - If you create a new properties file, use the same name as the template file except with a `.properties` extension (for example, `MyTemplate.properties`). This properties file will specify the paths to the icons that will be used in the new file wizard. You can find some examples in the `templates` directory of the Oxygen XML Developer plugin installation directory to help you get started.

When defining the icons, the properties file should look like this:

```
type=general
smallIcon=../icons/Article_16.png
bigIcon=../icons/Article_48.png
```



Important: For DITA files, the `type` property needs to be set to `dita`. Otherwise, the template will not appear in the dialog box for creating new DITA topics. For all other types of files, set it to `general`. The icons specified in this properties file will only be used for the new file wizards and not in any other part of the interface.



Note: If you created a new template and chose to use a custom directory for the new template (in [step 2 of the new template procedure](#)), make sure the path to the icons is relative to that directory.

- If you edit an existing template, simply define the icon paths as specified [above](#).
2. Save the properties file in the same directory as the document template.
 3. Open the new file wizard (**File > New > New from Templates**) and you should see your custom icons next to the document template in the appropriate folder.

Add a Prefix or Suffix to File Names for a Document Template

You can use a properties file for each document template to add a prefix or suffix to the file name that is proposed in certain dialog boxes when you create a new file from that template. This applies to the following new document dialog boxes:

- The new document dialog box that appears when you select **New > New from Templates > [Template Name] > Next** from the contextual menu in the [Navigator view](#). The prefix or suffix is added to the name of the file in the **File** field.

To add a prefix or suffix to the file names for a document template, follow these steps:

1. Create a new properties file or edit an existing one.
 - If you create a new properties file, use the same name as the template file except with a `.properties` extension (for example, `MyTemplate.properties`). This properties file will specify the prefix/suffix that will be used to propose the file name in the new file wizards.

When defining the prefix/suffix, the properties file should look something like this:

```
type=general
filenamePrefix=prod_
filenameSuffix=_test
```



Important: For DITA files, the `type` property needs to be set to `dita`. For all other types of files, set it to `general`.

- If you edit an existing template, simply define the prefix/suffix as specified [above](#).
2. Save the properties file in the same directory as the document template.
 3. Open the new document wizard ([using the methods described above](#)) and when you select the appropriate template, you should see your prefix or suffix in the file name that is proposed in that dialog box.

Related information

[Creating New Document Templates](#) on page 146

Opening Documents

To open a document in Oxygen XML Developer plugin, do one of the following:

- Go to **File > Open File** to display the **Open File** dialog box.
- Go to **File > Open URL** to display a 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 the drop-down action list:
 -  **Browse for local file** - Opens a local file browser dialog box, allowing you to select a local DITA map.
 -  **Browse for remote file** - Displays 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 that allows you to search for a DITA map.
 - Select the **Open** or **Open with** action from the contextual menu of the [Navigator view](#).

Opening the Current Document in System Application

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

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

Oxygen XML Developer plugin supports editing remote files, using the FTP, SFTP, and WebDAV 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](#)

A WebDAV resource can be locked when it is opened in Oxygen XML Developer plugin by checking the [Lock WebDAV files on open option](#) to prevent other users to modify it concurrently on the server. If a user tries to edit a locked file, Oxygen XML Developer plugin displays an error message that contains the lock owner's name. The lock is released automatically when the editor for that resource is closed in Oxygen XML Developer plugin.

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

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

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

Open URL

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.

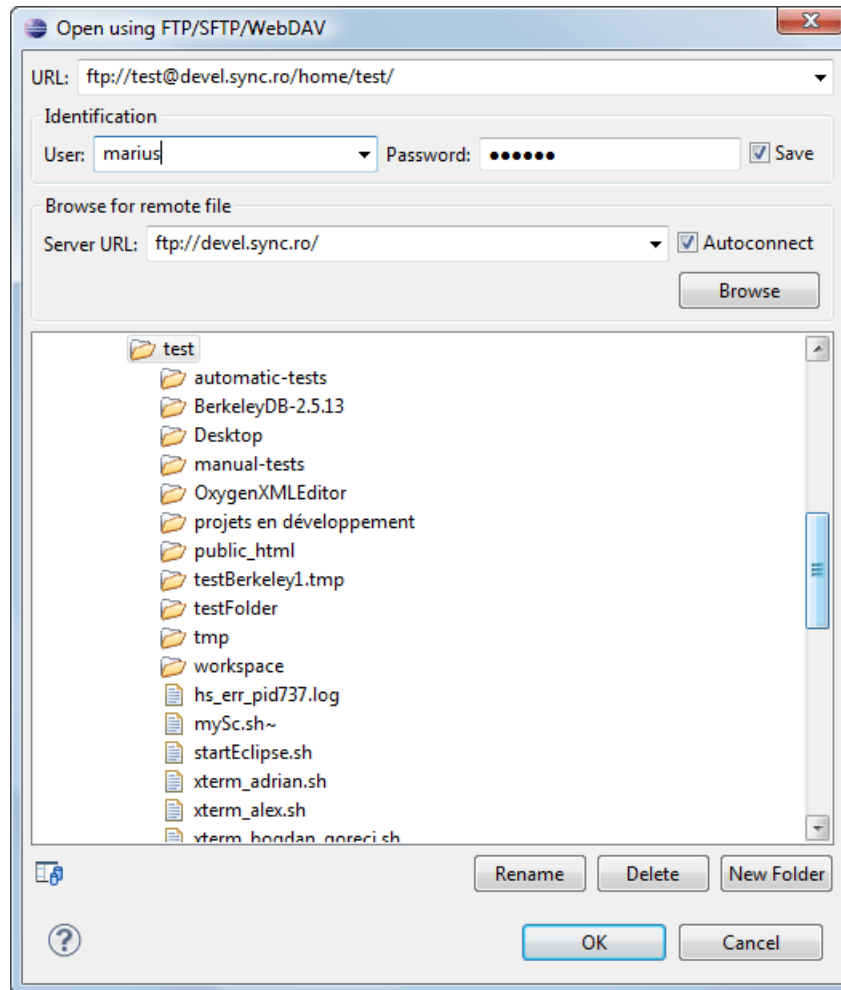


Figure 61: Open URL Dialog Box

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

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** checkbox 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 non-standard 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 a non-secure network, Oxygen XML Developer plugin allows you to load and save the documents over the HTTPS protocol (if the server understands this protocol) so that any data exchange with the WebDAV server is encrypted.

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

Troubleshooting HTTPS

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

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



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

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.

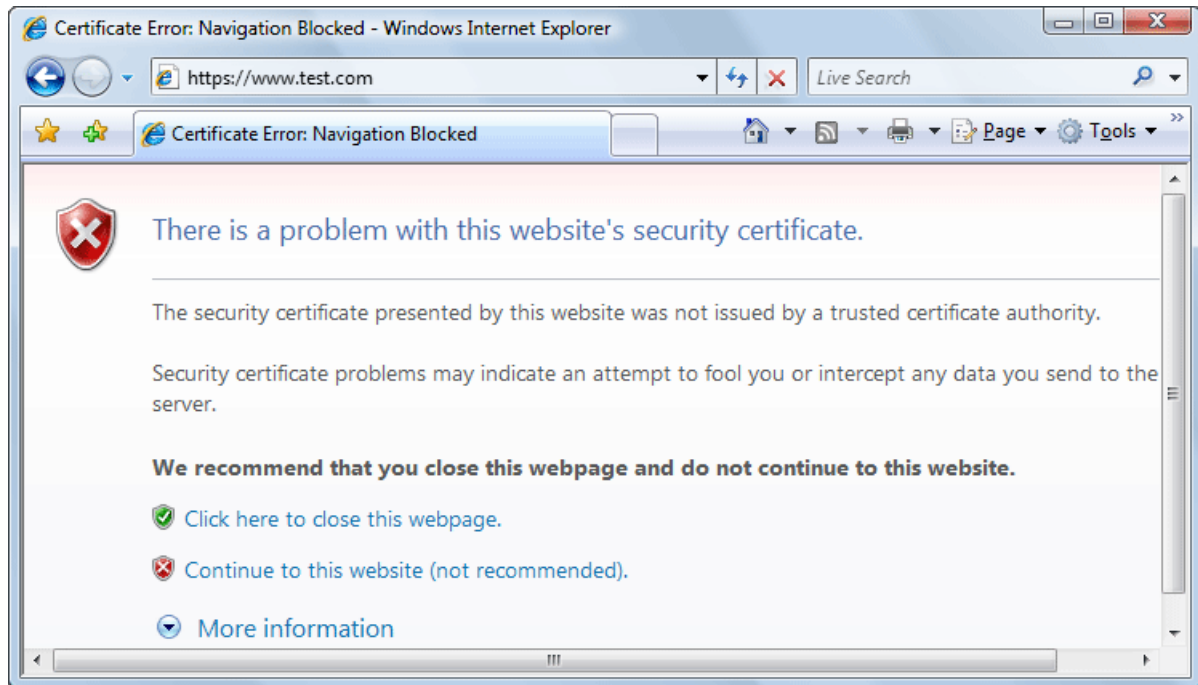


Figure 62: Security alert - untrusted certificate

- 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 Developer plugin.
 - a) Open a text-mode console with administrative rights.
If Oxygen XML Developer plugin has been installed in a user's home directory and includes a bundled JRE, administrative rights are not required. In all other cases administrative rights will be required.
 - b) Go to the `lib/security` directory of the JRE running Oxygen XML Developer plugin. You find the home directory of the JRE in the `java.home` property that is displayed in the **About** dialog box (**Installation Details > Configuration**).
On Mac OS X systems, the `lib/security` directory is usually located in `/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home` directory.
On OS X, if you have installed a distribution of Oxygen XML Developer plugin that is not bundled with a JRE, a JRE from Apple is required. The Apple Java version 1.6 stores the certificates in `/System/Library/Java/Support/CoreDeploy.bundle/Contents/Home/lib/security/cacerts`

with a symbolic link pointing to it from
/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/security/cacerts.
On OS X, if you have installed a distribution of Oxygen XML Developer plugin that bundles the JRE from Oracle, the JRE uses the .install4j/jre.bundle/Contents/Home/jre/lib/security/cacerts path within its installation 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 someone changed the default password, then that person is the only one who can perform the import.



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, as 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 Developer plugin.

Related information

[HTTP\(S\)/WebDAV Preferences](#) on page 73

HTTP Authentication Schemes

Oxygen XML Developer 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 Developer 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 Developer 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 actions that are available in the contextual menu of the current editor tab (or from the **File** menu):

Close (**Ctrl + F4** (**Command + F4** on OS X))

Closes the currently selected editor.

Close Others

If multiple files are opened, this action is available to close all opened editors in the current group/stack of tabs except for the one you are currently viewing.

Close All (**Ctrl + Shift + F4** (**Command + Shift + F4 on OS X**))

If multiple files are opened, this action is available to close all opened editors in the current group/stack of tabs. If this action is selected from the **File** menu, it closes all opened editors in **all** groups/stacks of tabs.

Contextual Menu of the Current Editor Tab

A contextual menu is available when you right-click the current editor tab label.



The actions that are available depend on the context and the number of files that are opened. The menu includes the following actions:

Close (**Ctrl + F4** (**Command + F4 on OS X**))

Closes the currently selected editor.

Close Others

If multiple files are opened, this action is available to close all opened editors in the current group/stack of tabs except for the one you are currently viewing.

Close All (**Ctrl + Shift + F4** (**Command + Shift + F4 on OS X**))

If multiple files are opened, this action is available to close all opened editors.

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 Developer 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 more information on managing projects and their content, see [Navigator View](#) on page 113.

Related information

[Using Projects to Group Documents](#) on page 153

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.

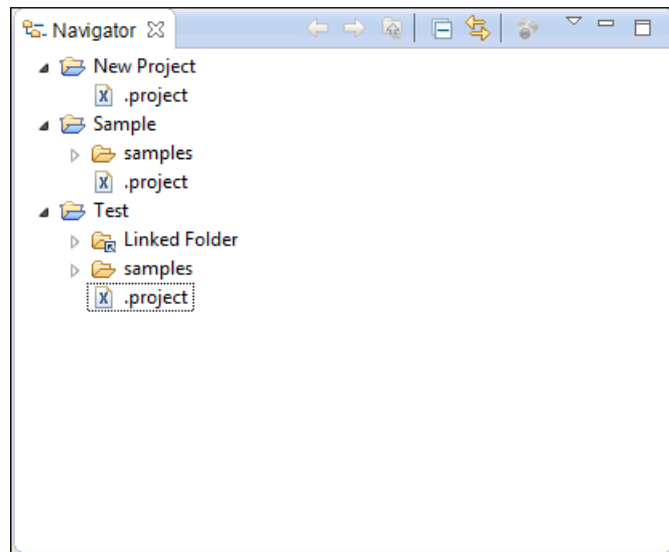


Figure 63: Navigator View

By default, the view is positioned on the left side of the Oxygen XML Developer plugin window, above the [Outline view](#). If the view has been closed, it can be reopened at any time from the **Window > Show View** menu.

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).

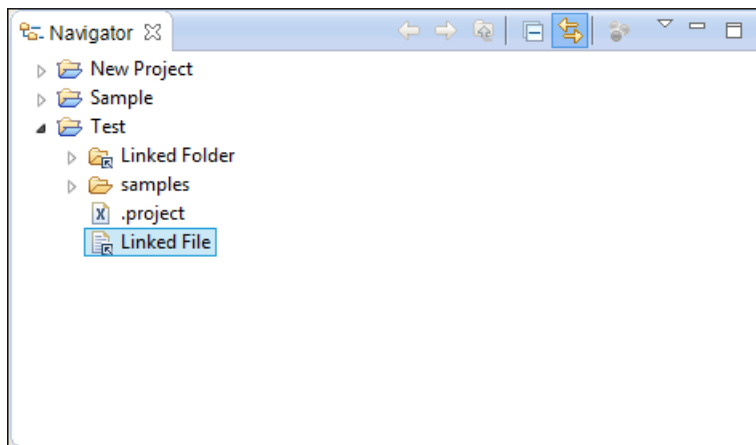


Figure 64: Navigator View with Examples of the Two Types of Resources

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 Developer plugin and the *Recycle Bin* is active) the file is moved to *Recycle Bin*.

Moving Files and Folders

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

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

Renaming Files and Folders

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

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

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

Locating and Opening Files

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

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

Saving the Project

The project file is automatically saved every time the content of the **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 Developer plugin in the **Package Explorer** view or in the **Navigator** view can be checked to be XML well-formed or validated against a schema (DTD, XML Schema, Relax NG, Schematron or NVDL) with one of the following contextual menu actions found in the **Validate** submenu:



Check Well-Formedness

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



Validate

Validates the selected file or files against their associated schema. EPUB files make an exception, because this action triggers a [Validate and Check for Completeness](#) operation.

Validate with Schema

Validates the selected file or files against a specified schema.



Configure Validation Scenario(s)

Allows you to configure and run a [validation scenario](#).



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 Developer plugin in the **Package Explorer** view or in the **Navigator** view can be transformed in one step with one of the following actions available from contextual menu in the **Transform** submenu:



Apply Transformation Scenario(s)

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



Configure Transformation Scenario(s)

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



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 Developer 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** submenu:

Rename resource

Allows you to change the name of a resource.

Move resource

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



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 files in the corresponding 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.

Related information

Working with EPUB on page 683

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 Developer plugin displays a warning dialog box.

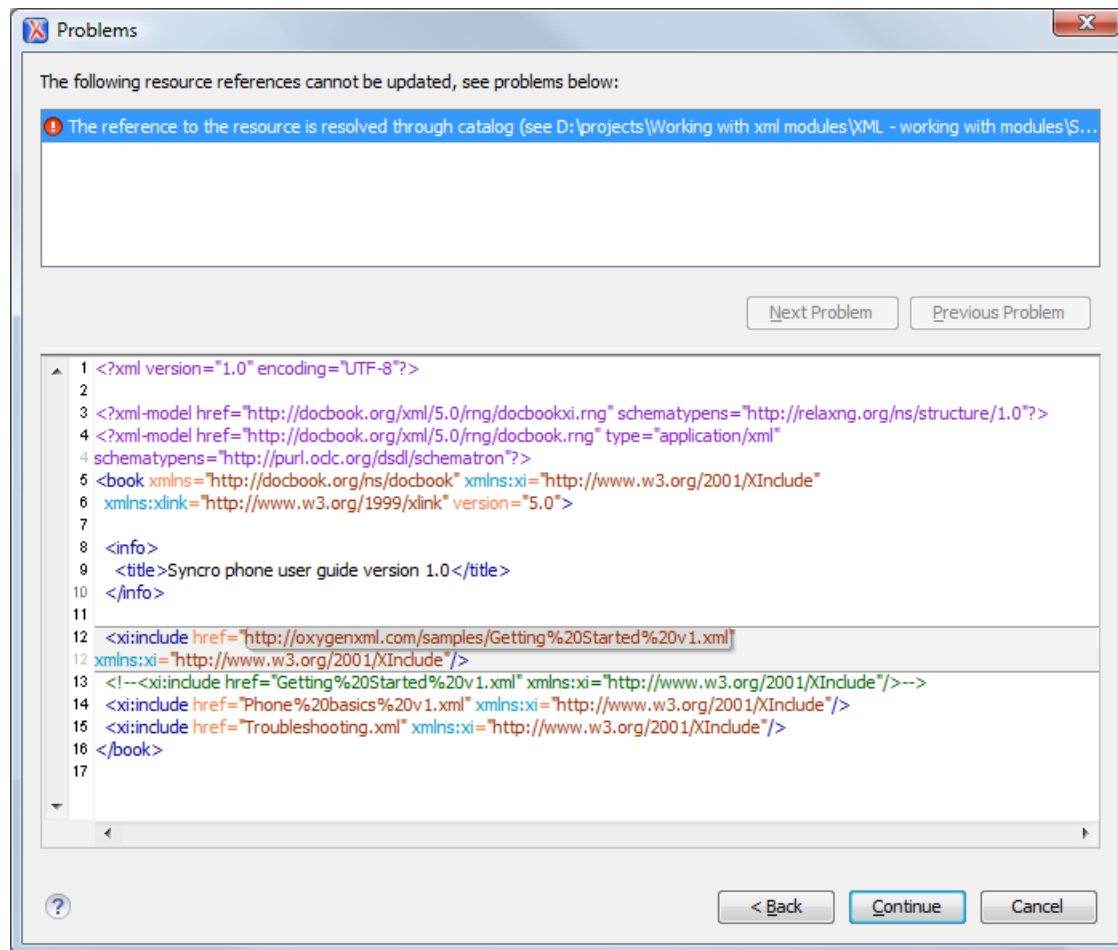


Figure 65: Problems Dialog Box

Master Files Support

This chapter details the *Master Files Support* available in Oxygen XML Developer plugin. A *Master File* typically refers to the root of an imported or included tree of modules and this support helps you simplify the configuration and development of XML projects.

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

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

Master Files Benefits

Using the *Master Files* support in Oxygen XML Developer plugin includes the following benefits:

- When the module is validated, Oxygen XML Developer plugin automatically identifies the *master files* that include the module and validates all of them.
- The **Content Completion Assistant** presents all the components that are collected from the *master files* for the modules they include.
- The **Outline** view displays all the components that are defined in the *master files* hierarchy.
- The *master files* that are defined for the current module determines the *scope of the search and refactoring actions*. Oxygen XML Developer plugin performs the search and refactoring actions in the context that the *master files* determine, thus improving the speed of execution.

Enabling the Master Files Support

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

Detecting Master Files

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

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

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

- **Possible master files** - The files presented on the first level in this category are not imported or 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 or import other files and are also not included or imported themselves. It is not necessary to set them as *master files*.

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

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 that 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 Files to the Master File Directory

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 various features in Oxygen XML Developer plugin for editing XML documents. It includes information about the user interface components and actions that are available in the various editing modes and numerous features to help you edit XML documents in any mode.

Related information

[Text Editing Mode](#) on page 112

[Grid Editing Mode](#) on page 126

Editing XML Documents in Text Mode

The Oxygen XML Developer plugin **Text** editing mode is designed to be a simple, yet powerful, XML source editor. You can use this mode to edit XML code, markup, and text and it provides support to help you transform, and debug XML-based documents. It is similar to other common text editors, but Oxygen XML Developer plugin also includes specialized editing actions, a powerful **Content Completion Assistant**, a helpful **Outline view**, and many other unique features.

To enter this mode, click the **Text** button at the bottom of the editing area.

Navigating the Document Content in Text Mode

Oxygen XML Developer plugin includes some useful features to help you navigate XML documents in **Text** mode.

Using the Keyboard

Oxygen XML Developer plugin allows you to quickly navigate through a document using the **Ctrl + CloseBracket (Command + CloseBracket on OS X)** key to go to the next XML node and **Ctrl + OpenBracket (Command + OpenBracket on OS X)** to go to the previous one.

To navigate one word forward or backwards, use **Ctrl + RightArrow (Command + RightArrow on OS X)**, and **Ctrl + LeftArrow (Command + LeftArrow 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.

Navigation Shortcuts

Oxygen XML Developer plugin includes some keyboard shortcuts to help you quickly navigate to a particular modification. They are also available as actions in the **Navigation** menu.

- **Ctrl+Q - Last Edit Location** - Moves the cursor to the last modification in any opened document.
- **Alt+LeftArrow (Command+OpenBracket on OS X) - Back** - Moves the cursor to the previous position.
- **Alt+RightArrow (Command+CloseBracket on OS X) - Forward** - Moves the cursor to the next position.

Navigating with the Outline View

Oxygen XML Developer plugin includes a very useful **Outline view** that displays a hierarchical tag overview of the currently edited XML Document.

You can use this view to quickly navigate through the current document by selecting nodes in the outline tree. It is synchronized with the editor area, so when you make a selection in the **Outline** view, the corresponding nodes are highlighted in the editor area.

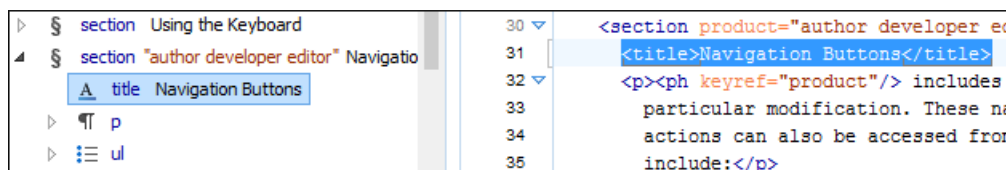


Figure 66: Outline View Navigation in Text Mode

Using the Breadcrumb to Navigate

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.

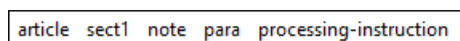


Figure 67: Breadcrumb in Text Mode

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 and navigates to it in the editor area.

Navigating with the Go To Dialog Box

In **Text** mode, you can navigate precisely to a location in the document you are editing by using the **Go To Line (Ctrl+L (Command+L on OS X))** action that is available in the **Navigation** menu.

Smart Editing in Text Mode

Oxygen XML Developer 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 Developer plugin will mirror-edit the name of the matching end tag. This feature can be controlled from the [Content Completion option page](#).
- **Auto-breaking the edited line** - The [Hard line wrap option](#) 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 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.

Shortcut Actions in Text Mode

Oxygen XML Developer plugin includes numerous shortcut actions to help you edit content in the **Text** editing mode.

Undo/Redo Actions

The typical undo and redo actions are available with shortcuts or in the **Edit** menu:

Undo (Ctrl + Z (Command + Z on OS X))

Reverses a maximum of 200 editing actions to return to the preceding state.



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

Redo (Ctrl + Y (Command + Shift + Z on OS X, Ctrl + Shift + Z on Linux/Unix))

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

Copy and Paste Actions

The typical copying and pasting actions are available with shortcuts or in the contextual menu (or the **Edit** menu):



Cut (Ctrl + X (Command + X on OS X))

Removes the current selected content from the document and places it in the clipboard.



Copy (Ctrl + C (Command + C on OS X))

Places a copy of the current selected content in the clipboard.

**Paste (Ctrl + V (Command + V on OS X))**

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

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

Selects the entire content of the current document.

Moving XML Nodes

You can use the following shortcuts to move XML elements or XSLT variables up or down in **Text** mode:

Ctrl + Alt + UpArrow (Command + Alt + UpArrow on OS X)

Moves the node up one line.

Ctrl + Alt + DownArrow (Command + Alt + DownArrow on OS X)

Moves the node down one line.



Note: The requirements for these node moving actions to work are as follows:

- The mechanism is designed to work without a selection. If you use these actions on a selection of content, it moves the entire selection. To make this mechanism work as intended, simply position the cursor somewhere on the line that you want to move.
- A start tag must be the first text occurrence on the line where the cursor is positioned.
- On the line where the element ends, only whitespaces are allowed after the end tag.

Miscellaneous Shortcut Actions in Text Mode

Oxygen XML Developer plugin also includes the following other miscellaneous shortcut actions in **Text** mode:

Ctrl + Delete (Command + Delete on OS X)

Deletes the next word.

Ctrl + Backspace (Command + Backspace on OS X)

Deletes the previous word.

Ctrl + W (Command + W on OS X)

Cuts the previous word.

Ctrl + K (Command + K on OS X)

Cuts to end of line.

Ctrl + Single-Click (Command + Single-Click on OS X)

Use this shortcut to open any of the following:

- Any absolute URL (URLs that have a protocol), regardless of their location in the document.
- URI attributes such as: *schemaLocation*, *noNamespaceSchemaLocation*, *href* and others.
- Processing instructions used for associating resources, xml-models, xml-stylesheets.

Ctrl + Shift + Y (Command + Shift + Y on OS X) (Document > Edit > Toggle Line Wrap)

Enables or disables line wrapping. When enabled, if text exceeds the width of the displayed editor, content is wrapped so that you do not have to scroll horizontally.

Editing XML Markup in Text Mode

Oxygen XML Developer plugin includes some useful actions that allow you to easily edit XML markup in **Text** mode. These actions are available in the **Refactoring** submenu of the contextual menu, and many of the actions can also be done with simple keyboard shortcuts.

Using the Breadcrumb

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 insert and edit specific elements in the document structure.

Figure 68: Breadcrumb in Text Mode

The last element listed in the *breadcrumb* is the element at the current cursor position. Clicking an element in 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.

Move Nodes

You can easily move XML nodes in the current document by using the following shortcut keys:

Alt + UpArrow

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

Alt + DownArrow

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

Rename Elements

You can rename elements by using the following actions in the **Refactoring** submenu of the contextual menu:

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 (Command + Shift + P on OS X)**)

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. *For example*, to change the `xmlns:p1="ns1"` association in the current element to `xmlns:p5="ns1"`, if the `xmlns:p1="ns1"` association is applied on the parent element, then Oxygen XML Developer plugin will introduce `xmlns:p5="ns1"` as a new declaration in the current element and will change the prefix from `p1` to `p5`. If `p5` is already associated with another namespace in the current element, then the conflict will be displayed in a dialog box. By pressing **OK**, the prefix is modified from `p1` to `p5` without inserting a new declaration.

- If you select the **Rename current prefix in all document** option, the application will apply the change on the entire document.
- To also apply the action inside attribute values, check the **Rename also attribute values that start with the same prefix** checkbox.

Surround Content with Tags (Wrap)

You can surround a selection of content with tags (*wrap* the content) by using the following action in the **Refactoring** submenu of the contextual menu:

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 *Position cursor between tags option* is enabled in the **Content Completion** preferences page, the cursor is placed between the start and end tag.
- If the *Position cursor 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 '[tag]' (Alt + Shift + ForwardSlash)

Surround the selected content with the last tag used.

Surround with <![CDATA]]> (Alt + Shift + C (Command + Alt + C on OS X))

Surround the selected content with a *CDATA* tag so that the parser will interpret it as textual data rather than markup.

Unwrap the Content of Elements

You can unwrap the content of an element by using the following action in the **Refactoring** submenu of the contextual menu:

Delete element tags (Alt + Shift + Comma)

Deletes the start and end tag of the current element.

Join or Split Elements

You can join or split elements in the current document by using the following actions in the **Refactoring** submenu of the contextual menu:

Join elements (Alt + Shift + F (Command + 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.

Split element

Split the element from the cursor position into two identical elements. The cursor must be inside the element.

Other Refactoring Actions

You can also manage the structure of the markup by using the other specific XML refactoring actions that are available in the **Refactoring** submenu of the contextual menu:

Attributes submenu

Contains predefined XML refactoring operations that pertain to attributes with some of the information preconfigured based upon the current context.

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.

Comments submenu

Contains predefined XML refactoring operations that pertain to comments with some of the information preconfigured based upon the current context.

Delete comments

Allows you to delete comments found inside one or more elements.

DITA submenu

Contains predefined XML refactoring operations that pertain to DITA documents with some of the information preconfigured based upon the current context.

Change topic ID to file name

Changes the ID of a topic to be the same as its file name.

Convert simple tables to CALS tables

Converts all DITA simple tables to CALS tables in the specified scope.

Elements submenu

Contains predefined XML refactoring operations that pertain to elements with some of the information preconfigured based upon the current context.

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 with some of the information preconfigured based upon the current context.

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.

JATSKit submenu

Contains predefined XML refactoring operations that pertain to JATS documents with some of the information preconfigured based upon the current context.

Add BITS DOCTYPE - NLM/NCBI Book Interchange 2.0

Adds an NLM 'BITS' 2.0 DOCTYPE declaration.

Add Blue DOCTYPE - NISO JATS Publishing 1.1

Adds a JATS 'Blue' 1.1 DOCTYPE declaration.

Normalize IDs

Assigned IDs are normalized and IDs are assigned to some elements that are missing them.

Related information

[Refactoring XML Documents](#) on page 228

[Contextual Menu Actions in Text Mode](#) on page 183

Folding XML Elements in Text Mode

When working with a large document, the **folding support** in Oxygen XML Developer plugin can be used to collapse some element content leaving only those that you need to edit in focus. Expanding and collapsing works on individual elements. Expanding an element leaves the child elements unchanged.

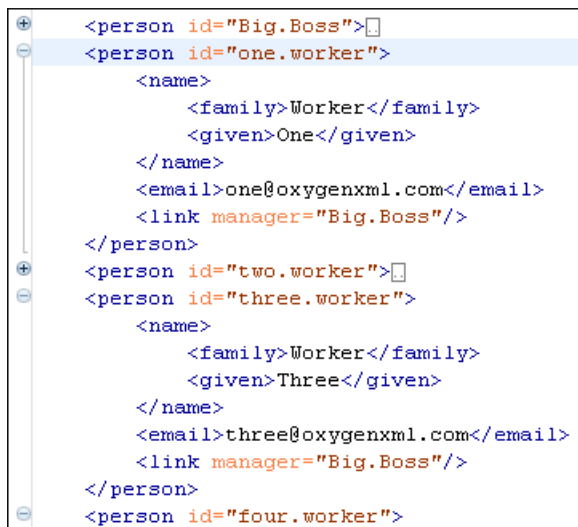


Figure 69: Folding of XML Elements in Text Mode

Folding Actions in Text Mode

Element folds are marked with a small icon ( / ) in the left stripe. To toggle the fold, simply click the icon. Also, if you right-click the icon, the following actions are available in the **Folding** sub-menu:

Toggle Fold

Toggles the state of the current fold.

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

Folds all the elements except the current element.

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

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

Expand Child Folds (**Ctrl + NumPad+ (Command + NumPad+ on OS X)**)

Unfolds all child elements of the currently selected element.

Expand All (**Ctrl + NumPad* (Command + NumPad* on OS X)**)

Unfolds all elements in the current document.

To watch our video demonstration about the folding support in Oxygen XML Developer plugin, go to

<https://www.oxygenxml.com/demo/FoldingSupport.html>.

Drag and Drop in Text Mode

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

You can also copy content from other applications and paste it into the document.

Selecting Content in Text Mode

Oxygen XML Developer plugin includes a variety of keyboard shortcuts that allow you to select content in **Text** mode. These include numerous standard continuous selection possibilities that are common to many text editors.

Standard Continuous Selection Shortcuts

Ctrl + A (Meta + A on Mac OS X)

Selects all content in the document.

Shift + Left/Right Arrow Keys

Begins a continuous selection at the cursor position and extends it one character at a time in the direction that you press the arrow keys.

Shift + Up/Down Arrow Keys

Begins a continuous selection at the cursor position and extends it one line at a time in the direction that you press the arrow keys.

Ctrl + Shift + Left/Right Arrow Keys (Meta + Shift + Left/Right Arrow Keys on Mac OS X)

Begins a continuous selection at the cursor position and extends it one word at a time in the direction that you press the arrow keys.

Shift + Home

Begins a continuous selection at the cursor position and extends it to the beginning of the current line (on Mac OS X, it extends to the beginning of the document).

Shift + End

Begins a continuous selection at the cursor position and extends it to the end of the current line (on Mac OS X, it extends to the end of the document).

Ctrl + Shift + Home

Begins a continuous selection at the cursor position and extends it to the beginning of the document.

Ctrl + Shift + End

Begins a continuous selection at the cursor position and extends it to the end of the document.

Shift + PageUp

Begins a continuous selection at the cursor position and extends it up one screen page.

Shift + PageDown

Begins a continuous selection at the cursor position and extends it down one screen page.

Double-Click

Selects certain text, depending on the position of the click in the document. See [Smart Editing Double-Click](#) for the specifics.

Triple-Click

Selects entire regions of text, depending on the position of the click in the document. See the [Smart Editing Triple-Click](#) for the specifics.

Right-Click > Select > Element

Selects the entire element at the current cursor position.

Right-Click > Select > 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.

Right-Click > Select > Attributes

Selects all the attributes of the element at the current cursor position.

Right-Click > Select > Parent

Selects the entire parent element at the current cursor position.

Content Completion Assistant in Text Mode

Oxygen XML Developer plugin includes an intelligent **Content Completion Assistant** that offers rapid, in-line identification and insertion of structured language elements, attributes, and attribute values. Oxygen XML Developer plugin shows the available entries that are valid in the current editing context.

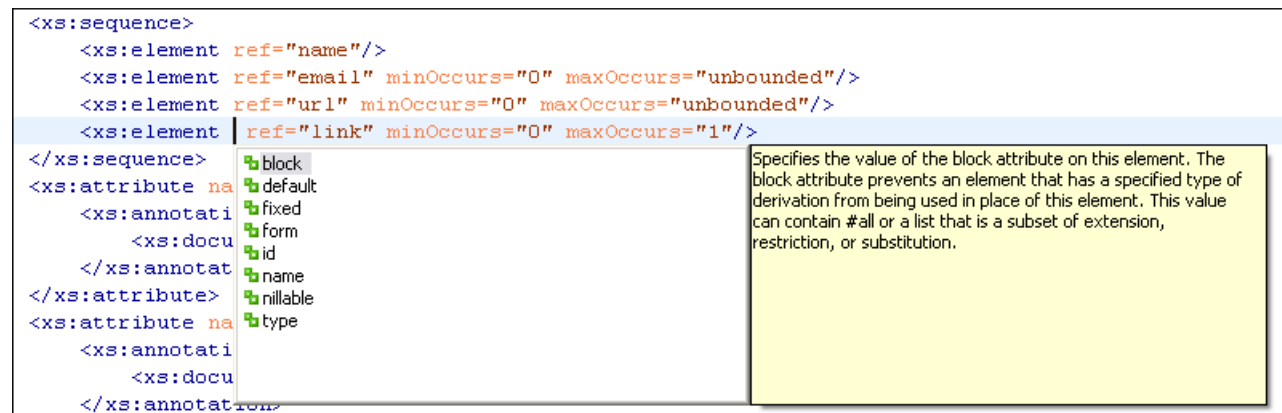


Figure 70: Content Completion Assistant

The **Content Completion Assistant** feature is schema-driven (XML Schema, DTD, and RELAX NG) and status information about the detected schema is logged in the [Status view](#).

The **Content Completion Assistant** is enabled by default. To disable it, [open the Preferences dialog box](#), go to **Editor > Content Completion**, and disable the [Enable content completion option](#).

Using the Content Completion Assistant in Text Mode

The feature is activated in **Text** mode in the following situations:

- After you press the $\leq$ character when inserting an element, it is automatically activated after a short delay. You can adjust the activation delay with the [Activation delay of the proposals window \(ms\) option](#) from the **Content Completion** preferences page.
- After typing a partial element or attribute name, you can activate it by pressing **Ctrl + Space (Command + Space on OS X)** or **Alt + ForwardSlash (Command + Alt + ForwardSlash on OS X)**. If there is only one valid proposal at the current location, it is inserted without displaying the list of proposals.

When active, the **Content Completion Assistant** displays a list of context-sensitive proposals valid at the current cursor position. You can navigate through the list of proposals by using the **Up** and **Down** 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 in **Text** mode, do one of the following:

- Press **Enter** or **Tab** to insert both the start and end tags and position the cursor inside the start tag in a position suitable for inserting attributes.
- Press **Ctrl + Enter (Command + Enter on OS X)** to insert both the start and end tags and positions the cursor between the tags in a position where 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](#) (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, XML Schema, or RELAX NG schema. To activate these features, select the [Add optional content](#) and [Add first Choice particle](#) options in the **Content Completion** preferences page.

After inserting an element, the cursor is positioned:

- Before the **>** character of the start tag, if the element allows attributes, 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.

Where the Content Completion Assistant is Displayed

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.

Types of Proposals Listed in the Content Completion Assistant

The following things are considered for determining the proposals that are listed in the content completion window:

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



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

- The number and type of elements displayed by the **Content Completion Assistant** is dependent on the cursor's current position in the structured document. The child elements displayed within a given element are defined by the structure of the specified DTD, XML Schema, Relax NG (full or compact syntax) schema, or NVDL schema.
- A schema may declare certain attributes as *ID* or *IDREF/IDREFS*. When the document is validated, Oxygen XML Developer plugin checks the uniqueness and correctness of the *ID* attributes. It also collects the attribute values declared in the document to prepare the list of proposals. This is available for documents that use DTD, XML Schema, and Relax NG schema.
- 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 an enumeration of tokens as the type. Also, if a default value or 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 to be Used for Content Completion in Text Mode

The list of proposals in the **Content Completion Assistant** depend on the associated schemas. The DTD, XML Schema, Relax NG, or NVDL schema used to populate the **Content Completion Assistant** is specified in the following methods, in the order of their precedence:

- The *schema specified explicitly in the document*. In this case, Oxygen XML Developer plugin reads the beginning of the document and resolves the location of the DTD, XML Schema, Relax NG schema, or NVDL schema.
- The *default schema declared* in the **Document Type configuration dialog box** that matches the edited document type.
- For XSLT stylesheets, the schema specified in the Oxygen XML Developer plugin **XSL preferences page**. Oxygen XML Developer plugin will read the content completion settings when the prolog fails to provide or resolve the location of a DTD, XML Schema, Relax NG, or NVDL schema.
- For XML Schemas, the schema specified in the Oxygen XML Developer plugin **XSD preferences page**. Oxygen XML Developer plugin will read the content completion settings and the specified schema will enhance the content completion inside the `xs:annotation/xs:appinfo` elements of the XML Schema.

Oxygen XML Developer plugin creates the content completion lists by analysing the detected schema and the current context (the position in the editor). If you change the schema, then the list of tags to be inserted is also updated.

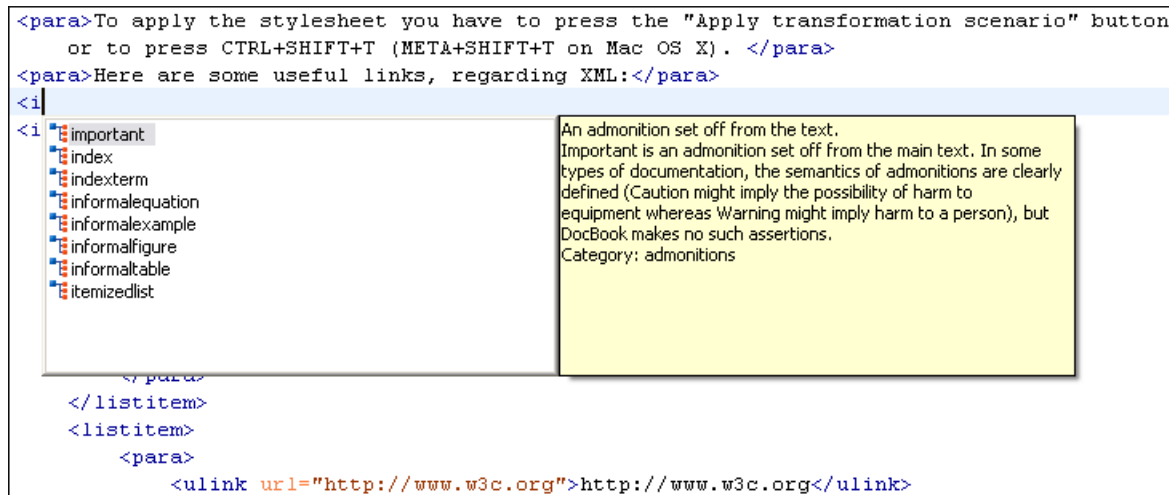


Figure 71: Example: Content Completion Driven by DocBook DTD

Schema Annotations in Text Mode

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

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

The schema annotations support is available if the schema type is one of the following:

- XML Schema
- Relax NG
- NVDL schema
- DTD

This feature is enabled by default, but you can disable it by deselecting the **Show annotations in Content Completion Assistant** option in the **Annotations** preferences page.

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 (such as `div`, `body`, `p`, `br`, `table`, `ul`, or `ol`).

The HTML rendering is controlled by the **Show annotations using HTML format, if possible** option in the **Annotations** preferences page. When this options is disabled, the annotations are converted and displayed as plain text and 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. For example, `p` begins a new paragraph, `br` breaks the current line, `ul` encloses a list of items, and `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>
```

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

Collecting Annotations from Relax NG Schemas

For Relax NG schema, element and attribute annotations 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 in the **Annotations** preferences page.

Collecting Annotation from DTDs

For DTD, Oxygen XML Developer plugin defines a custom mechanism for annotations using comments enabled from the [Prefer DTD comments that start with "doc:" as annotations](#) option in the **Annotations** preferences page. The 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.

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. By default, it is located on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

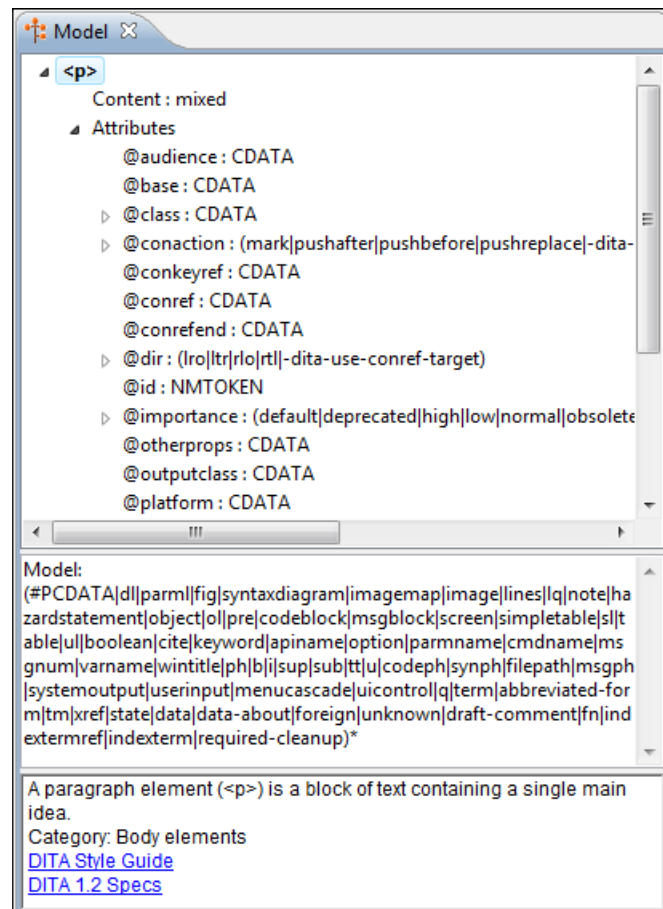


Figure 72: Model View

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.

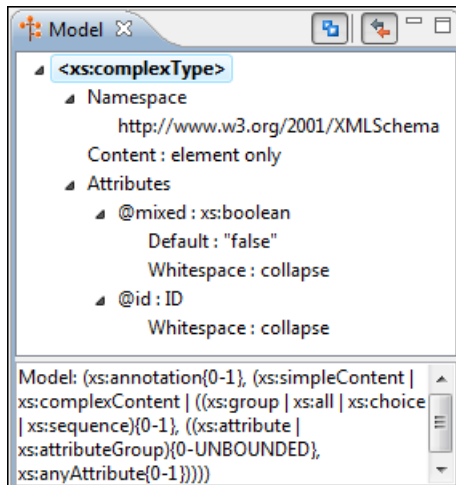


Figure 73: Element Structure Panel

Annotation Panel

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

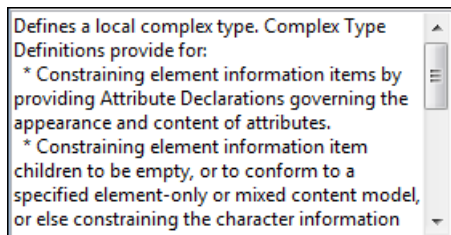


Figure 74: Annotation panel

Attributes View in Text Mode

The **Attributes** view presents all the attributes of the current element determined by the schema of the document. By default, it is located on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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 are rendered with a bold font, and their values with a plain font.
- 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.

To edit the value of the corresponding attribute, double-click a cell in the **Value** column. 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 either select the value from a list or manually enter it.

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.

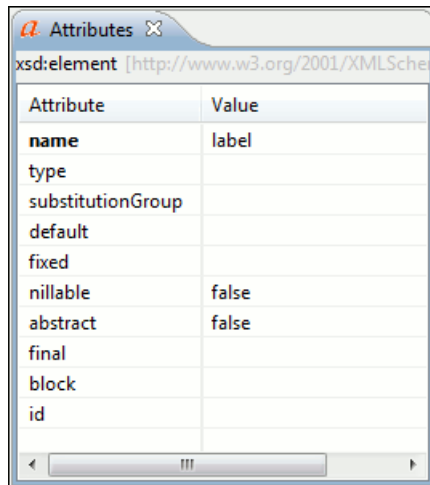


Figure 75: Attributes View

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.

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.

Elements View in Text Mode

The **Elements** view presents a list of all defined elements that are valid at the current cursor position according to the schema associated to the document. By default, it is located on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

Double-clicking any of the listed elements inserts that element into the edited document, at the current cursor position.

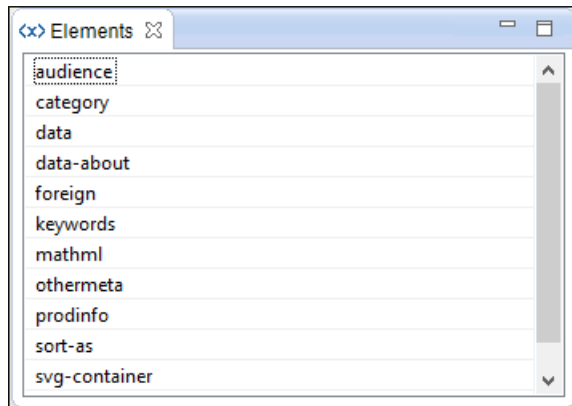


Figure 76: Elements View in Text Mode

Entities View

Entities provide abbreviated entries that can be used in XML files when there is a need of repeatedly inserting certain characters or large blocks of information. An *entity* is defined using the ENTITY statement either in the DOCTYPE declaration or in a DTD file associated with the current XML file.

There are three types of entities:

- *Built-in or Predefined* - Entities that are part of the predefined XML markup (<, >, &, ', ").
- *Internal* - Defined in the DOCTYPE declaration header of the current XML.
- *External* - Defined in an external DTD module included in the DTD referenced in the XML DOCTYPE declaration.



Note: If you want to add internal entities, you would need to switch to the Text editing mode and manually modify the DOCTYPE declaration. If you want to add external entities, you need to open the DTD module file and modify it directly.

The **Entities** view displays a list with all entities declared in the current document, as well as built-in ones. By default, it is located on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

Double-clicking one of the entities will insert it at the current cursor position in the XML document. You can also sort entities by name and value by clicking the column headers.

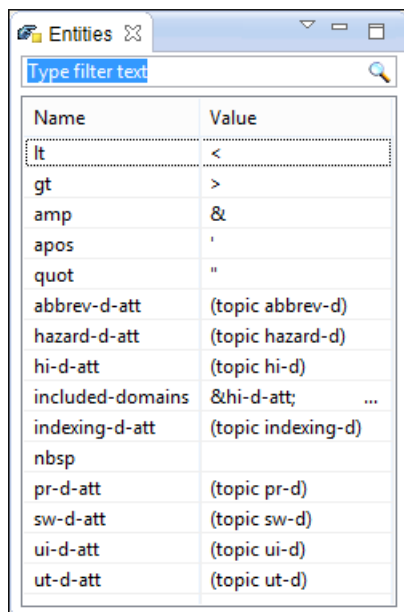


Figure 77: Entities View

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 Developer plugin includes a set of built-in code templates for CSS, Schematron, XSL, XQuery, and XML Schema document types. You can also *define your own code templates and share them with others*.

To get a complete list of available code templates, press **Ctrl + Shift + Space** 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 https://www.oxygenxml.com/demo/Code_Templates.html.

Outline View in Text Mode

The **Outline** view in **Text** mode displays a general tag overview of the currently edited XML Document. When you edit a document, the **Outline** view dynamically follows the changes that you make, displaying the node that you modify. This functionality gives you great insight on the location of your modifications in the current document. It also shows the correct hierarchical dependencies between elements. This makes it easy for you to be aware of the document structure and the way element tags are nested.

Outline View Features

The **Outline** view allows you to:

- Quickly navigate through the document by selecting nodes in the **Outline** tree.
- 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 editor area. It is synchronized with the editor area, so when you make a selection in the editor area, the corresponding nodes are highlighted in the **Outline** view, and vice versa.
- View document errors, as they are highlighted in the **Outline** view. A tooltip also provides more information about the nature of the error when you hover over the faulted element.

Outline View Interface

By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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.

Drop and Drop Actions in the 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 node, 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, a copy operation will be performed instead of a move.

The drag and drop actions in the **Outline** view can be *disabled and enabled from a Preferences page*.

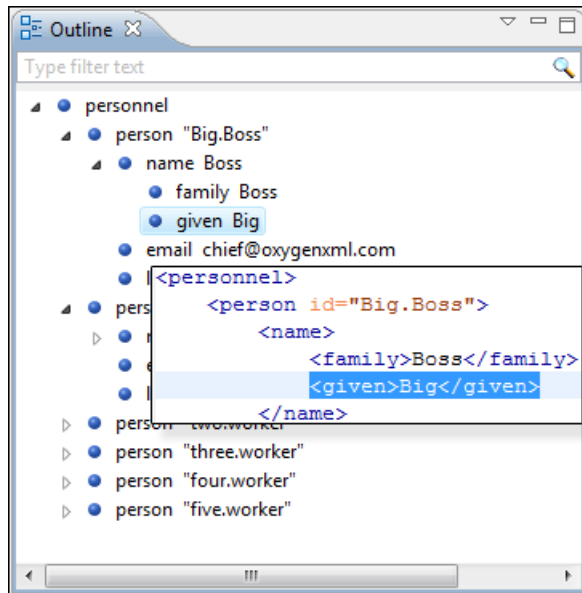


Figure 78: Outline View in Text Mode

Outline View Filters in Text Mode

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](#).

Outline View Contextual Menu Actions in Text Mode

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.

Quick Assist Support for IDs and IDREFS

The Quick Assist support is activated automatically when you place the cursor inside an ID or IDREF in **Text** mode. 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 and references of the ID located at the cursor position in the current document.

Formatting and Indenting XML Documents

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

- **Document > Source >  Format and Indent** - Formats and indents the whole document.
- **Document > Source >  Indent Selection** - Indents the current selection (but does not add line breaks). 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 that 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 Developer plugin formats and indents XML. Many of these settings have to do with how whitespace is handled.

Significant and insignificant whitespace in XML

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

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

```
<p>NO Free man shall be taken or imprisoned, or be stripped of his Freedom,  
or Liberties, or free Customs, or be outlawed, or exiled, or any otherwise  
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 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. The paragraph above could have been written on one line because the XML parser would see it as exactly the same paragraph since all multiple consecutive whitespaces will be replaced with a single whitespace. 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 Developer plugin determines when whitespace is significant

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

To determine what whitespace is significant, Oxygen XML Developer plugin assigns each element in the document to one of four categories:

Ignore space

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

Normalize space

In the normalize space category, a single whitespace character between character strings is considered significant and all other spaces are considered insignificant. Therefore, all consecutive whitespaces will be replaced with a single space. This generally applies to elements that contain text content only.

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 a <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. However, child elements may be in another category, and may be treated differently.

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

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

To assign elements to these categories, Oxygen XML Developer plugin consults information from the following sources:

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.

Schema aware formatting

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

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

Schema aware formatting can be turned on and off.

- To turn it on or off for the **Text** editing mode, *open the [Preferences dialog box](#)*, go to **Editor > Format > XML**, and enable/disable the *[Schema aware format and indent option](#)*.

Preserve space elements list

If an element is listed in the *[Preserve space tab of the Element Spacing 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 tab of the Element Spacing 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 tab of the Element Spacing list](#)* in the *[XML formatting preferences](#)*, it is promoted to the mixed content category.

Element content

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

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

Text node content

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

How Oxygen XML Developer plugin formats and indents XML

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

- *[Format preferences page](#)*
- *[XML Formatting preferences page](#)*
- *[Whitespaces preferences page](#)*

When Oxygen XML Developer plugin formats and indents XML

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

- In **Text** mode when you select one of the format and indent actions (**Document > Source > Format and Indent**, **Document > Source > Indent Selection**, or **Document > Source > Format and Indent Element**).
- When saving documents in **Design** mode.
- When switching from **Design** mode to another mode.
- When saving or switching to **Text** mode from **Grid** mode, if the *[Format and indent when passing from grid to text or on save option](#)* is selected in the **Grid** preferences page.

Setting an Indent Size to Zero

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



Note: Changing the indent size does not override the rules that Oxygen XML Developer plugin uses for handling whitespace when formatting and indenting XML documents. Therefore, changing the indent size will have no effect on elements that require whitespaces to be maintained.

There are two cases to consider.

Maintaining zero indent in documents with zero indent

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

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

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

Enforcing zero indent for all documents

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

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

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

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

Format and Indent (Pretty Print) Multiple Files

Oxygen XML Developer plugin provides support for formatting and indenting (*pretty 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.

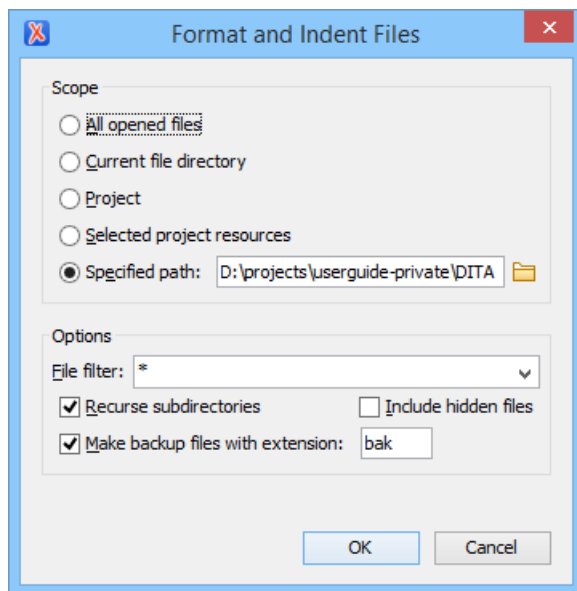


Figure 79: Format and Indent Files Dialog Box

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 Developer 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 Developer 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 multiple results, 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.

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**.

Contextual Menu Actions in Text Mode

When editing XML documents in **Text** mode, Oxygen XML Developer 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 (Command + 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 + CloseBracket (Command + CloseBracket on OS X))

Moves the cursor to the end of the next tag.

Go after Previous Tag (Ctrl + OpenBracket (Command + OpenBracket 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 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 (Command + 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 (Command + Shift + I on OS X))

Pretty prints the element that surrounds the current cursor position.

Convert Hexadecimal Sequence to Character (Ctrl + Shift + H (Command + Shift + H on OS X))

Converts a sequence of hexadecimal characters to the corresponding Unicode character. The action can be invoked if there is a selection containing a valid hexadecimal sequence or if the cursor is placed at the right side of a valid hexadecimal sequence. A valid hexadecimal sequence can be composed of 2 to 4 hexadecimal characters and may or may not be preceded by the 0x or 0X prefix. Examples of valid sequences: 0x0045, 0X0125, 1253, 265, 43.

Base64 Encode/Decode submenu

This submenu include the following actions for encoding or decoding *base64* schemes:

Import File to Encode and Insert

Encodes a file and then inserts the encoded content into the current document at the cursor position.

Decode Selection and Export to File

Decodes a selection of text from the current document and then exports (saves) the result to another file.

Encode Selection

Replaces a selection of text with the result of encoding that selection.

Decode Selection

Replaces a selection of text with the result of decoding that selection.

Modify All Matches

Use this option to modify (in-place) all the occurrences of the selected content (or the contiguous fragment where 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.

Base32 Encode/Decode submenu

This submenu include the following actions for encoding or decoding *base32* schemes:

Import File to Encode and Insert

Encodes a file and then inserts the encoded content into the current document at the cursor position.

Decode Selection and Export to File

Decodes a selection of text from the current document and then exports (saves) the result to another file.

Encode Selection

Replaces a selection of text with the result of encoding that selection.

Decode Selection

Replaces a selection of text with the result of decoding that selection.

Modify All Matches

Use this option to modify (in-place) all the occurrences of the selected content (or the contiguous fragment where 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.

Hex Encode/Decode submenu

This submenu include the following actions for encoding or decoding *hex* schemes:

Import File to Encode and Insert

Encodes a file and then inserts the encoded content into the current document at the cursor position.

Decode Selection and Export to File

Decodes a selection of text from the current document and then exports (saves) the result to another file.

Encode Selection

Replaces a selection of text with the result of encoding that selection.

Decode Selection

Replaces a selection of text with the result of decoding that selection.

Modify All Matches

Use this option to modify (in-place) all the occurrences of the selected content (or the contiguous fragment where 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.

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 where 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 (Command + Shift + P on OS X)**)

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. *For example*, to change the `xmlns:p1="ns1"` association in the current element to `xmlns:p5="ns1"`, if the `xmlns:p1="ns1"` association is applied on the parent element, then Oxygen XML Developer plugin will introduce `xmlns:p5="ns1"` as a new declaration in the current element and will change the prefix from `p1` to `p5`. If `p5` is already associated with another namespace in the current element, then the conflict will be displayed in a dialog box. By pressing **OK**, the prefix is modified from `p1` to `p5` without inserting a new declaration.
- If you select the **Rename current prefix in all document** option, the application will apply the change on the entire document.
- To also apply the action inside attribute values, check the **Rename also attribute values that start with the same prefix** checkbox.



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 *Position cursor between tags option* is enabled in the **Content Completion** preferences page, the cursor is placed between the start and end tag.
- If the *Position cursor 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 '[tag]' (**Alt + Shift + ForwardSlash**)

Surround the selected content with the last tag used.

Surround with <![CDATA]]> (**Alt + Shift + C (Command + Alt + C on OS X)**)

Surround the selected content with a *CDATA* tag so that the parser will interpret it as textual data rather than markup.



Delete element tags (**Alt + Shift + Comma**)

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 (Command + 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 with some of the information preconfigured based upon the current context.

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.

Comments submenu

Contains predefined XML refactoring operations that pertain to comments with some of the information preconfigured based upon the current context.

Delete comments

Allows you to delete comments found inside one or more elements.

DITA submenu

Contains predefined XML refactoring operations that pertain to DITA documents with some of the information preconfigured based upon the current context.

Change topic ID to file name

Changes the ID of a topic to be the same as its file name.

Convert simple tables to CALS tables

Converts all DITA simple tables to CALS tables in the specified scope.

Elements submenu

Contains predefined XML refactoring operations that pertain to elements with some of the information preconfigured based upon the current context.

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 with some of the information preconfigured based upon the current context.

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.

JATSKit submenu

Contains predefined XML refactoring operations that pertain to JATS documents with some of the information preconfigured based upon the current context.

Add BITS DOCTYPE - NLM/NCBI Book Interchange 2.0

Adds an NLM 'BITS' 2.0 DOCTYPE declaration.

Add Blue DOCTYPE - NISO JATS Publishing 1.1

Adds a JATS 'Blue' 1.1 DOCTYPE declaration.

Normalize IDs

Assigned IDs are normalized and IDs are assigned to some elements that are missing them.

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 and references of the ID in the current document.

Quick Assist (**Ctrl + 1** (**Command + 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.

Editing XML Documents in Grid Mode

The **Grid** mode in Oxygen XML Developer plugin displays the XML document as a structured grid of nested tables where the text content can be modified without directly interacting with the XML markup. This is helpful for non-technical users who want to edit text content without modifying the XML markup. You can easily expand or collapse elements within the table and the document structure can be changed with simple drag/drop or copy/paste operations. A useful **Content Completion Assistant** is also available in **Grid** mode.

To enter this mode, click the **Grid** button at the bottom of the editing area.

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

Editing Actions in Grid Mode

A variety of editing actions are available in **Grid** mode from the contextual menu, the **Document** menu, the toolbar, and with shortcut keys. This section explains some of those useful editing actions.

Sorting a Table Column

You can sort certain table columns by using the  **Sort Ascending** or  **Sort Descending** actions that are available on the toolbar or from the contextual menu.

The sorting result depends on the data type of the column content. It could be a numerical sorting for numbers or an alphabetical sorting for text information. 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 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 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.

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 Editing Mode

You can easily arrange 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 Editing Mode

The selection in the **Grid** mode is a bit complex compared to the selection in a text component. It consists of a currently 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 currently selected cell is painted with a color that is different from the rest of the selection.

You can also select discontinuous regions of nodes and place them in the clipboard with the copy action. To deselect one of the selected fragments, use **Ctrl + Single-Click (Command + Single-Click on OS X)**.

Pasting Content Within Grid Mode

You can paste the copied nodes relative to the currently selected cell using one of the following actions (available in the contextual menu):

-  **Paste (Ctrl + V (Command + V on OS X))** - Pastes the content, as a sibling, just below (after) the current selection.
- **Paste as Child** - Pastes the content as the last child of the current selection.

Pasting Content from Grid Mode to Other Editors

Nodes that are 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.

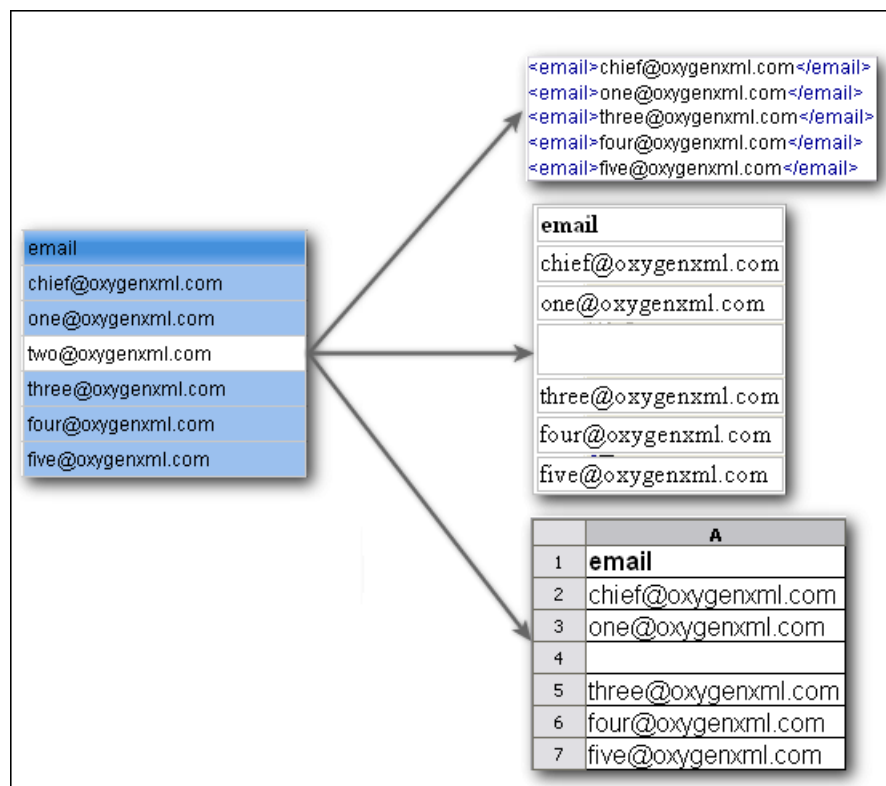
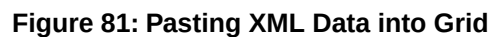


Figure 80: Copying from Grid to Other Editors

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



The diagram illustrates a mapping from a simple table to a table with an `@Id` column. On the left, a small table has two columns: `Id` and `Email`, with rows `Id1 Email1`, `Id2 Email2`, and `Id3 Email3`. An arrow points from this table to a larger table on the right. The larger table has two columns: `@Id` and `email`. The first row of the larger table is `1 BigBoss chief@oxygenxml.com`. Below this, the rows from the first table are mapped: `Id1` maps to `Email1`, `Id2` maps to `Email2`, and `Id3` maps to `Email3`. The rows in the larger table are numbered 2, 3, and 4, corresponding to the mapped rows.

Content Completion Assistant in Grid Mode

To display the content completion pop-up menu, simply start editing a cell (for example, double-click a cell) or press **Ctrl + Space (Command + Space on OS X)** on your keyboard.



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, for example, it is possible to create documents with lots of errors (for instance, 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 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 Developer 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 *Well-formed XML* and is also *Namespace-wellformed* and *Namespace-valid*.

Well-Formedness Rules

The XML Syntax rules for *Well-formed XML* are as follows:

- 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 as follows:

- 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 as follows:

- The *xml* prefix is by definition bound to the namespace name: <http://www.w3.org/XML/1998/namespace>. It MAY be declared, but MUST NOT be undeclared or bound to any other namespace name. Other prefixes MUST NOT be bound to this namespace name.
- The *xmlns* prefix 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.

Check for Well-Formedness

To check if a document is *Namespace Well-Formed XML*, select the  **Check Well-Formedness (Alt + Shift + V, W (Command + Alt + V, W on OS X))** action from the **XML** menu or from the  ▾ **Validation** drop-down menu on the toolbar.

The selected files in the current project can also 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](#).

If any errors are found, the result is displayed in the message panel at the bottom of the editor. Each error is displayed as 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 non *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 the record in the result list and it will locate and highlight the approximate position of the error. In this case, identify the tag that is missing an end tag and insert `</tag>`.

A non Namespace-wellformed Document

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

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

```
Element does not match QName production: QName::=(NCName ':' )?NCName.
```

A non 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.
```

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 also allows you to embed Schematron rules. For Schematron validations you can also select the *validation phase*.

Related information

[Presenting Schematron Validation Issues](#)

[Associating a Schema to XML Documents](#) on page 211

Automatic Validation

Oxygen XML Developer plugin can be configured to automatically mark validation errors in the document as you are editing. The [Enable automatic validation option](#) in the [Document Checking preferences page](#) controls whether or not all validation errors and warnings will automatically be highlighted 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. Hovering over a validation error presents a tooltip message with more details about the error.

Related information

[Manual Validation Actions](#) on page 194

[Presenting Validation Errors in Text Mode](#) on page 125

Manual Validation Actions

You can choose to validate documents at any time by using the manual validation actions that are available in Oxygen XML Developer plugin.

Manually Validate Current Document

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

Validate (Alt + Shift + V, V)

Available from the  ▾ **Validation** drop-down menu on the toolbar, the **XML** menu, or from the **Validate** submenu when invoking the contextual menu in the **Navigator** view.

An error list is presented in the message panel at the bottom of the editor. Markup of the 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.

Validate (cached)

Available from the  ▾ **Validation** drop-down menu on the toolbar or the **XML** 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.

Validate with

Available from the  ▾ **Validation** drop-down menu on the toolbar, (or **XML** menu).

This action opens a dialog box that allows you to *specify a schema for validating the current document*.

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 at the bottom of the editor. Markup of current document is checked to conform with the specified schema rules.

Validate with Schema

Available from the **Validate** submenu when invoking contextual menu in the **Navigator** view.

This action opens a dialog box that allows you to *specify a schema for validating all selected files*.

Other Validation Options

To quickly open the schema used for validating the current document, select the  **Open Associated Schema** action from the toolbar (or **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 [Document Checking preferences page](#)*.

Related information

[Automatic Validation](#) on page 194

[Presenting Validation Errors in Text Mode](#) on page 125

Presenting Validation Errors in Text Mode

Oxygen XML Developer plugin can be configured to *automatically validate documents* while editing in the **Text** mode, and actions are also available to *manually validate documents* on-request.

A line with a validation issue is marked in the editor panel by underlining the problem region with a colored line, according to the type of issue. The issues are also marked with a colored marker in the right vertical stripe.

- **Validation Errors** - By default, underlined with a red squiggly line in the editing pane and a red marker in the right vertical stripe.
- **Validation Warnings** - By default, underlined with a yellow squiggly line in the editing pane and a yellow marker in the right vertical stripe.
- **Validation Info** - By default, underlined with a blue squiggly line in the editing pane and a blue marker in the right vertical stripe.

Hovering over a validation issue presents a tooltip message with more details about the problem and *possible quick fixes* (if available for that issue).

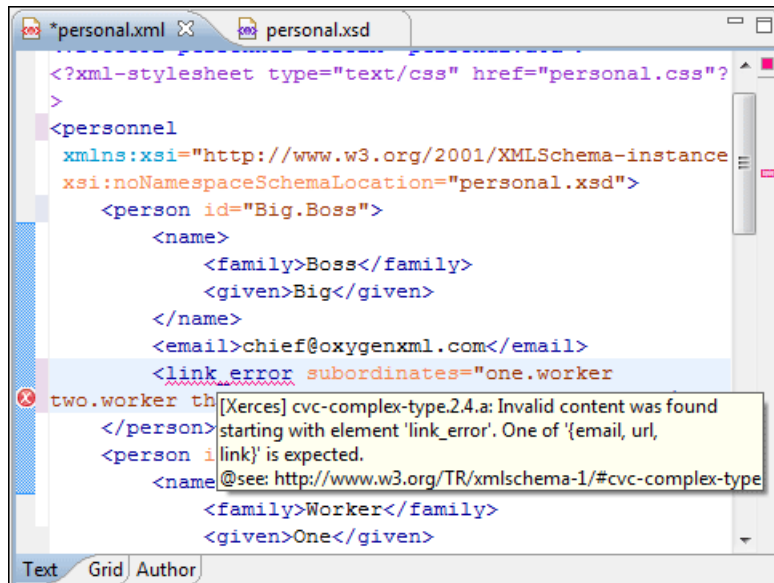


Figure 84: Presenting Validation Errors in Text Mode

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

Upper Part of the Stripe

A success indicator square will turn green if the validation is successful or only info messages are found, red if validation errors are found, or yellow if only validation warnings are found. More details about the issues are displayed in a tool tip when you hover over indicator square. If there are numerous problems, only the first three are presented in the tool tip.

Middle Part of the Stripe

Errors are depicted with red markers, warnings with yellow markers, and info message with blue markers. If you want to limit the number of markers that are displayed, [open the Preferences dialog box](#), go to **Editor > Document checking**, and specify the desired limit in the [Maximum number of validation highlights option](#).

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

Bottom Part of the Stripe

Two navigation arrows () allow you to jump to the next or previous issue. The same actions can be triggered from **Document > Automatic validation > Next error (Ctrl + Period (Command + Period on OS X))** and **Document > Automatic validation > Previous error (Ctrl + Comma (Command + 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 the **View** preferences page).

If you want to see all the validation messages grouped in the [Results view](#), you should use the  **Validate** action from the toolbar or **XML** menu. This action also collects the validation messages and displays them in the **Problems** view if the validated file is in the current workspace or in a custom **Errors** view if the validated file is outside the workspace.

Related information

[Validating XML Documents Against a Schema](#) on page 194

[Presenting Schematron Validation Issues](#)

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.

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 [Custom Validation Engines preferences page](#). 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 that do not support the [output message format](#) of Oxygen XML Developer plugin for linked messages:

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

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

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

 **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 Developer plugin. It is associated to XML Editor and XSD Editor. It is able to validate XML Schema schemas and XML documents against XML Schema schemas. The validation is done according to the W3C XML Schema 1.0 or 1.1. This can be [configured in Preferences](#).
- **MSXML 4.0** - Included in Oxygen XML Developer plugin (Windows edition only). It is associated to XML Editor, XSD Editor and XSL Editor. It is able to validate the edited document against XML Schema, internal DTD (included in the XML document), external DTD or a custom schema type.
- **MSXML.NET** - Included in Oxygen XML Developer plugin (Windows edition only). It is associated to XML Editor, XSD Editor and XSL Editor. It is able to validate the edited document against XML Schema, internal DTD (included in the XML document), external DTD or a custom schema type.
- **XSV** - Not included in Oxygen XML Developer plugin. Windows and Linux distributions of XSV can be downloaded from <http://www.cogsci.ed.ac.uk/~ht/xsv-status.html>. The executable path is [already configured in Oxygen XML Developer plugin](#) for the `[OXYGEN_INSTALL_DIR]/xsv` installation folder. If it is installed in a different folder, the predefined executable path must be [corrected in Preferences](#). It is associated to XML Editor and XSD Editor. It is able to validate the edited document against XML Schema or a custom schema type.
- **SQC (Schema Quality Checker from IBM)** - Not included in Oxygen XML Developer plugin. It can be downloaded [from here](#) (it comes as a .zip file, at the time of this writing SQC2.2.1.zip is about 3 megabytes). The executable path and working directory are already configured for the SQC installation directory `[OXYGEN_INSTALL_DIR]/sqc`. If it is installed in a different folder, the predefined executable path and working directory must be [corrected in the Preferences page](#). It is associated to XSD Editor.

Linked Output Messages of an External Engine

Validation engines display messages in an output view at the bottom of the Oxygen XML Developer plugin window. If such an output message (*warning*, *error*, *fatal error*, etc) spans between three to six lines of text and has the format specified below, 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 Scenarios

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

A typical example is the chunking of a DocBook XSL stylesheet that 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 various environments and have the same results. For example, an XSLT stylesheet may be applied with Saxon 6.5, Xalan, and MSXML 4.0 engines in different production systems.

Other examples of documents that can benefit from a validation scenario include:

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



Note: If a master file is associated with the current file, the validation scenarios defined in the master file are used and take precedence over the default scenarios defined for the particular framework. For more information on master files, see [Master Files Support](#) on page 159 or [Working with Modular XML Files in the Master Files Context](#) on page 218.

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

Related information

[Validating XML Documents Against a Schema](#) on page 194

[Presenting Validation Errors in Text Mode](#) on page 125

Creating a New Validation Scenario

To create a validation scenario, follow these steps:

1. Select the **Configure Validation Scenario(s)** from the toolbar, or from the **XML** menu (or 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 predefined and user-defined scenarios. The predefined scenarios are organized in categories depending on the type of file they apply to and you can identify them by a yellow key icon that marks them as *read-only*. The user-defined scenarios are organized under a single category. The default scenarios for the particular framework are rendered in bold.



Note: If a master file is associated with the current file, the validation scenarios defined in the master file are used and take precedence over the default scenarios defined for the particular framework. For more information on master files, see [Master Files Support](#) on page 159 or [Working with Modular XML Files in the Master Files Context](#) on page 218.

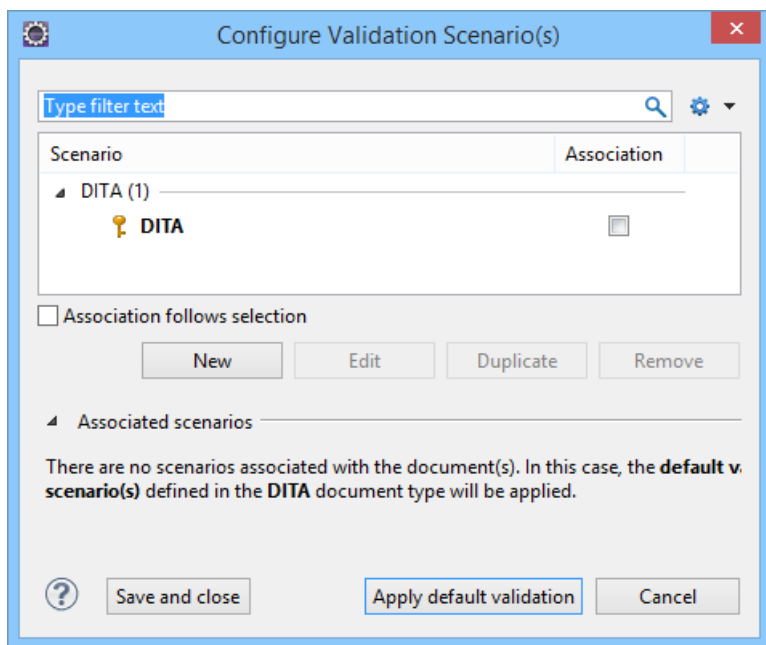


Figure 85: Configure Validation Scenario Dialog Box

The top section of the dialog box contains a filter that allows you to search through the scenarios list and the **Settings** button allows you to configure the following options:

Show all scenarios

Select this option to display all the available scenarios, regardless of the document they are associated with.

Show only the scenarios available for the editor

Select this option to only display the scenarios that Oxygen XML Developer plugin can 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 Developer plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:

- Keep or replace the existing scenario.
- Keep both scenarios.



Note: When you keep both scenarios, Oxygen XML Developer plugin adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export selected scenarios individually. Oxygen XML Developer plugin creates a `scenarios` file that contains the scenarios that you export. This is useful if you want to share scenarios with others or export them to another computer.

2. To add a scenario, click the **New** button.
The **New scenarios** dialog box is displayed and it lists all the validation units for the scenario.

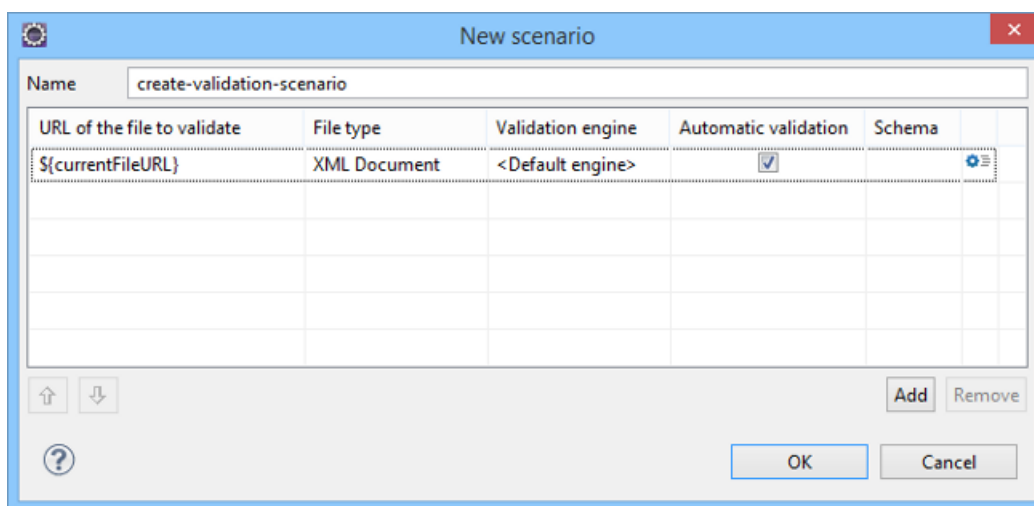


Figure 86: Create New Validation Scenario

This scenario configuration dialog box allows you to configure the following information and options:

Name

The name of the 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. To edit the URL, click its cell and 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*.

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

Figure 87: Insert an Editor Variable

File type

The type of the document that is validated in the current validation unit. Oxygen XML Developer plugin automatically selects the file type depending on the value of the **URL of the file to validate** field.

Validation engine

You can select one of the engines available in Oxygen XML Developer plugin for validation of the particular document type.

Default engine means that the default engine is used to run the validation for the current document type, as specified in the preferences page for that type of document (for example, [XSLT preferences page](#), [XQuery preferences page](#), [XML Schema preferences page](#)).

The **DITA Validation** engine performs DITA-specific checks in the context of the specifications.

The **Table Layout Validation** engine looks for table layout problems.

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 the [Document Checking preferences page](#), 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** and allows you to specify the schema used for the validation unit.

Specify Schema

Opens the **Specify Schema** dialog box that allows you to set a schema to be used for validating XML documents.

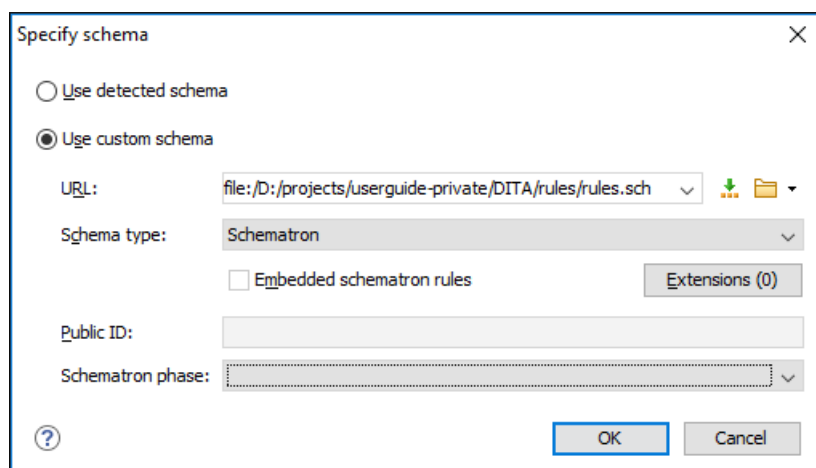


Figure 88: Specify Schema Dialog Box

The **Specify Schema** dialog box contains the following options:

Use detected schema

Uses the [schema detected for the particular document](#).

Use custom schema

Allows you to specify the schema using the following options:

- **URL** - Allows you to specify or select a URL for the schema. It also keeps a history of the last used schemas. The URL must point to the schema file that can be loaded from the local disk or from a remote server through HTTP(S), FTP(S). You can specify the URL by using the text field, the history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list.
- **Schema type** - Select a possible schema type from this combo box that is populated based on the extension of the schema file that was entered in the **URL** field. The possible schema types are: XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, or NVDL.
- **Embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules and you want to use those embedded rules, enable this option.
- **Extensions** - Opens a dialog box that allows you to specify Java extension JARs to be used during the validation.
- **Public ID** - Allows you to specify a public ID if you have selected a DTD.
- **Schematron phase** - If you select a Schematron schema, this drop-down list allows you to select a Schematron phase that you want to use for validation. The listed phases are defined in the Schematron document.

Move Up

Moves the selected scenario up one spot in the list.

Move Down

Moves the selected scenario down one spot in the list.

Add

Adds a new validation unit to the list.

Remove

Removes an existing validation unit from the list.

3. Configure any of the existing validation units according to the information above, and you can use the buttons at the bottom of the table to add, remove, or move validation units. Note that if you add a Schematron validation unit, you can also select the *validation phase*.
4. Press **OK**.
The newly created validation scenario will now be 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.

Editing a Validation Scenario

To edit an existing validation scenario, follow these steps:

1. Select the  **Configure Validation Scenario(s)** from the toolbar, or from the **XML** menu (or 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 predefined and user-defined scenarios. The predefined scenarios are organized in categories depending on the type of file they apply to and you can identify them by a yellow key icon that marks them as *read-only*. The user-defined scenarios are organized under a single category. The default scenarios for the particular framework are rendered in bold.



Note: If a master file is associated with the current file, the validation scenarios defined in the master file are used and take precedence over the default scenarios defined for the particular framework. For more information on master files, see [Master Files Support](#) on page 159 or [Working with Modular XML Files in the Master Files Context](#) on page 218.

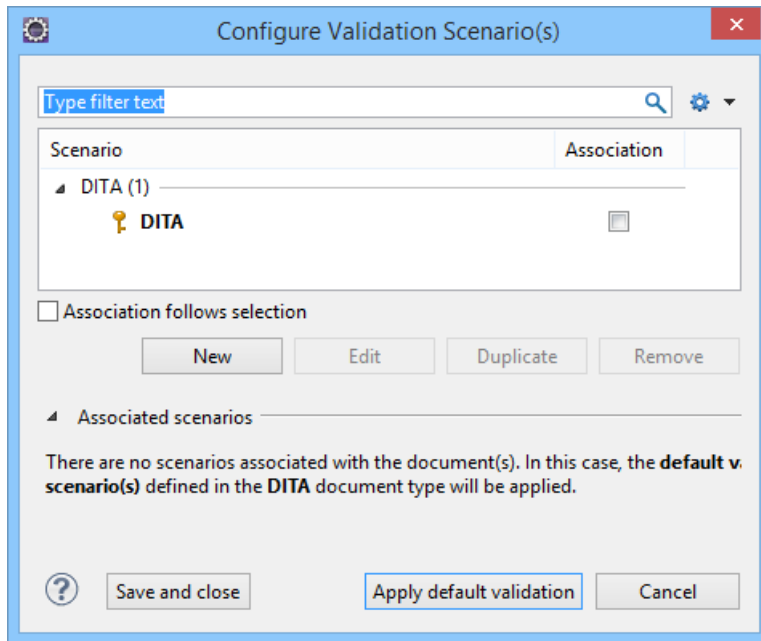


Figure 89: Configure Validation Scenario Dialog Box

The top section of the dialog box contains a filter that allows you to search through the scenarios list and the  **Settings** button allows you to configure the following options:

Show all scenarios

Select this option to display all the available scenarios, regardless of the document they are associated with.

Show only the scenarios available for the editor

Select this option to only display the scenarios that Oxygen XML Developer plugin can 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 Developer plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:

- Keep or replace the existing scenario.
- Keep both scenarios.



Note: When you keep both scenarios, Oxygen XML Developer plugin adds `imported` to the name of the imported scenario.

 Export selected scenarios

Use this option to export selected scenarios individually. Oxygen XML Developer plugin creates a `scenarios` file that contains the scenarios that you export. This is useful if you want to share scenarios with others or export them to another computer.

2. Select the scenario and press the **Edit** button. If you try to edit one of the *read-only* predefined scenarios, you will receive a warning message that Oxygen XML Developer plugin needs to create customizable duplicate (you can also use the **Duplicate** button).

The **Edit scenario** dialog box is displayed and it lists all the validation units for the scenario.

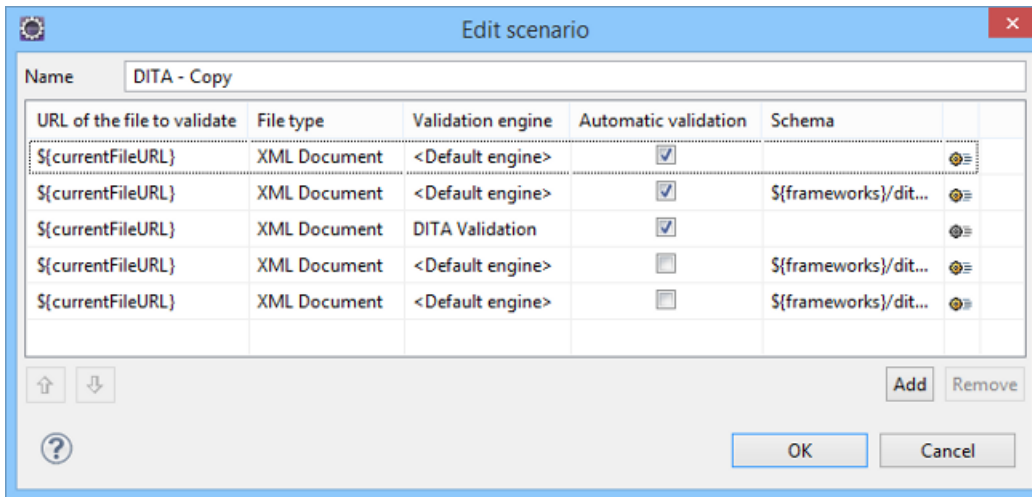


Figure 90: Edit Validation Scenario

This scenario configuration dialog box allows you to configure the following information and options:

Name

The name of the 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. To edit the URL, click its cell and 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*.

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

Figure 91: Insert an Editor Variable

File type

The type of the document that is validated in the current validation unit. Oxygen XML Developer plugin automatically selects the file type depending on the value of the **URL of the file to validate** field.

Validation engine

You can select one of the engines available in Oxygen XML Developer plugin for validation of the particular document type.

Default engine means that the default engine is used to run the validation for the current document type, as specified in the preferences page for that type of document (for example, [XSLT preferences page](#), [XQuery preferences page](#), [XML Schema preferences page](#)).

The **DITA Validation** engine performs DITA-specific checks in the context of the specifications.

The **Table Layout Validation** engine looks for table layout problems.

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 the [Document Checking preferences page](#), 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** and allows you to specify the schema used for the validation unit.

Specify Schema

Opens the **Specify Schema** dialog box that allows you to set a schema to be used for validating XML documents.

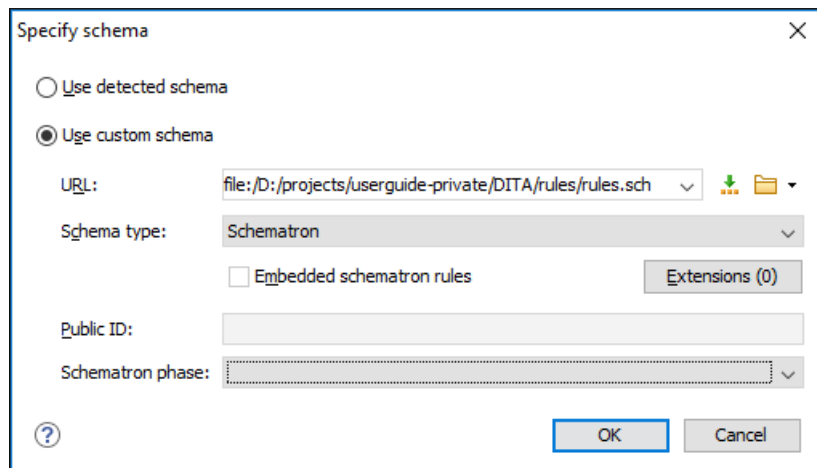


Figure 92: Specify Schema Dialog Box

The **Specify Schema** dialog box contains the following options:

Use detected schema

Uses the [schema detected for the particular document](#).

Use custom schema

Allows you to specify the schema using the following options:

- **URL** - Allows you to specify or select a URL for the schema. It also keeps a history of the last used schemas. The URL must point to the schema file that can be loaded from the local disk or from a remote server through HTTP(S), FTP(S). You can specify the URL by using the text field, the history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list.
- **Schema type** - Select a possible schema type from this combo box that is populated based on the extension of the schema file that was entered in the **URL** field. The possible schema types are: XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, or NVDL.
- **Embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules and you want to use those embedded rules, enable this option.
- **Extensions** - Opens a dialog box that allows you to specify Java extension JARs to be used during the validation.
- **Public ID** - Allows you to specify a public ID if you have selected a DTD.
- **Schematron phase** - If you select a Schematron schema, this drop-down list allows you to select a Schematron phase that you want to use for validation. The listed phases are defined in the Schematron document.

Move Up

Moves the selected scenario up one spot in the list.

Move Down

Moves the selected scenario down one spot in the list.

Add

Adds a new validation unit to the list.

Remove

Removes an existing validation unit from the list.

3. Configure any of the existing validation units according to the information above, and you can use the buttons at the bottom of the table to add, remove, or move validation units. Note that if you add a Schematron validation unit, you can also select the *validation phase*.
4. When you are done configuring the scenario, press **OK**.
The modified validation scenario will now be included in the list of scenarios in the **Configure Validation Scenario(s)** dialog box. If you chose to duplicate an existing one, the modified scenario will be listed with the word *copy* at the end of its name.

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.

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 actually be used 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 like this:

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

An XML catalog can also be used to map a W3C XML Schema specified with a 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 this:

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

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 (`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|functsynopsis
|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 `listitem` element is recommended. However, the error message does offer a clue as to the source of the problem, indicating that “The content of element type must match”.

Fortunately, most standards-based DTDs, XML Schemas, and Relax NG schemas are supplied with reference documentation. This enables you 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.

Finding and Replacing Text in the Current File

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

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

Find/Replace Dialog Box

To open the **Find/Replace** dialog box, use the **Find/Replace** action that is available in the **Edit** menu or by pressing **Ctrl + F (Command + F on OS X)**. It is also invoked by the **Find/Replace** contextual menu action found in certain views.

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

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



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

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

- Press **Ctrl + Enter (Command + Enter on OS X)** on your keyboard.
- Use the **Insert newline** contextual menu action.

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

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

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

The **Find/Replace** dialog box contains the following options:

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



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



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

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

- **Direction** - Specifies if the search direction is from current position to end of file (**Forward**) or to start of file (**Backward**).
- **Scope** -
- **Case sensitive** - When checked, the search operation follows the exact letter case of the text entered in the **Find** field.
- **Whole words only** - Only entire occurrences of a word are included in the search operation.
- **Incremental** - The search operation is started every time you type or delete a letter in the **Find** text box.
- **Regular expression** - When this option is enabled, you can use regular expressions in *Perl-like regular expressions syntax* to look for specific pieces of text.
 - **Dot matches all** - A dot used in a regular expression also matches end of line characters.
 - **Canonical equivalence** - If enabled, two characters will be considered a match if, and only if, their full *canonical* decompositions match. For example, the ã symbol can be inserted as a single character or as two characters (the a character, followed by the tilde character). This option is disabled by default.
- **Wrap search** - When the end of the document is reached, the search operation is continued from the start of the document, until its entire content is covered.
- **Find** - Executes a find operation for the next occurrence of the target. It stops after highlighting the find match in the editor panel.
- **Replace** - Executes a replace operation for the target.
- **Replace/Find** - Executes a replace operation for the target followed by a find operation for the next occurrence.
- **Replace All** - Executes a replace operation in the entire scope of the document.

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.

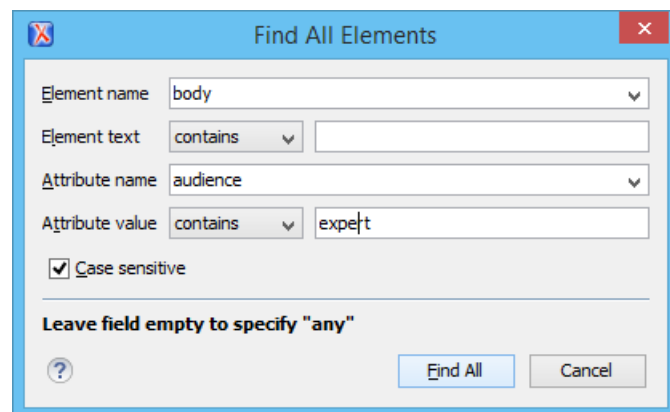


Figure 93: Find All Elements Dialog Box

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 Developer plugin tries to find the items that match all the search parameters. The results of the operation are presented as a list in the message panel.

Regular Expressions Syntax

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

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

When using regular expressions, note that some sets of characters from [XPath/XML Schema/Schematron](#) are slightly different than the ones used by Oxygen XML Developer plugin/Java in the text searches from the **Find/Replace** dialog boxes. The most common example is with the `\w` and `\W` set of characters. To ensure consistent results between the two, it is recommended that you use the following constructs in the **Find/Replace** dialog boxes:

- `/w` - `[#x0000-#x10FFFF] - [\p{P}\p{Z}\p{C}]` instead of `\w`
- `/W` - `[\p{P}\p{Z}\p{C}]` instead of `\W`

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 where 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 such as `+`, `?` and `*`.

Related information

[Comparison between the Java and Perl 5 regular expression syntax](#)

Search and Refactor Actions for IDs and IDREFS

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

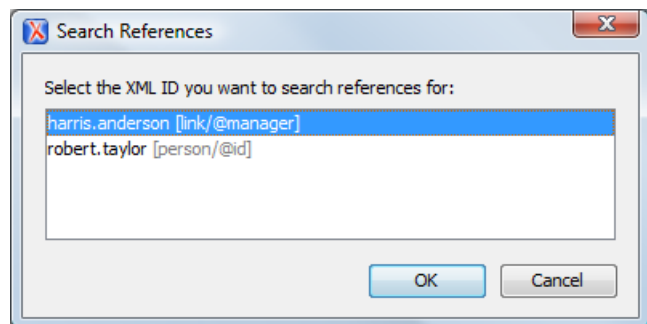


Figure 94: Selecting an ID in the Author Mode

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**

checkbox 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](#).

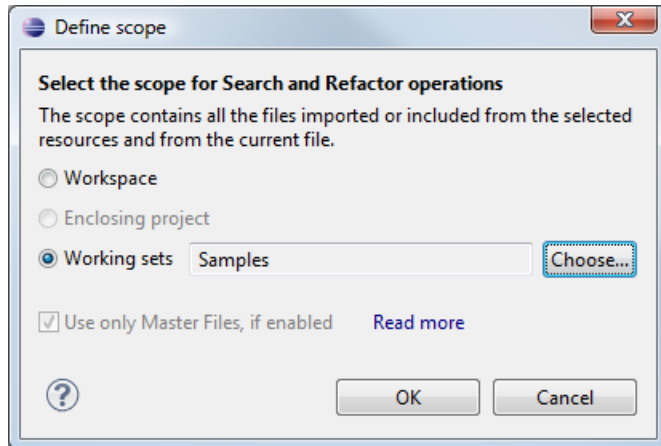


Figure 95: Change Scope Dialog Box

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.

Associating a Schema to XML Documents

Oxygen XML Developer plugin relies on schemas to validate XML documents and to compute valid proposals for the Content Completion Assistant.

Supported Types of Schema

The following schema types are supported:

- 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)

Detecting a Schema

For the purposes of using validation and content completion mechanisms, Oxygen XML Developer plugin tries to detect a schema by searching multiple locations, in the following order:

- The [schema defined in a validation scenario](#).
- The [schema defined in the validation scenarios associated with the particular document type](#) (if defined).
- The [schema that is associated directly in the XML document](#).
- The [schema defined in the Document Type Association](#).

In addition, the locations of the schema can be mapped through the use of an [XML Catalog](#). For more information, see [Resolving Schema Locations Through XML Catalogs](#) on page 218.



Tip: To quickly open the schema used for validating the current document, select the  **Open Associated Schema** action from the toolbar (or **XML** menu).

Associating a Schema Through a Validation Scenario

Oxygen XML Developer plugin uses the rules defined in the detected schema to report errors and warnings during automatic and manual validations that help maintain the structural integrity of your XML documents. You can specify the schema to be used for validation directly in [validation scenarios](#) and there are several methods that can be used to do so.

Configure a Validation Scenario and Specify the Schema

To associate a schema to a validation scenario to be used whenever the scenario is invoked, follow these steps:

1. Select the  **Configure Validation Scenario(s)** from the toolbar, or from the **XML** menu (or the **Validate** submenu when invoking the contextual menu on a file in the **Navigator** view).
2. Press the **New** button to create a new validation scenario or the **Edit** button to modify an existing one.
3. *Add or configure validation units* according to your needs and click the  **Specify Schema** button.

Step Result: The **Specify Schema** dialog box is displayed:

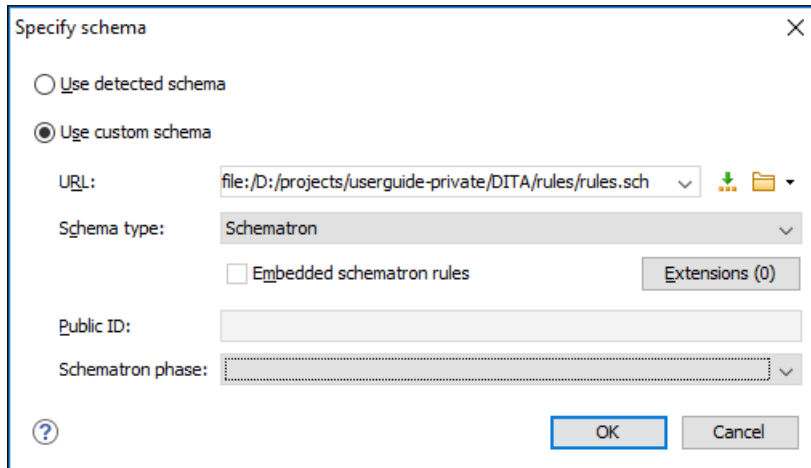


Figure 96: Specify Schema Dialog Box

The **Specify Schema** dialog box contains the following options:

Use detected schema

Uses the *schema detected for the particular document*.

Use custom schema

Allows you to specify the schema using the following options:

- **URL** - Allows you to specify or select a URL for the schema. It also keeps a history of the last used schemas. The URL must point to the schema file that can be loaded from the local disk or from a remote server through HTTP(S), FTP(S). You can specify the URL by using the text field, the history drop-down, the  **Insert Editor Variables** button, or the browsing tools in the  **Browse** drop-down list.
 - **Schema type** - Select a possible schema type from this combo box that is populated based on the extension of the schema file that was entered in the **URL** field. The possible schema types are: XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, or NVDL.
 - **Embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules and you want to use those embedded rules, enable this option.
 - **Public ID** - Allows you to specify a public ID if you have selected a DTD.
 - **Extensions** - Opens a dialog box that allows you to specify Java extension JARs to be used during the validation.
 - **Schematron phase** - If you select a Schematron schema, this drop-down list allows you to select a Schematron phase that you want to use for validation. The listed phases are defined in the Schematron document.
4. Select the schema to be associated with the validation unit and configure the rest of the options according to your preferences.
 5. Click **OK** on both dialog boxes.

Result: The schema is now associated with that validation scenario whenever it is invoked.

Use the Validate with Action to Specify a Schema for Validating the Current Document

To validate the current document using a specified schema, follow these steps:

1. Select the **Validation with** action from the  **Validation** drop-down menu on the toolbar (or **XML** menu).

Step Result: The **Validate with** dialog box is displayed:

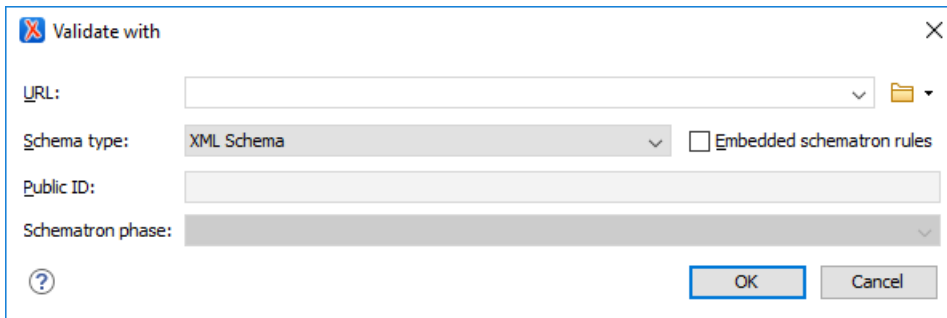


Figure 97: Validate with Dialog Box

This dialog box contains the following options:

- **URL** - Allows you to specify or select a URL for the schema. It also keeps a history of the last used schemas. The URL must point to the schema file that can be loaded from the local disk or from a remote server through HTTP(S), FTP(S). You can specify the URL by using the text field, the history drop-down, the **Insert Editor Variables** button, or the browsing tools in the **Browse** drop-down list.
 - **Schema type** - Select a possible schema type from this combo box that is populated based on the extension of the schema file that was entered in the **URL** field. The possible schema types are: XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, or NVDL.
 - **Embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules and you want to use those embedded rules, enable this option.
 - **Public ID** - Allows you to specify a public ID if you have selected a DTD.
 - **Extensions** - Opens a dialog box that allows you to specify Java extension JARs to be used during the validation.
 - **Schematron phase** - If you select a Schematron schema, this drop-down list allows you to select a Schematron phase that you want to use for validation. The listed phases are defined in the Schematron document.
2. Select the schema to be associated with the manual validation and configure the rest of the options according to your preferences.
 3. Click **OK**.

Result: The current document is validated using the schema you specified.

Tip: To quickly open the schema used for validating the current document, select the **Open Associated Schema** action from the toolbar (or **XML** menu).

Use the Validate with Schema Action to Specify a Schema for Validating all Selected Documents

To validate multiple documents using a specified schema, follow these steps:

1. Select all the documents you want to validate in the **Navigator** view.
2. Invoke the contextual menu (right-click) and select the **Validate with Schema** action from the **Validate** submenu.

Step Result: The **Validate with** dialog box is displayed:

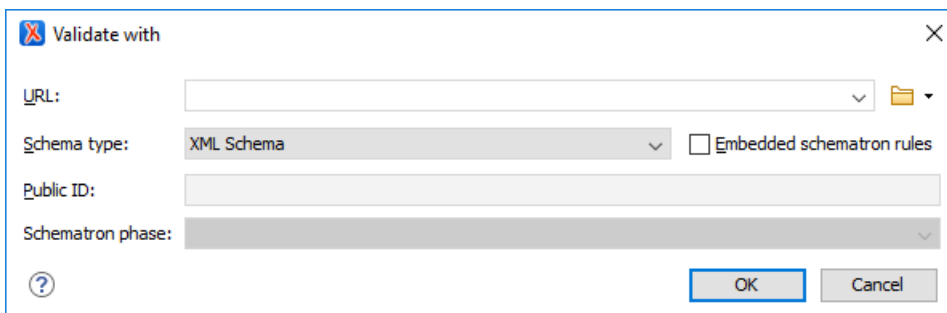


Figure 98: Validate with Dialog Box

This dialog box contains the following options:

- **URL** - Allows you to specify or select a URL for the schema. It also keeps a history of the last used schemas. The URL must point to the schema file that can be loaded from the local disk or from a remote server through HTTP(S), FTP(S). You can specify the URL by using the text field, the history drop-down, the  **Insert Editor Variables** button, or the browsing tools in the  **Browse** drop-down list.
 - **Schema type** - Select a possible schema type from this combo box that is populated based on the extension of the schema file that was entered in the **URL** field. The possible schema types are: XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, or NVDL.
 - **Embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules and you want to use those embedded rules, enable this option.
 - **Public ID** - Allows you to specify a public ID if you have selected a DTD.
 - **Extensions** - Opens a dialog box that allows you to specify Java extension JARs to be used during the validation.
 - **Schematron phase** - If you select a Schematron schema, this drop-down list allows you to select a Schematron phase that you want to use for validation. The listed phases are defined in the Schematron document.
3. Select the schema that you want to use to validate all selected documents and configure the rest of the options according to your preferences.
 4. Click **OK**.

Result: The selected documents are validated using the schema you specified.

Associating a Schema in Validation Scenarios Defined in the Document Type

To report errors and warnings during automatic and manual validations that help maintain the structural integrity of particular XML document types, Oxygen XML Developer plugin uses rules defined in the schema that is detected in the validation scenarios that are associated to each particular document type.

To associate a schema in validation scenarios defined in the *Document Type Association*, follow these steps:

1. [Open the Preferences dialog box](#) and go to **Document Type Association**.
2. Select your particular document type and click the **Edit** or **Duplicate** button to modify an existing framework (or use the **New** button to create a new one).

Step Result: This opens a [Document type configuration dialog box](#).

3. Go to the [Validation tab](#).
4. Create or edit a validation scenario:
 - a. To [create a new validation scenario](#), click the  **New** button.
 - b. To [edit the properties of an existing validation scenario](#), select it and click the  **Edit** button (you can also use the  **Duplicate** button to copy an existing scenario and edit its properties).
5. [Add or configure validation units](#) according to your needs and click the  **Specify Schema** button.

Step Result: The **Specify Schema** dialog box is displayed:

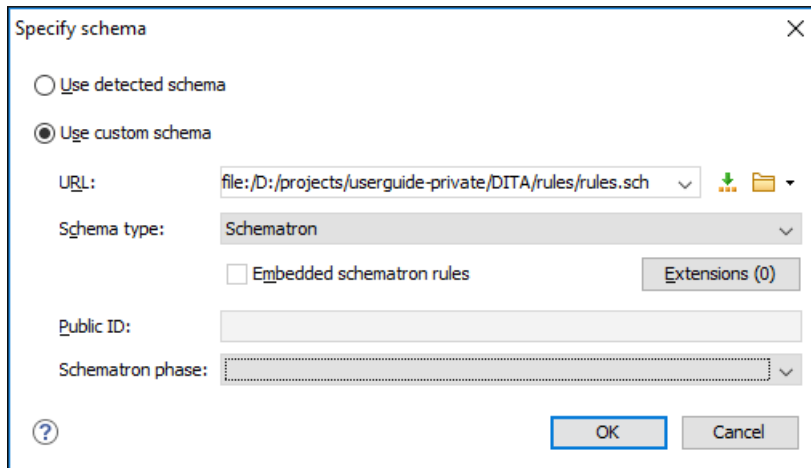


Figure 99: Specify Schema Dialog Box

The **Specify Schema** dialog box contains the following options:

Use detected schema

Uses the *schema detected for the particular document*.

Use custom schema

Allows you to specify the schema using the following options:

- **URL** - Allows you to specify or select a URL for the schema. It also keeps a history of the last used schemas. The URL must point to the schema file that can be loaded from the local disk or from a remote server through HTTP(S), FTP(S). You can specify the URL by using the text field, the history drop-down, the *Insert Editor Variables* button, or the browsing tools in the **Browse** drop-down list.
- **Schema type** - Select a possible schema type from this combo box that is populated based on the extension of the schema file that was entered in the **URL** field. The possible schema types are: XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, or NVDL.
- **Embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules and you want to use those embedded rules, enable this option.
- **Public ID** - Allows you to specify a public ID if you have selected a DTD.
- **Extensions** - Opens a dialog box that allows you to specify Java extension JARs to be used during the validation.
- **Schematron phase** - If you select a Schematron schema, this drop-down list allows you to select a Schematron phase that you want to use for validation. The listed phases are defined in the Schematron document.

6. Select the schema to be associated with the validation unit and configure the rest of the options according to your preferences.
7. Click **OK** on both dialog boxes.

Result: The schema is now associated with the validation scenario you just configured for that particular document type.

Associating a Schema Directly in XML Documents

Associate Schema Action

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

- **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.

To associate a schema to the current document, follow these steps:

1. Select the **Associate Schema** action from the toolbar (or **Document** > **Schema** menu).

Step Result: The **Associate Schema** dialog box is displayed:

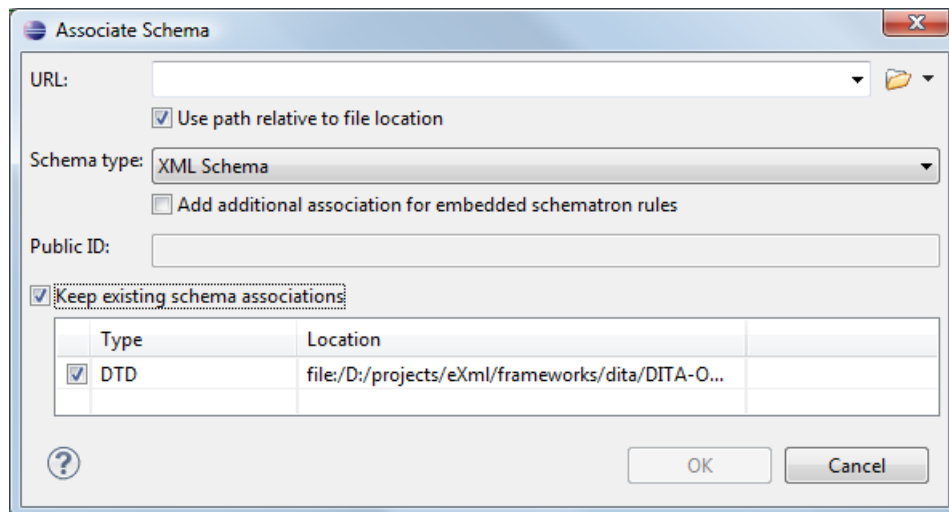


Figure 100: Associate Schema Dialog Box

This dialog box contains the following options:

- **URL** - Allows you to specify or select a URL for the schema. It also keeps a history of the last used schemas. The URL must point to the schema file that can be loaded from the local disk or from a remote server through HTTP(S), FTP(S).
 - **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 multiple project locations on physical disk.
 - **Schema type** - Select a possible schema type from this combo box that is populated based on the extension of the schema file that was entered in the **URL** field. The possible schema types are: XML Schema, DTD, Relax NG, Relax NG Compact, Schematron, or NVDL.
 - **Add additional association for embedded schematron rules** - If you have selected XML Schema or Relax NG schemas with embedded Schematron rules and you want to use those embedded rules, enable this option.
 - **Public ID** - Allows you to specify a public ID if you have selected a DTD.
 - **Keep existing schema associations** - Enable this option to use the existing schema associations of the currently edited document.
2. Select the schema that will be associated with the XML document and configure the rest of the options according to your preferences.
 3. Click **OK**.

Result: The schema association is created based upon the specified type.

- **XML Schema** - The association with an XML Schema is added as an attribute of the root element with 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.
- **DTD** - The association with a DTD is added as a `DOCTYPE` declaration.
- **Other** - The association with a Relax NG, Schematron, or NVDL schema is added as *xml-model processing instruction*.



Tip: To quickly open the schema used for validating the current document, select the  **Open Associated Schema** action from the toolbar (or **XML** menu).

Associate Schema with 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 prolog. 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/nvd1/ns/structure/1.0`.
- **phase** - The phase name for the validation function in Schematron schema. This is an optional attribute. To run all phases from the Schematron, use the special `#ALL` value. If the phase is not specified, the default phase that is configured in the Schematron will be applied.
- **title** - The title for the associated schema. This is an optional attribute.

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

- **RNGSchema** - Specifies the path to the Relax NG schema that is associated with the current document.
- **type** - Specifies the type of Relax NG schema. It is used along with the **RNGSchema** attribute and can have the value `xml` or `compact`.
- **NVDLSchema** - Specifies the path to the NVDL schema that is associated with the current document.
- **SCHSchema** - Specifies the path to the SCH schema that is associated with the current document.



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

Related information

[Validating XML Documents](#) on page 192

[Content Completion Assistant in Text Mode](#) on page 169

Associating a Schema in the Document Type Association

The schema used to compute valid proposals in the Content Completion Assistant and by the document validation engine to report errors and warnings can be defined in each particular document type (framework). This schema will be used only if one is not [detected in the current XML file](#).

To associate a schema in a particular *Document Type Association*, follow these steps:

1. [Open the Preferences dialog box](#) and go to **Document Type Association**.
2. Select your particular document type and click the **Edit** or **Duplicate** button to modify an existing framework (or use the **New** button to create a new one).

Step Result: This opens a [Document type configuration dialog box](#).

3. Go to the [Schema tab](#).
4. Select the schema type and its URI.
5. Click **OK**.

Result: The schema is now associated with the particular document type and will be used by the Content Completion Assistant and validation engine if a schema is not detected in the current XML document.

Resolving Schema Locations Through XML Catalogs

Schema locations can be mapped using an [XML Catalog](#). Oxygen XML Developer plugin resolves the location of a schema in the following order:

- First, it attempts to resolve the schema location as a URI (`uri`, `uriSuffix`, `rewriteUri`, `delegateUri` mappings from the *XML Catalog*). If this succeeds, the process ends here.
- If the [Resolve schema locations also through system mappings option](#) is selected, it attempts to resolve the schema location as a system ID (`system`, `systemSuffix`, `rewriteSuffix`, `rewriteSystem` from the *XML Catalog*). If this succeeds, the process ends here.
- If the [Process "schemaLocation" namespaces through URI mappings for XML Schema option](#) is selected, the target namespace of the imported XML Schema is resolved through URI mappings. If the schema specified in the `schemaLocation` attribute is not resolved successfully, the namespace of the root element is taken into account. If this succeeds, the process ends here.
- If none of these succeeds, the actual [schema location](#) is used.

Learn Document Structure when Schema is not Detected

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

When you open a document that is not associated with a schema, Oxygen XML Developer plugin automatically learns the document structure and uses it for [content completion](#). To disable this feature, disable the [Learn on open document option in the user preferences](#).

Related information

[Detecting a Schema](#) on page 211

Create a DTD from Learn Document Structure Option

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

To create a DTD from the learned structure, follow these steps:

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

Working with Modular XML Files in the Master Files Context

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

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

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

The advantages of working with modular XML files in the context of a master file include:

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

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

Related information

[Master Files Support](#) on page 159

[XML Resource Hierarchy/Dependencies View](#) on page 219

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 Developer plugin > Resource Hierarchy/Dependencies**. As an alternative, right-click the current document and either select **Resource Hierarchy** or **Resource Dependencies**.

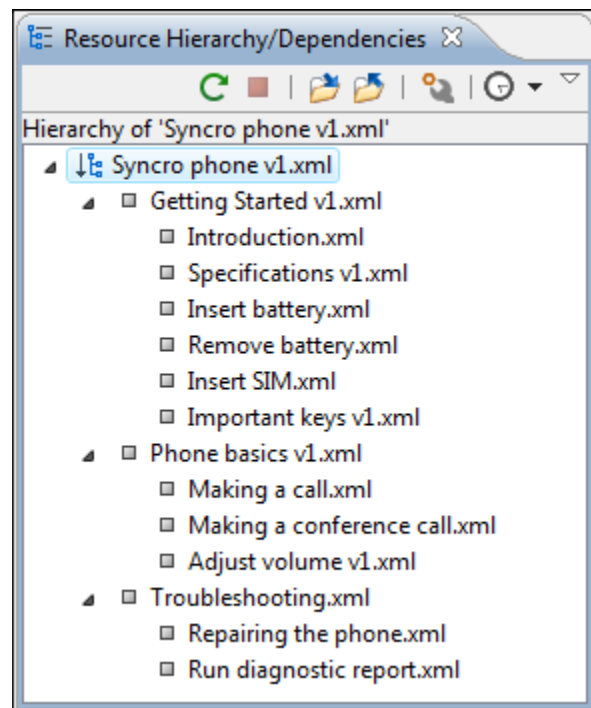


Figure 101: Resource Hierarchy/Dependencies View - Hierarchy for Syncro phone v1.xml

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

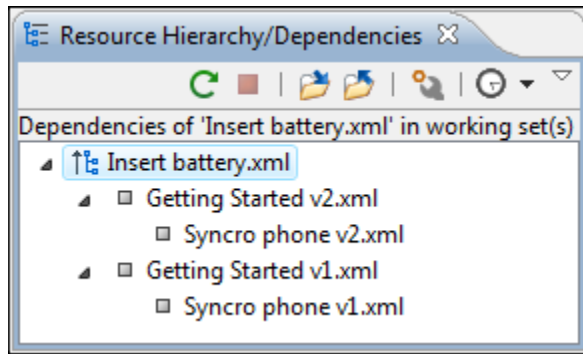


Figure 102: Resource Hierarchy/Dependencies View - Dependencies for Insert battery.xml

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.

Related information

[Search and Refactor Operations Scope](#) on page 210

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.

Working with XML Catalogs

An *XML Catalog* maps a system ID or a 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 (such as referenced schemas and stylesheets), the use of an XML Catalog becomes a necessity when Internet access is not available or the Internet connection is slow.

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

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, public ID, or a URI to a local copy using only a suffix of the ID or URI used in the actual document. This is done using the catalog elements [systemSuffix](#) and [uriSuffix](#).

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

Table 3: Catalog Mappings

Document	Referenced Resource	Mappings
XML	DTD	<i>system</i> or <i>public</i> The Prefer option controls which one of the mappings should be used.
	XML Schema	The following strategy is used (if one step fails to provide a resource, the next is applied): 1. resolve the schema using <i>URI</i> catalog mappings. 2. resolve the schema using <i>system</i> 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 <i>namespace</i> using <i>URI</i> catalog mappings.
	Relax NG	
	Schematron	
	NVDL	
XSL	XSL/ANY	<i>URI</i>

Document	Referenced Resource	Mappings
CSS	CSS	<i>URI</i>
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 <i>URI</i> catalog mappings. 2. resolve schema reference using <i>system</i> catalog mappings. This happens only if the Resolve schema locations also through system mappings option is enabled (it is by default). 3. resolve schema <i>namespace</i> using <i>uri</i> 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	

Creating an XML Catalog with a Template

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

  <!-- "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>
```

How Oxygen XML Developer plugin Determines which Catalog to Use

An XML catalog is used to resolve references for document validation and transformations and it maps such references to the built-in local copies of the schemas of the Oxygen XML Developer plugin frameworks (DocBook, DITA, TEI, XHTML, SVG, etc.)

Oxygen XML Developer plugin includes a built-in catalog set as default, but you can also create your own and Oxygen XML Developer plugin looks for catalogs in the following order:

- Global user-defined catalogs that are set in the [XML Catalog preferences page](#).
- User-defined catalogs that are set at document type level, in the [Catalog tab](#) from the [Document Type configuration dialog box](#).
- Default 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 this:

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

Related information

[XML Catalog Preferences](#) on page 77

Resolving Schema Locations Through XML Catalogs

Schema locations can be mapped using an [XML Catalog](#). Oxygen XML Developer plugin resolves the location of a schema in the following order:

- First, it attempts to resolve the schema location as a URI (`uri`, `uriSuffix`, `rewriteUri`, `delegateUri` mappings from the *XML Catalog*). If this succeeds, the process ends here.
- If the [Resolve schema locations also through system mappings option](#) is selected, it attempts to resolve the schema location as a system ID (`system`, `systemSuffix`, `rewriteSuffix`, `rewriteSystem` from the *XML Catalog*). If this succeeds, the process ends here.
- If the [Process "schemaLocation" namespaces through URI mappings for XML Schema option](#) is selected, the target namespace of the imported XML Schema is resolved through URI mappings. If the schema specified in the *schemaLocation* attribute is not resolved successfully, the namespace of the root element is taken into account. If this succeeds, the process ends here.
- If none of these succeeds, the actual [schema location](#) is used.

Editing Large XML Documents with DTD Entities or XInclude

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, which includes smaller instances, 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 you to manage content in a modular fashion that is akin to Object Oriented methods used in languages such as Java, C++ or C#.

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

Include a chapter in an article using XInclude

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

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

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

Main article file looks like this:

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

In this example, note the following:

- 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.

 **Notice:** Oxygen XML Developer plugin supports the *XPointer Framework* and the *XPointer element() Scheme*, but it does NOT support the *XPointer xpointer() Scheme*.

For example, if the master file is:

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

and the `a.xml` file is:

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

after resolving the XPointer reference the document is:

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

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

Viewing Status Information

Status information generated by operations such as *schema detection*, *manual validation*, *automatic validation*, and *transformations* are fed into the **Console** view, allowing you to monitor how the operation is being executed (the [Enable oxygen consoles option](#) must be enabled in the [View preferences page](#)).

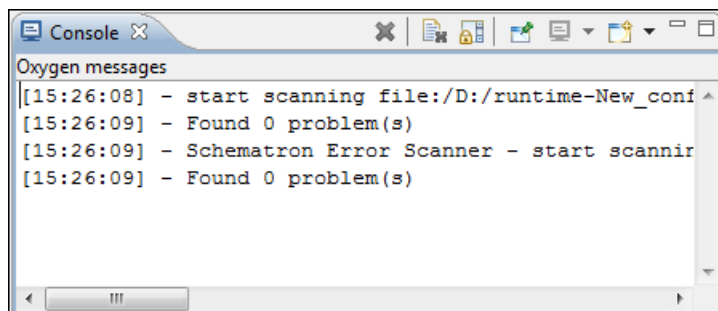


Figure 103: Console View Messages

Messages contain a timestamp, the name of the thread that generated it and the actual status information. The number of displayed messages can be controlled with the [Limit console output option in the View](#) preference page.

To make the view visible, select **Window > Show View > Console**.

Editor Highlights

An *editor highlight* is a text fragment emphasized by a colored background.

Highlights are generated when the following actions generate results:

- Find/Replace
- Find All Elements
- XPath in Files
- Search References
- Search Declarations

By default, Oxygen XML Developer 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 can 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 Developer plugin Quick Fix support helps you resolve errors that appear in an XML document by offering quick fixes to problems such as missing required attributes or invalid elements. Quick fixes are available in **Text** 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 pop-up window and the available quick fixes include a link that can be used to perform the fix.

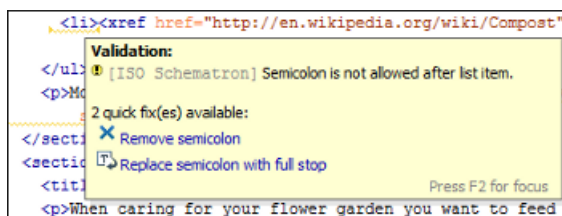


Figure 104: Quick Fix Presented in a Tooltip in Text Mode

- 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 Developer plugin displays the list of available fixes.

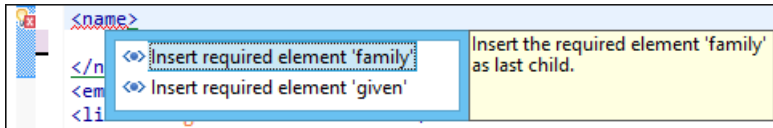


Figure 105: Quick Fix Menu Invoked by Clicking on the  Icon

- 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 (Command + 1 on OS X)** on your keyboard.

Whenever you make a modification in the XML document or you apply a fix, the list of quick fixes is recomputed to ensure that you always have valid proposals.

 **Note:** A quick fix that adds an element inserts it along with required and optional elements, and required and fixed attributes, depending on how the [Content Completion Assistant options](#) are configured.

Quick Fixes for DTD, XSD, and Relax NG Errors

Oxygen XML Developer plugin offers quick fixes for common errors that appear in XML documents that are validated against DTD, XSD, or Relax NG schemas.

 **Note:** For XML documents validated against XSD schemas, the quick fixes are only available if you use the default Xerces validation engine.

Quick fixes are available in **Text** mode .

Oxygen XML Developer plugin provides quick fixes for numerous 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

Related information

[Schematron Quick Fixes \(SQF\)](#) on page 227

Schematron Quick Fixes (SQF)

Oxygen XML Developer plugin provides support for Schematron Quick Fixes (SQF). They help you resolve issues 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 validation messages. Specifically, they are associated with *assert* or *report* messages.

A typical use case is using Schematron Quick Fixes to assist content authors 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 post: <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.

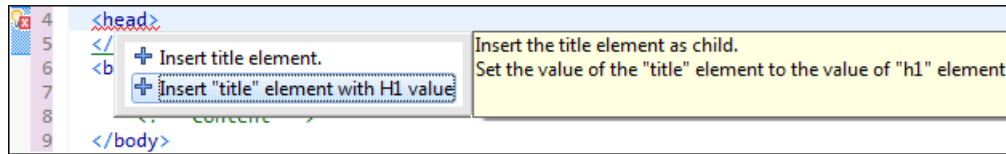


Figure 106: Example of a Schematron Quick Fix

Related information

[Editing Schematron Quick Fixes](#) on page 395

[Schematron Quick Fix Specifications](#)

[Presenting Schematron Validation Issues](#)

Refactoring XML Documents

In the life cycle of XML documents there are instances when the XML structure needs to be changed to accommodate various needs. For example, when an associated schema is updated, an attribute may have been removed, or a new element added to the structure.

These types of situations cannot be resolved with a traditional *Find/Replace* tool, even if the tool accepts regular expressions. The problem becomes even more complicated if an XML document is computed or referenced from multiple modules, since multiple resources need to be changed.

To assist you with these types of refactoring tasks, Oxygen XML Developer plugin includes a specialized **XML Refactoring** tool that helps you manage the structure of your XML documents.

XML Refactoring Tool

The **XML Refactoring** tool is presented in the form of an easy to use wizard that is designed to reduce the time and effort required to perform various structure management tasks. For example, you can insert, delete, or rename an attribute in all instances of a particular element that is found in all documents within your project.

To access the tool, select the  **XML Refactoring** action from one of the following locations:

- The **XML Tools** menu.
- The **Refactoring** submenu from the contextual menu in the **Navigator** view.



Note: The predefined refactoring operations are also available from the **Refactoring** submenu in the contextual menu of **Text** mode. This is useful because by selecting the operations from the contextual menu, Oxygen XML Developer 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.

XML Refactoring Wizard

The XML Refactoring tool includes the following wizard pages:

Refactoring operations

The first wizard page presents the available operations, grouped by category. To search for an operation, you can use the filter text box at the top of the page.

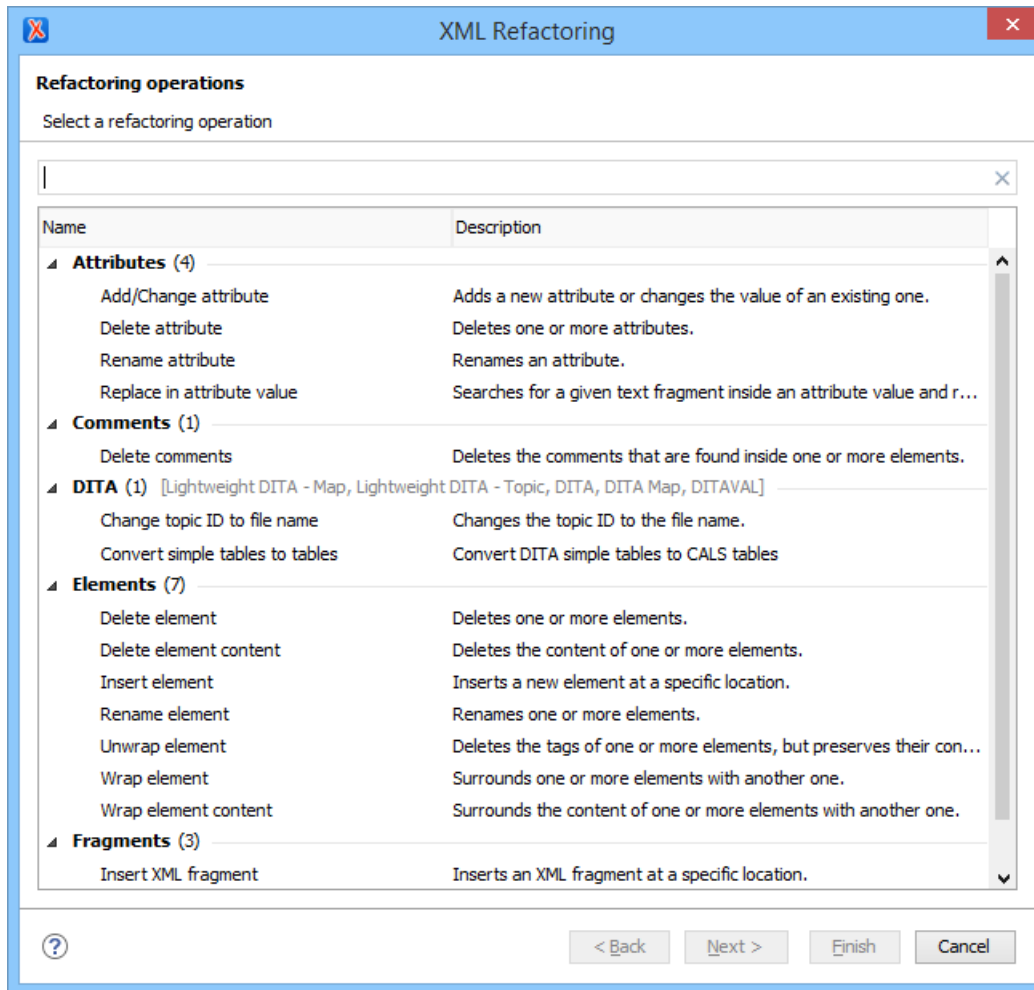


Figure 107: XML Refactoring Wizard

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.

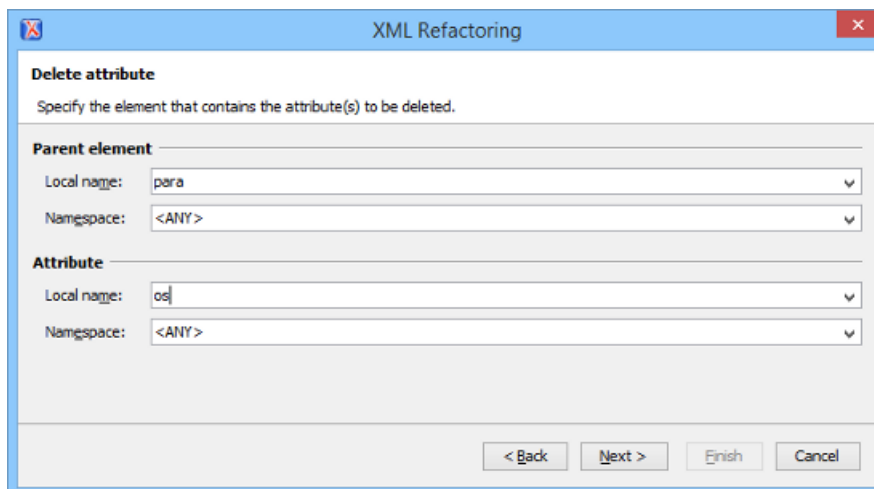


Figure 108: XML Refactoring 2nd Wizard Page (Delete Attribute Operation)

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.

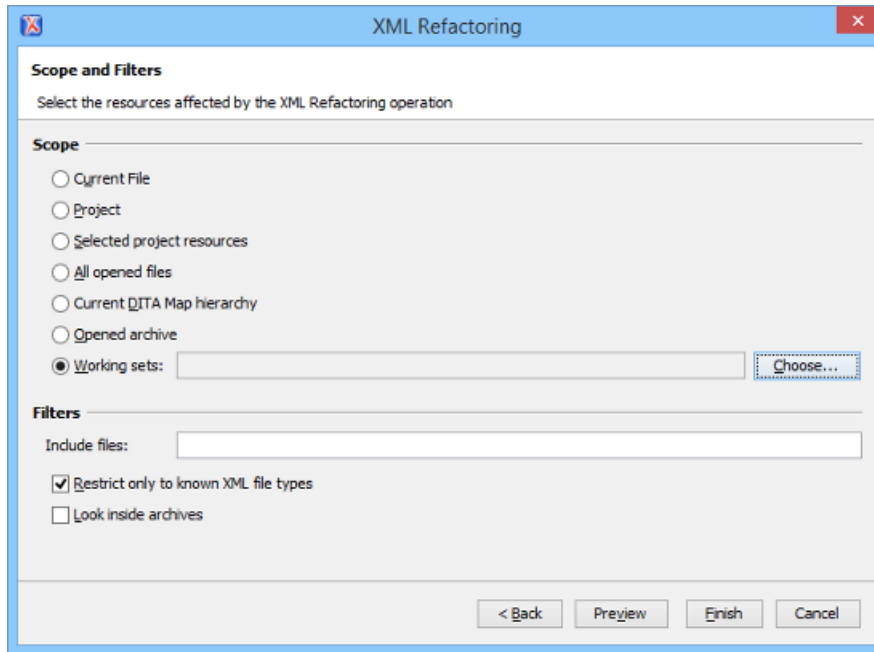


Figure 109: XML Refactoring - Scope and Filters Wizard Page

If an operation takes longer than expected you can use the  **Stop** button in the progress bar to cancel the operation.



Note: It is recommended that you use the **Preview** button to review all the changes that will be made by the refactoring operation before applying the changes.



Warning: After clicking the **Finish** button, the operation will be processed and Oxygen XML Developer plugin provides no automatic means for reverting the operations. Any **Undo** action will only revert changes on the current document.

Predefined Refactoring Operations

The XML Refactoring tool includes a variety of predefined operations that can be used for common refactoring tasks. They are grouped by category in the **Refactoring operations** wizard page. You can also access the operations from the **Refactoring** submenu in the contextual menu of **Text** mode. The operations are also grouped by category in this submenu. When selecting the operations from the contextual menu, Oxygen XML Developer 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.



Tip: Each operation includes a link in the lower part of the wizard that opens the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

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. This operation allows you to specify the following parameters:

- **Parent element** section
 - **Element** - 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.
- **Attribute** section
 - **Local name** - The local name of the affected attribute.
 - **Namespace** - The namespace of the affected attribute.
 - **Value** - The value for the affected attribute.
- **Options** section
 - You can choose between one of the following options for the **Operation mode**:
 - **Add the attribute in the parent elements where it is missing**
 - **Change the value in the parent elements where the attribute already exists**
 - **Both**

Delete attribute

Use this operation to remove one or more attributes. This operation requires you to specify the following parameters:

- **Element** - 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.
- **Attribute** - The name of the attribute to be deleted.

Rename attribute

Use this operation to rename an attribute. This operation requires you to specify the following parameters:

- **Element** - 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.
- **Attribute** - The name of the attribute to be renamed.
- **New local name** - The new local name of the attribute.

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. This operation allows you to specify the following parameters:

- **Target attribute** section
 - **Element** - 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.
 - **Attribute** - The name of the attribute to be modified.
- **Find / Replace** section
 - **Find** - The text fragments to find. You can use Perl-like regular expressions.
 - **Replace with** - The text fragment to replace the target with. This parameter can bind regular expression capturing groups (\$1, \$2, etc.) from the find pattern.

Refactoring Operations for *Comments*

Delete comments

Use this operation to delete comments from one or more elements. This operation requires you specify the following parameter:

- **Element** - The target element (or elements) for which comments will be deleted, in the form of a local name from any namespace, a local name with a namespace prefix, or an XPath expression.



Note: Comments that are outside the root element will not be deleted because the *serializer* preserves the content before and after the root.

Refactoring Operations for *DITA*

Change topic ID to file name

Use this operation to change the ID of a topic to be the same as its file name. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Convert simple tables to CALS tables

Use this operation to convert DITA simple tables to CALS tables. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Refactoring Operations for *Elements*

Delete element

Use this operation to delete elements. This operation requires you to specify the following parameter:

- **Element** - 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.

Delete element content

Use this operation to delete the content of elements. This operation requires you to specify the following parameter:

- **Element** - The target element whose 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.

Insert element

Use this operation to insert new elements. This operation allows you to specify the following parameters:

- **Element** section
 - **Local name** - The local name of the element to be inserted.
 - **Namespace** - The namespace of the element to be inserted.
- **Location** section
 - **XPath** - An XPath expression that identifies an existing element to which the new element is relative, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
 - **Position** - The position where the new element will be inserted, in relation to the specified existing element. The possible selections in the drop-down menu are: **After**, **Before**, **First child**, or **Last child**.

Rename element

Use this operation to rename elements. This operation requires you to specify the following parameters:

- **Target elements (XPath)** - The target elements to be renamed, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- **New local name** - The new local name of the element.

Unwrap element

Use this operation to remove the surrounding tags of elements, while keeping the content unchanged. This operation requires you to specify the following parameter:

- **Target elements (XPath)** - The target elements whose 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.

Wrap element

Use this operation to surround elements with element tags. This operation allows you to specify the following parameters:

- **Target elements (XPath)** - The target elements 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.
- **Wrapper element** section
 - **Local name** - The local name of the *Wrapper element*.
 - **Namespace** - The namespace of the *Wrapper element*.

Wrap element content

Use this operation to surround the content of elements with element tags. This operation allows you to specify the following parameters:

- **Target elements (XPath)** - The target elements whose content will 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.
- **Wrapper element** section
 - **Local name** - The local name of the *Wrapper element* that will surround the content of the target.
 - **Namespace** - The namespace of the *Wrapper element* that will surround the content of the target.

Refactoring Operations for *Fragments*

Insert XML fragment

Use this operation to insert an XML fragment. This operation allows you to specify the following:

- **XML Fragment** - The XML fragment to be inserted.
- **Location** section
 - **XPath** - An XPath expression that identifies an existing element to which the inserted fragment is relative, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
 - **Position** - The position where the fragment will be inserted, in relation to the specified existing element. The possible selections in the drop-down menu are: **After**, **Before**, **First child**, or **Last child**.

Replace element content with XML fragment

Use this operation to replace the content of elements with an XML fragment. This operation allows you to specify the following parameters:

- **Target elements (XPath)** - The target elements whose 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.
- **XML Fragment** - The XML fragment with which to replace the content of the target element.

Replace element with XML fragment

Use this operation to replace elements with an XML fragment. This operation allows you to specify the following parameters:

- **Target elements (XPath)** - The target elements to be replaced, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- **XML Fragment** - The XML fragment with which to replace the target element.

Refactoring Operations for JATSKit

Add BITS DOCTYPE - NLM/NCBI Book Interchange 2.0

Use this operation to add an NLM 'BITS' 2.0 DOCTYPE declaration. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Add Blue DOCTYPE - NISO JATS Publishing 1.1

Use this operation to add a JATS 'Blue' 1.1 DOCTYPE declaration. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Normalize IDs

Use this operation to normalize assigned IDs and assigned IDs to elements that are missing them.. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Additional Notes



Note: There are some operations that allow <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

While Oxygen XML Developer plugin includes a variety of predefined XML refactoring operations to help you accomplish particular tasks, you can also create custom operations according to your specific needs. For example, you could create a custom refactoring operation to convert an attribute to an element and insert the element as the first child of the parent element.

An XML Refactoring operation is defined as a pair of resources:

- An *XQuery Update script* or *XSLT stylesheet* that Oxygen XML Developer plugin will run to refactor the XML files.
- An *XML Operation Descriptor* file that contains information about the operation (such as the name, description, and parameters).

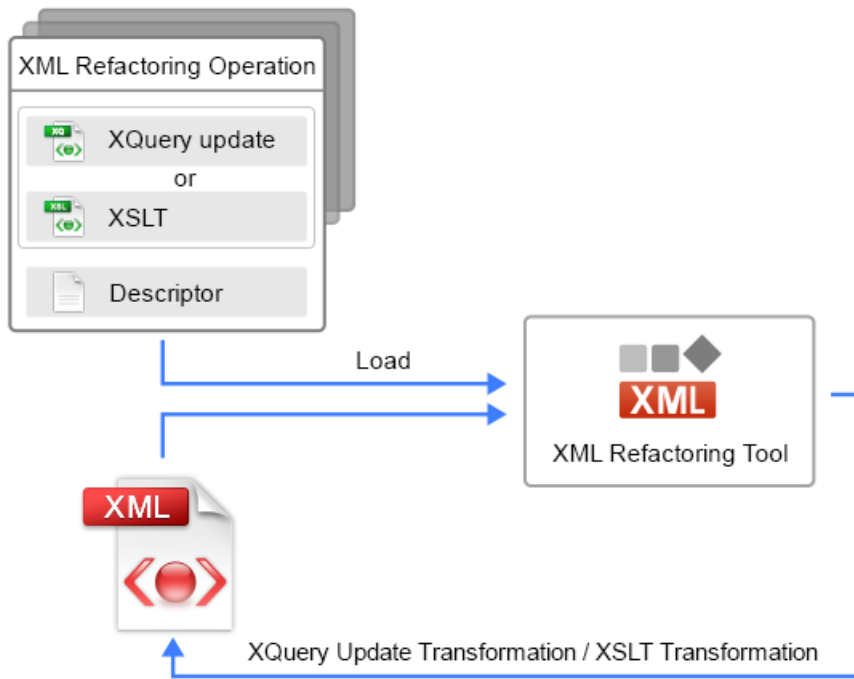


Figure 110: Diagram of an XML Refactoring Operation

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 Developer 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://www.oxygenxml.com/ns/xmlRefactoring/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 Developer plugin](#) scans when loading the custom operations.

Result: Once you run the **XML Refactoring** tool again, the custom operation appears in [the Refactoring Operations wizard page](#).

Related information

[Storing and Sharing Refactoring Operations](#) on page 242

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 Developer 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](#).

Related information

[Example of an XML Refactoring Operation](#) on page 239

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 Developer plugin will use a text input with corresponding content completion and syntax highlighting.



Note: The value of this parameter is transferred as plain text to the XQuery Update 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 Developer plugin will display a text area specialized for inserting XML documents.
- **NC_NAME** - The parameter for **NC_NAME** values. It is useful when you want to specify the local part of a *QName* for an element or attribute.
- **BOOLEAN** - Used to edit boolean parameters.
- **TEXT_CHOICE** - It is useful for parameters whose value should be from a list of possible values. Oxygen XML Developer plugin renders each possible value as a radio button 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 Developer 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>Local name of the parent element.</description>
  </localName>
  <namespace label="Namespace" name="element_namespace" allowsAny="true">
    <description>Local name of the 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>Namespace of the attribute to be converted.</description>
  </namespace>
</attributeParameter>
```



Note: All predefined operations are loaded from the `[OXYGEN_INSTALL_DIR]/refactoring` folder.

Related information

[Example of an XML Refactoring Operation](#) on page 239

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.

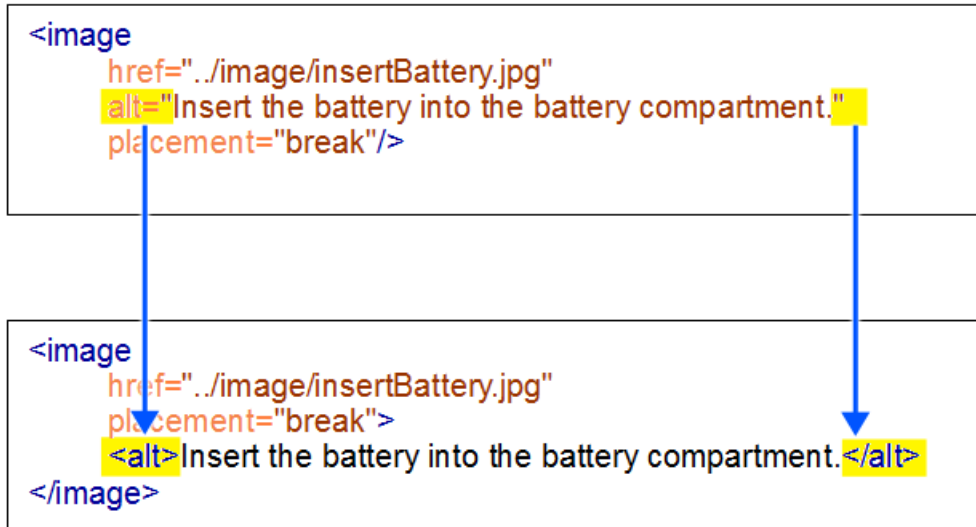


Figure 111: Example: Custom XML Refactoring Operation

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/attribute[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="attribute[local-name=$attribute_localName and
                namespace=$attribute_namespace]">
            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>Local name of the parent element.</description>
        </localName>
        <namespace label="Namespace" name="element_namespace" allowsAny="true">
          <description>Local name of the parent element</description>
        </namespace>
      </elementParameter>
    </section>
    <section label="Attribute">
      <attributeParameter dependsOn="elemID">
        <localName label="Name" name="attribute_localName">
          <description>Name of the attribute to be converted.</description>
        </localName>
        <namespace label="Namespace" name="attribute_namespace" allowsAny="true">
          <description>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 Developer plugin when it loads the custom operation*. When the XML Refactoring tool is started again, you will see the created operation.

Since various parameters can be specified, this custom operation can also be used for other similar tasks. The following image shows the parameters that can be specified in our example of the custom operation to convert an attribute to an element:

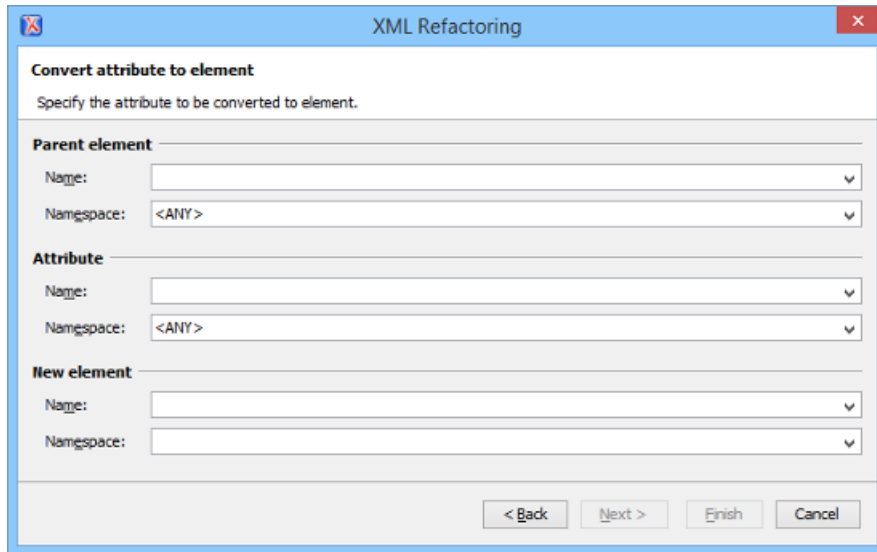


Figure 112: Example: XML Refactoring Wizard for a Custom Operation

Storing and Sharing Refactoring Operations

Oxygen XML Developer plugin scans the following locations when looking for XML Refactoring operations to provide flexibility:

- A refactoring folder, created inside a directory that is associated to a *framework* you are customizing.
- Any folder. In this case, you need to *open the Preferences dialog box*, go to **XML > XML Refactoring**, and specify the same folder in the *Load additional refactoring operations from text box*.
- The refactoring folder from the Oxygen XML Developer plugin installation directory (`[OXYGEN_INSTALL_DIR]/refactoring/`).

Sharing Custom Refactoring Operations

The purpose of Oxygen XML Developer plugin scanning multiple locations for the XML Refactoring operations is to provide more flexibility for developers who want to share the refactoring operations with the other team members. Depending on your particular use case, you can attach the custom refactoring operations to other resources, such as frameworks or projects.

After storing custom operations, you can share them with other users by sharing the resources.

Localizing XML Refactoring Operations

Oxygen XML Developer plugin includes localization support for the XML refactoring operations.

The translation keys for the built-in refactoring operations are located in `[OXYGEN_INSTALL_DIR]/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 possible values for each parameter.

Translated refactoring information uses the following form:

```
${i18n(translation_key)}
```

Oxygen XML Developer plugin scans the following locations to find the `translation.xml` files that are used to load the translation keys:

- A `refactoring/i18n` folder, created inside a directory that is associated to a customized *framework*.
- A `i18n` folder, created inside a directory that is associated to a customized *framework*.
- An `i18n` folder inside any specified folder. In this case, you need to [open the Preferences dialog box](#), go to **XML > XML Refactoring**, and specify the folder in the **Load additional refactoring operations from** text box.
- The `refactoring/i18n` folder from the Oxygen XML Developer plugin installation directory (`[OXYGEN_INSTALL_DIR]/refactoring/i18n`).

Example of a Refactoring Operation Descriptor File with *i18n* Support

```
<?xml version="1.0" encoding="UTF-8"?>

<refactoringOperationDescriptor
  xmlns="http://www.oxygenxml.com/ns/xmlRefactoring" 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

Oxygen XML Developer plugin includes a built-in editor for XSLT stylesheets. This section presents the features of the XSLT editor and how these features can be used. The features of the XSLT editor include:

- **Create new XSLT files and templates** - You can use the built-in new file wizards to [create new XSLT documents or templates](#).
- **Open and Edit XSLT files** - XSLT files can be opened and edited in the source editor (**Text mode**).
- **Validation** - Presents validation errors in XSLT files.
- **Content completion** - Offers proposals for properties and the values that are available for each property.
- **Syntax highlighting** - The syntax highlighting in Oxygen XML Developer plugin makes XSLT files more readable.

To watch our video demonstration about basic XSLT editing and transformation scenarios in Oxygen XML Developer plugin, go to https://www.oxygenxml.com/demo/XSL_Editing.html.

Editing XSLT Stylesheets in the Master Files Context

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

You can set a main XSLT stylesheet either using the [master files support from the Navigator view](#), or using a validation scenario.

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

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

- Correct validation of a module in the context of a larger stylesheet structure.
- **Content Completion Assistant** displays all components valid in the current context.
- The **Outline** displays the components collected from the entire stylesheet structure.

To watch our video demonstration about editing XSLT stylesheets in the master files context, go to <https://www.oxygenxml.com/demo/MasterFilesSupport.html>.

Related information

[XSLT Resource Hierarchy/Dependencies View](#) on page 256

[XSLT Component Dependencies View](#) on page 258

Validating XSLT Stylesheets

Oxygen XML Developer plugin performs the validation of XSLT documents with the help of an XSLT processor *that you can configure in the preferences pages* according to the XSLT version. For XSLT 1.0, the options are: Xalan, Saxon 6.5.5, Saxon 9.6.0.7 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 Developer 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 Developer 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.

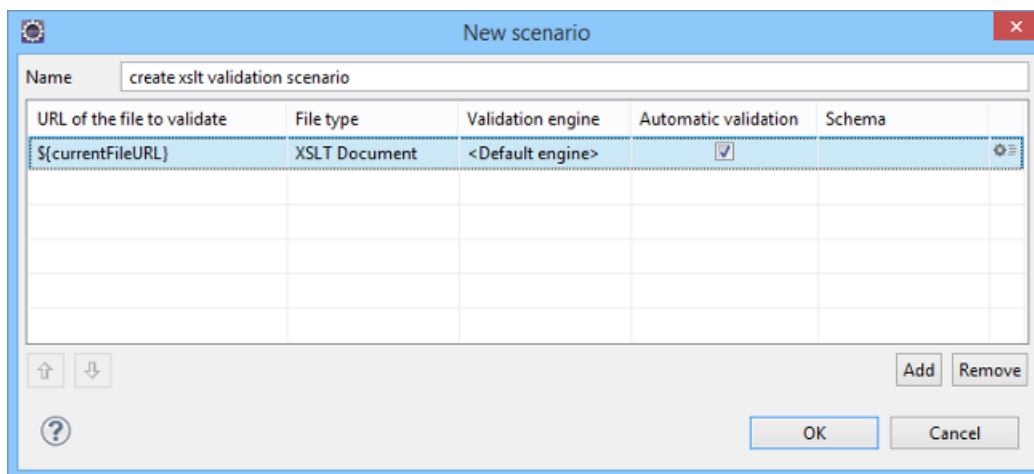


Figure 113: Add / Edit a Validation Unit

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 Developer plugin preferences. After a custom validation engine is *properly configured*, it can be applied on the current document by selecting it from the list of custom validation engines in the ☒ **Validation** toolbar drop-down 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 Developer plugin (Windows edition). It is associated to the XSL Editor type in [Preferences page](#).
- **MSXML.NET** - included in Oxygen XML Developer plugin (Windows edition). It is associated to the XSL Editor type in [Preferences page](#).

Content Completion in XSLT Stylesheets

The list of proposals offered by the **Content Completion Assistant** are context-sensitive and includes proposals that 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.

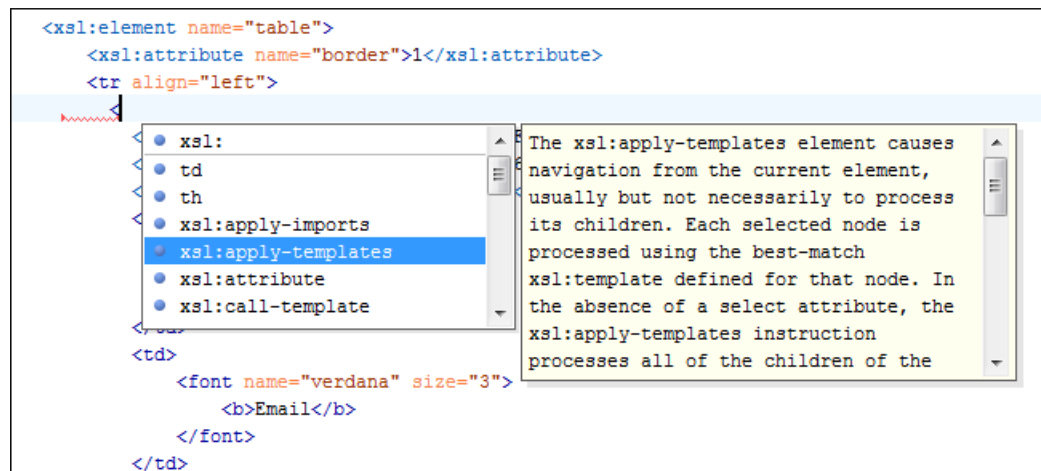


Figure 114: XSLT Content Completion Assistant

The feature is activated in **Text** mode in the following situations:

- After you press the `<` character when inserting an element, it is automatically activated after a short delay. You can adjust the activation delay with the *Activation delay of the proposals window (ms)* option from the **Content Completion** preferences page.

- After typing a partial element or attribute name, you can activate it by pressing **Ctrl + Space (Command + Space on OS X)** or **Alt + ForwardSlash (Command + Alt + ForwardSlash on OS X)**. If there is only one valid proposal at the current location, it is inserted without displaying the list of proposals.

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 included 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 list of content completion assistance proposals 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 assistance window to speed up the insertion into the document of prefixed elements.

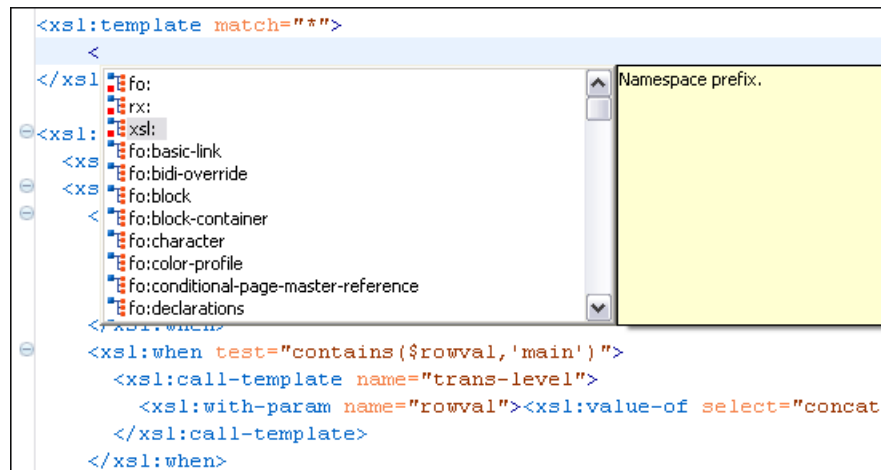


Figure 115: Namespace Prefixes in the Content Completion Assistant

For the common namespaces such as XSL namespace (<http://www.w3.org/1999/XSL/Transform>), XML Schema namespace (<http://www.w3.org/2001/XMLSchema>), or Saxon namespace (<http://icl.com/saxon> for version 1.0, <http://saxon.sf.net/> for version 2.0 / 3.0), Oxygen XML Developer plugin provides an easy mode to declare them by proposing a prefix for these namespaces.

Content Completion in XPath Expressions

In XSLT stylesheets, the **Content Completion Assistant** provides *all the features available in the XML editor* and also adds some enhancements. In XPath expressions used in attributes of XSLT stylesheets (elements such as `match`, `select`, and `test`), the **Content Completion Assistant** offers the names of XPath and XSLT functions, 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 proposals presented in the **Content Completion Assistant** 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 subordinatess="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 pop-up 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.

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

The child axis contains the children of the context node.
Sample: `child::para` selects the `para` element children of the context node.

<http://www.w3.org/TR/1999/REC-xslt-19991116>
W3C Recommendation 16 November 1999
Copyright © [16 November 1999] World Wide Web Consortium,
(Massachusetts Institute of Technology, European Research
Consortium for Informatics and Mathematics, Keio University).
All Rights Reserved. <http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231>

Figure 116: 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:

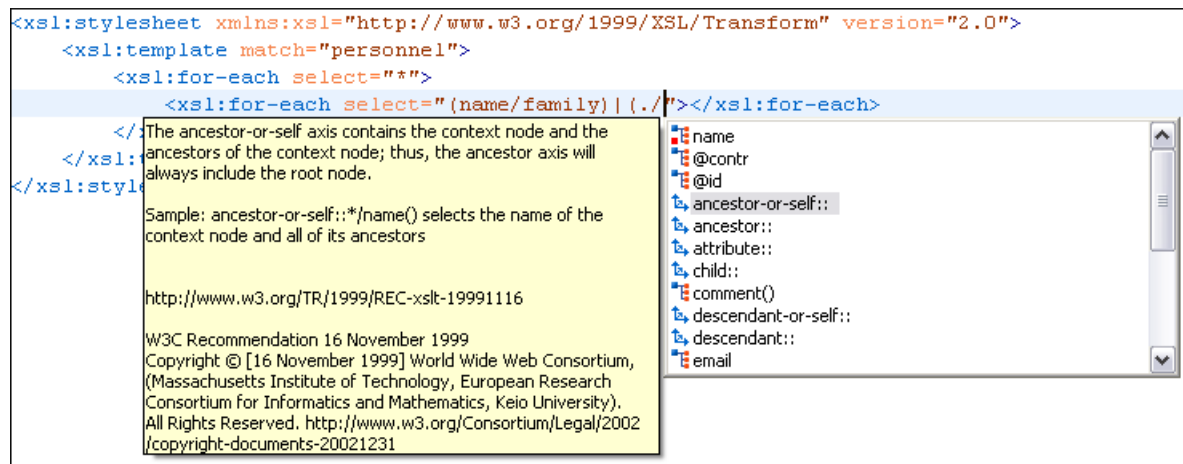


Figure 117: Content Completion in the select Attribute

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

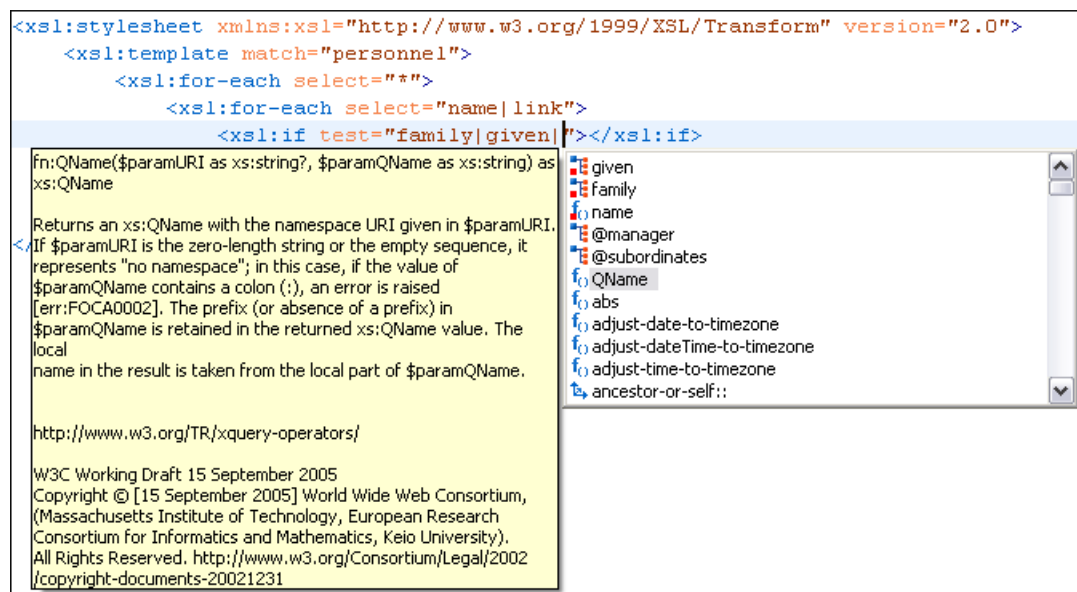


Figure 118: Content Completion in the test Attribute

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



Figure 119: 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.

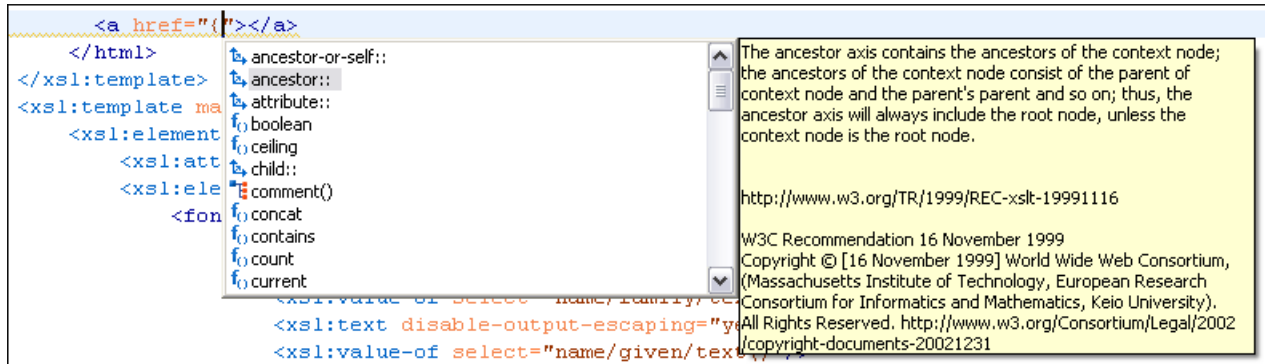


Figure 120: Content Completion in Attribute Value Templates

The time delay (*configured in the [Content Completion preferences page](#)*) for all content completion assistance windows is also applied for the content completion in XPath expressions.

Tooltip Helper for the XPath Functions Arguments

When editing the arguments of an XPath/XSLT function, Oxygen XML Developer plugin tracks the current entered argument by displaying a tooltip containing the function signature. The currently edited argument is highlighted with a bolder font.

When moving the 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 Developer plugin identifies it as the first argument of the *concat* function. The tooltip shows in bold font the following information about the first argument:

- Its name is *\$arg1*.
- Its type is *xdt:anyAtomicType*.
- It is optional (note the ? sign after the argument type).

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

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

Figure 121: 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:

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

Figure 122: XPath Tooltip Helper - Identify the abs Function's Argument

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

```

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

```

Figure 123: XPath Tooltip Helper - Identify the concat Function's Second Argument

The tooltip helper is also available in the [XPath Builder view](#).

Syntax Highlighting in XSLT

Oxygen XML Developer plugin supports syntax highlighting in XSLT documents to improve the readability of the content and reduce the time it takes to internalize the semantics of the structured content.

To customize the colors or styles used for the syntax highlighting colors for XSLT files, follow these steps:

1. [Open the Preferences dialog box](#).
2. Go to **Editor > Syntax Highlight**.
3. Select and expand the **XML** section in the top pane.
4. Select the component you want to change and customize the colors or styles using the selectors to the right of the pane.
5. Select the **XSL** tab in the **Preview** pane to see the effects of your changes.

Related information

[Customize Syntax Highlight colors](#) on page 67

XSLT Outline View

The **Outline** view for XSLT stylesheets 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](#).

By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

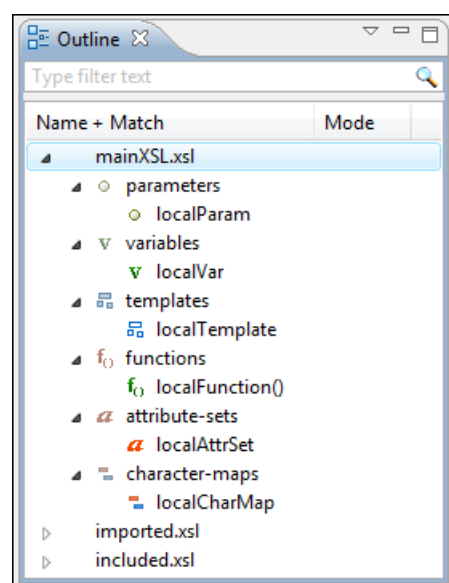


Figure 124: XSLT Outline View

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 (Command + Shift + R on OS X)

Searches all references of the item found at current cursor position in the defined scope, if any. See [Finding XSLT References and Declarations](#) for more details.

Search References in

Searches all references of the item found at current cursor position in the specified scope. See [Finding XSLT References and Declarations](#) for more details.

Component Dependencies

Allows you to see the dependencies for the current selected component. See [Component Dependencies View](#) for more details.

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

- Keyboard arrow keys
- **Enter** key
- **Tab** key
- **Shift-Tab** key combination

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

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

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

If no wildcards are specified, the string to search is used as a partial match.

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 Developer plugin allows you to sort the components of the tree in the **Outline** view.

 **Note:** Sorting groups in the **Outline** view is not supported.

Oxygen XML Developer plugin has a predefined order of the groups in the **Outline** view:

- For location, the names of the files are sorted alphabetically. The main file is the one you are editing and it is located at the top of the list.
- For type, the order is: parameters, variables, templates, functions, set attributes, character-map.

 **Note:** When no grouping is available and the table is not sorted, Oxygen XML Developer plugin sorts the components depending on their order in the document. Oxygen XML Developer plugin also takes into account the name of the file that the components are part of.

XSLT/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 the **XSLT/XQuery Input** view. If the view is not displayed, it can be opened from the **Window > Show View** menu. The tree nodes represent the elements of the documents.

XSLT

If you click a node in the **XSLT/XQuery Input** view, 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.

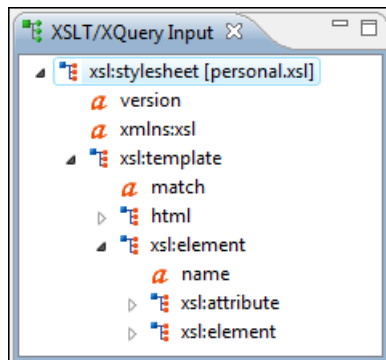


Figure 125: XSLT Input View

XSLT 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:

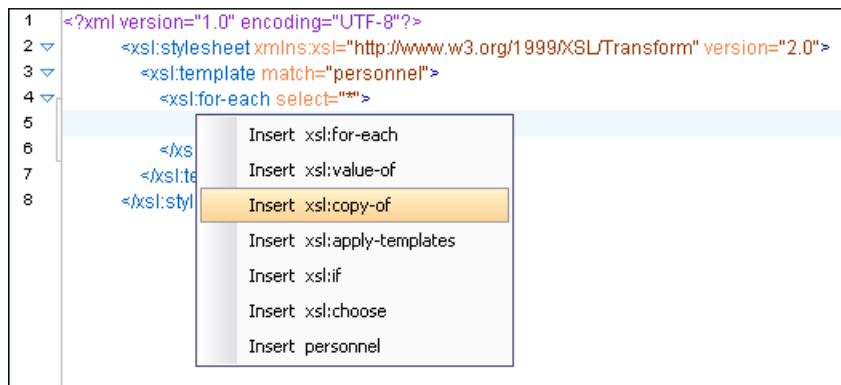


Figure 126: XSLT Input Drag and Drop Pop-up Menu

If you select **Insert xsl:copy-of** (for example), the resulting document will look like this:

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

Figure 127: XSLT Input Drag and Drop Result

XQuery

You can also use the **XSLT/XQuery Input** view to drag and drop a node into the editing area to quickly insert XQuery expressions.

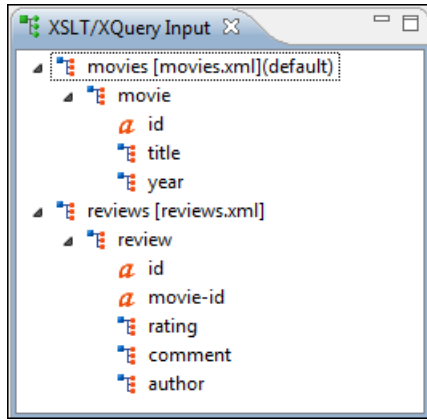


Figure 128: XQuery Input View

XQuery Example:

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>
```

```
<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, the following pop-up menu is displayed:

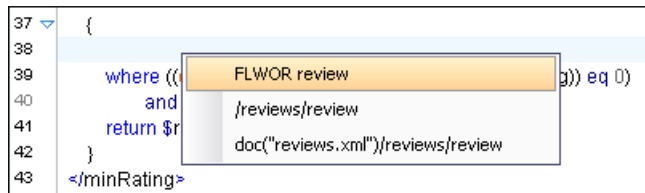


Figure 129: XQuery Input Drag and Drop Pop-up Menu

Select **FLWOR review**, the resulting document will look like this:

```

37 {
38     for $review in doc("reviews.xml")/reviews/review
39     return
40     where ((compare($rev/rating/text(), string($minRating)) eq 0)
41           and ($rev/@movie-id = $movie/@id))
42     return $rev/author
43 }
```

Figure 130: XQuery Input Drag and Drop Result

XSLT Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a stylesheet. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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

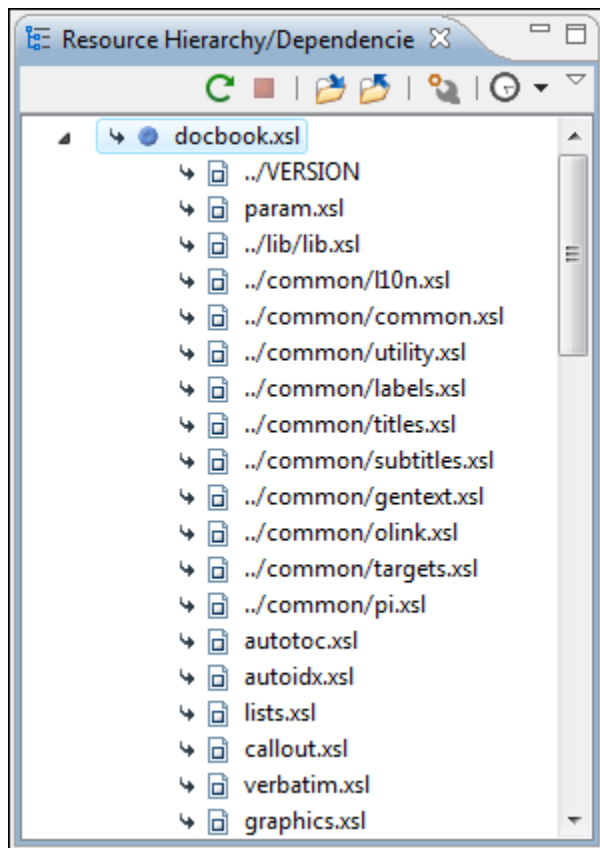


Figure 131: Resource Hierarchy/Dependencies View - Hierarchy for docbook.xsl

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

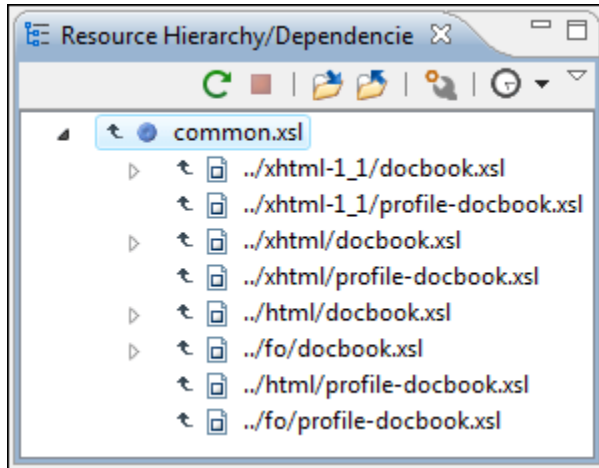


Figure 132: Resource Hierarchy/Dependencies View - Dependencies for common.xsl

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 .

Related information

[Search and Refactor Operations Scope](#) on page 210

Moving/Renaming XSLT Resources

You can 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. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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).

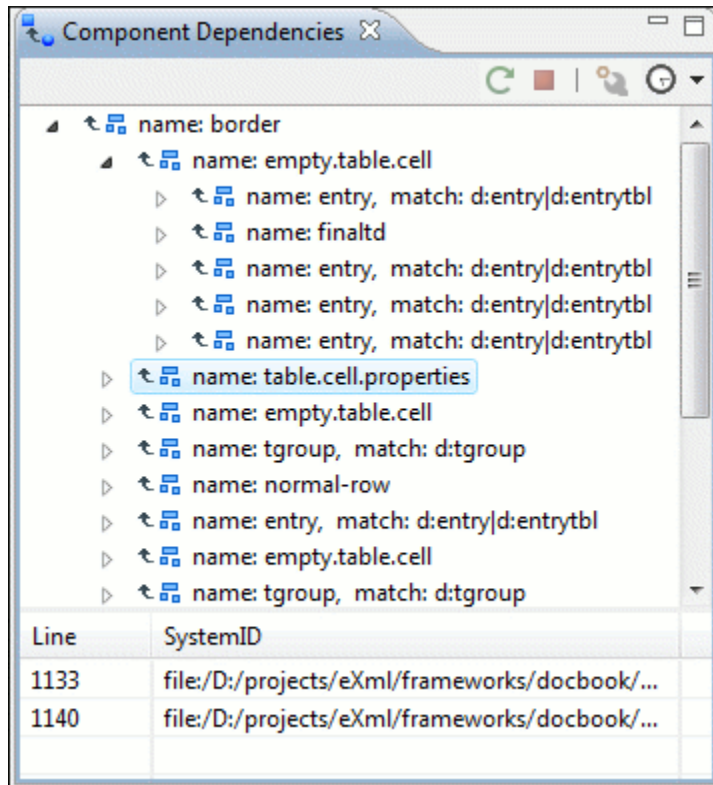


Figure 133: Component Dependencies View - Hierarchy for table.xsl

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

Related information

[Search and Refactor Operations Scope](#) on page 210

Highlight Component Occurrences

When a component (for example variable or named template) is found at current cursor position, Oxygen XML Developer plugin performs a search over the entire document to find the component declaration and all its references. When found, they are highlighted both in the document and in the stripe bar, at the right side of the document.



Note: Oxygen XML Developer plugin also supports occurrences highlight for template modes.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected and a new search is performed. All occurrences are removed when you start to edit the document.

This feature is enabled by default. To configure it, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**. A search can also be triggered with the **Search > Search Occurrences in File (Ctrl + Shift + U (Command + Shift + U on OS X))** contextual menu action. Matches are displayed in separate tabs of the **Results** view.

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.

Related information

[Search and Refactor Operations Scope](#) on page 210

XSLT Stylesheet Component Documentation Support

Oxygen XML Developer plugin offers built-in support for documenting XSLT stylesheets. If the expanded *QName* of the element has a non-null namespace URI, the `xsl:stylesheet` element may contain any element not from the XSLT namespace. Such elements are referenced as user-defined data elements. Such elements can contain the documentation for the stylesheet and its elements (top-level elements whose names are in the XSLT namespace). Oxygen XML Developer plugin offers its own XML schema that defines such documentation elements. The schema is named `stylesheet_documentation.xsd` and can be found in `[OXYGEN_INSTALL_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.

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 (such as a template or function), a documentation block is generated for that element only.

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

```
<xd:doc>
  <xd:desc>
    <xd:p>Search inside parameter <xd:i>string</xd:i>
      for the last occurrence of parameter
    <xd:i>searched</xd:i>. The substring starting from the 0 position
      to the identified last occurrence will be returned.
    <xd:ref name="f:substring-after-last" type="function"
      xmlns:f="http://www.oxygenxml.com/doc/xsl/functions">See also
    </xd:ref>
  </xd: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>
```

Related information

[Generating Documentation for an XSLT Stylesheet](#) on page 261

Generating Documentation for an XSLT Stylesheet

You can use Oxygen XML Developer plugin to generate detailed documentation in HTML format for the elements (top-level elements whose names are in the XSLT namespace) of an XSLT stylesheet. You can 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.

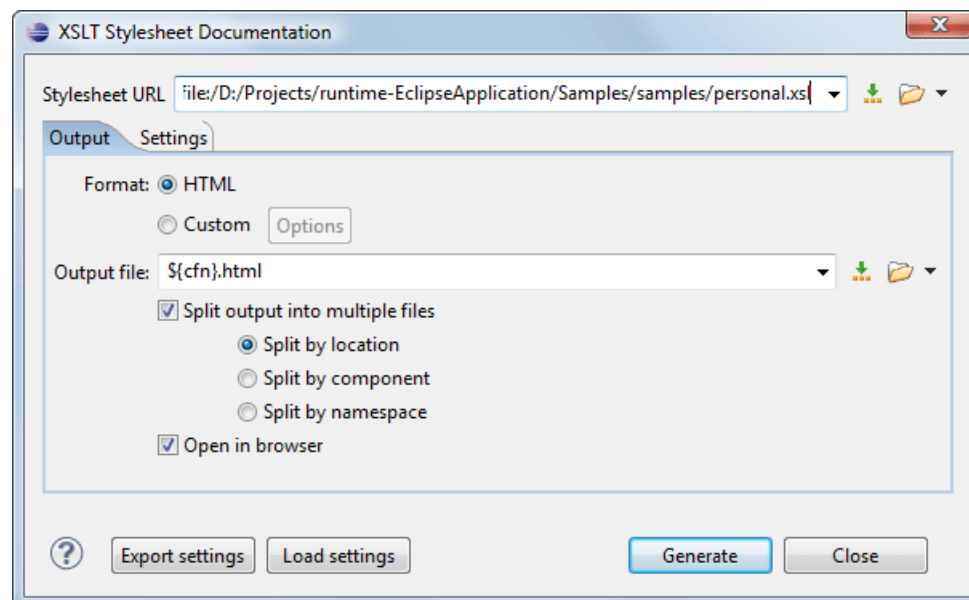


Figure 134: XSLT Stylesheet Documentation Dialog Box

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.

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, choosing another split criterion may generate smaller output files, providing 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, go to **Window > Preferences > General > Web Browser** and specify it there. This will take precedence over the default system application settings.

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.

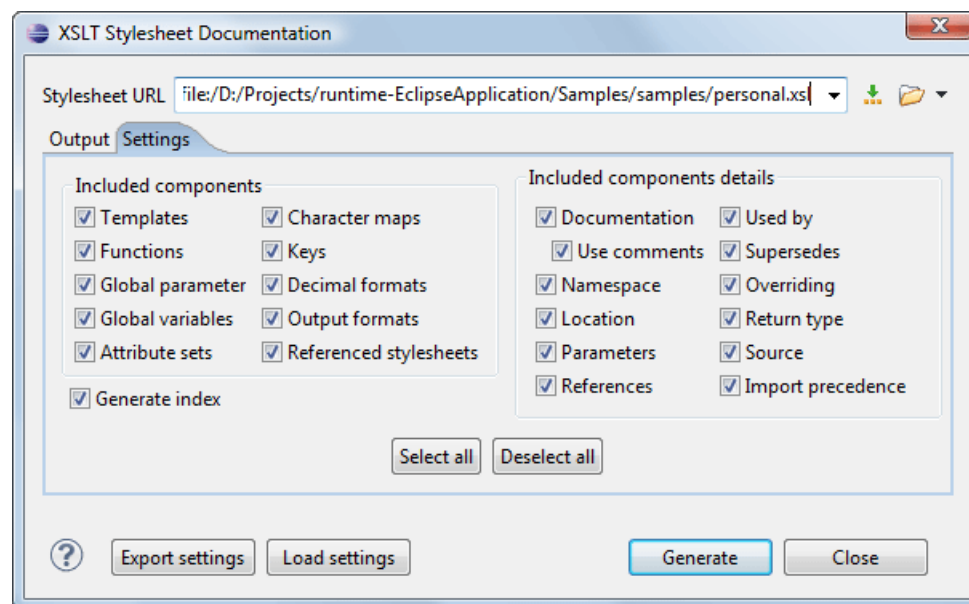


Figure 135: Settings Tab of the XSLT Stylesheet Documentation Dialog Box

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 Developer plugin built-in XSLT documentation schema.

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

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

- **Use comments** - Controls whether 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.

Related information

XSLT Stylesheet Component Documentation Support on page 260

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.

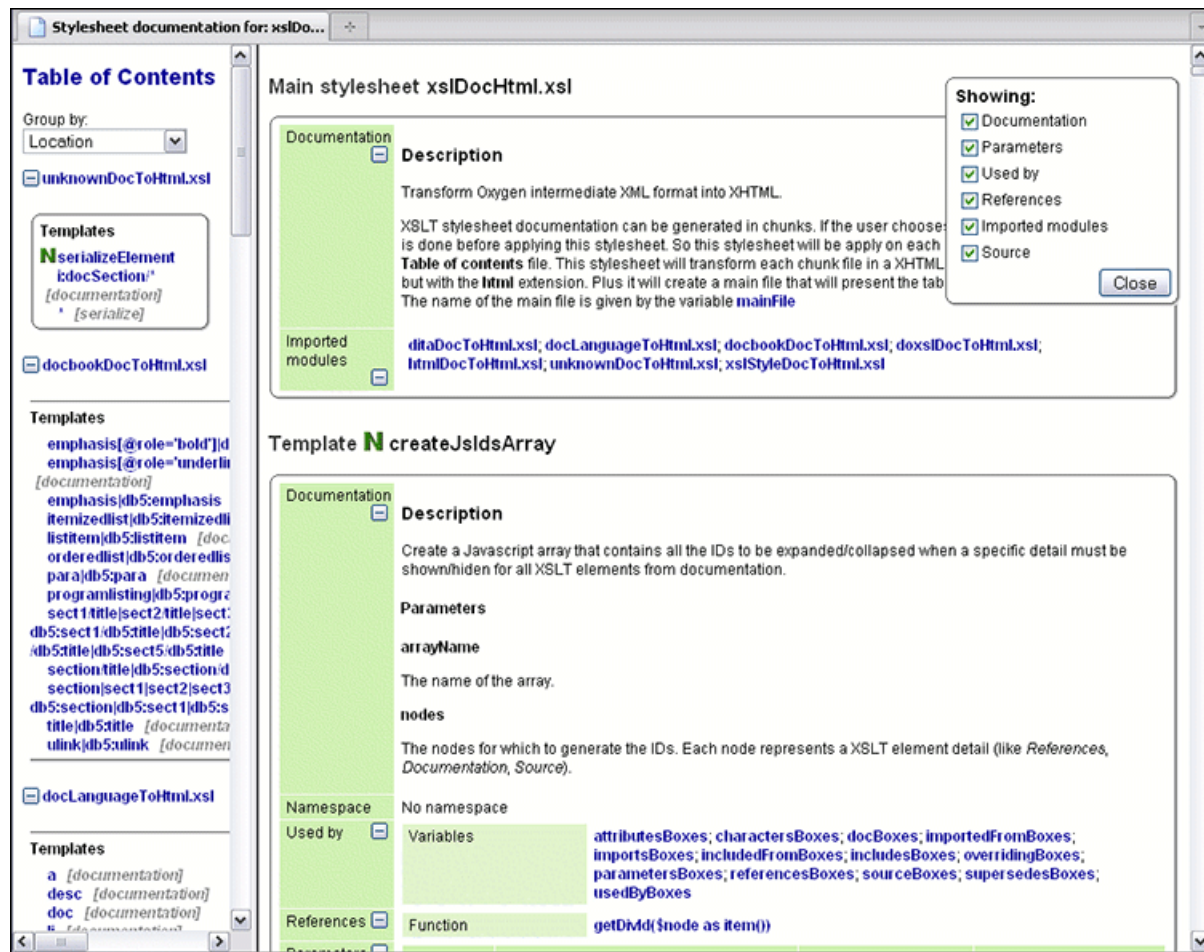


Figure 136: XSLT Stylesheet Documentation Example

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.

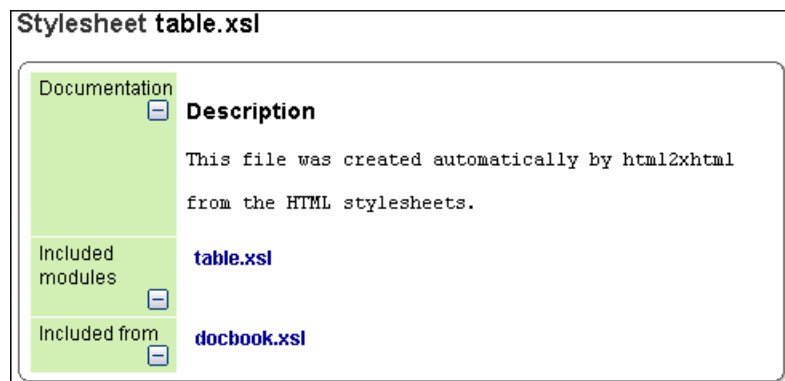


Figure 137: Information About an XSLT Stylesheet

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.

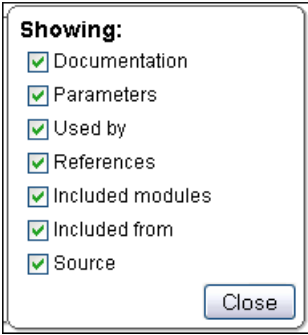


Figure 138: Showing Options

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).

Function **func:substring-before-last**

Documentation 

Namespace

Type

Used by 

References 

Parameters 

Import precedence

Source 

Description

Get the substring before the last occurrence of the given substring

Parameters

string
The string in which to search

searched
The string to search

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

http://www.oxygenxml.com/doc/xsl/functions

xs:string

Template	Nindex
Function	func:substring-before-last(\$string as item(), \$searched as item())
Variable	indexFile

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

QName	Namespace
searched	No namespace
string	No namespace

7

```
<xsl:function as="xs:string" name="func:substring-before-last">
  <xsl:param name="string"/>
  <xsl:param name="searched"/>
  <xsl:variable name="toReturn">
    <xsl:choose>
      <xsl:when test="contains($string, $searched)">
        <xsl:variable name="before" select="substring-before($string, $searched)"/>
        <xsl:variable name="rec" select="func:substring-before-last(substring-after($string, $searched), $searched)"/>
        <xsl:concat($before, $rec)
      </xsl:when>
      <xsl:otherwise>
        $string
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  $toReturn
</xsl:function>
```

Figure 139: 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_INSTALL_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_INSTALL_DIR]/frameworks/stylesheet_documentation/xsl`.

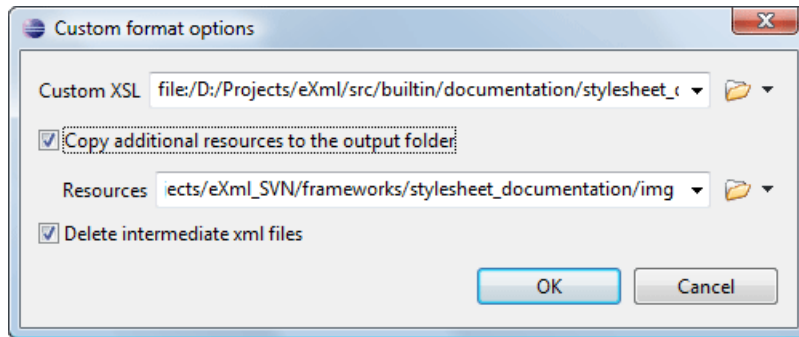


Figure 140: Custom Format Options Dialog Box

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

The command-line parameter of the script is the relative path to the exported XML settings file. The files 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>
      </xsdDocumentationOptions>
    </entry>
  </map>
</serialized>
```

```

<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>

```

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 (💡) placed at the current line in the stripe on the left side of the editor. Also, you can invoke the quick assist menu by using the **Ctrl + 1** (**Meta 1** on Mac OS X) keyboard shortcuts.

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.

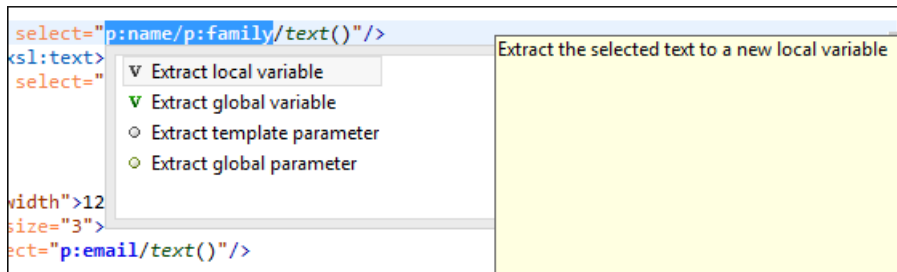


Figure 141: XSLT Quick Assist Support - Refactoring Actions

- 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.

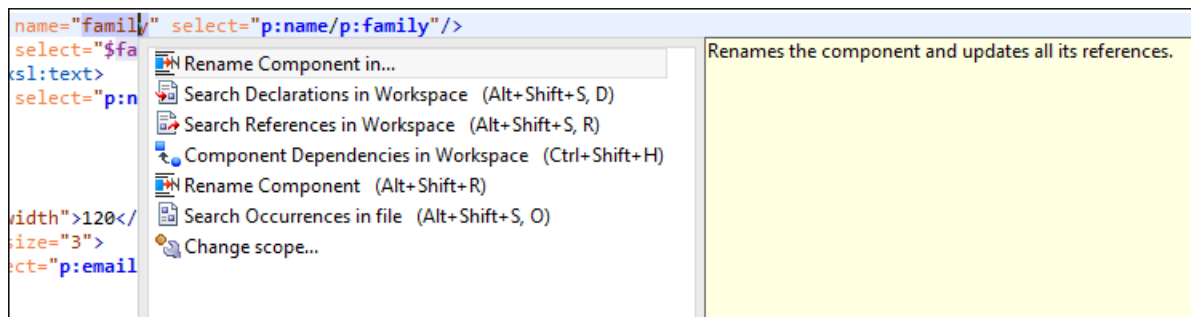


Figure 142: XSLT Quick Assist Support - Component Actions

Related information

[Component Dependencies View](#) on page 258

[XSLT Hierarchy View](#) on page 256

[XSLT Refactoring Actions](#) on page 270

[Search and Refactor Operations Scope](#) on page 210

XSLT Quick Fix Support

The Oxygen XML Developer 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 pop-up 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 Developer 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 (Command + 1 on OS X)** on your keyboard.



Note: The quick fixes are available only when validating an XSLT file with Saxon HE/PE/EE.

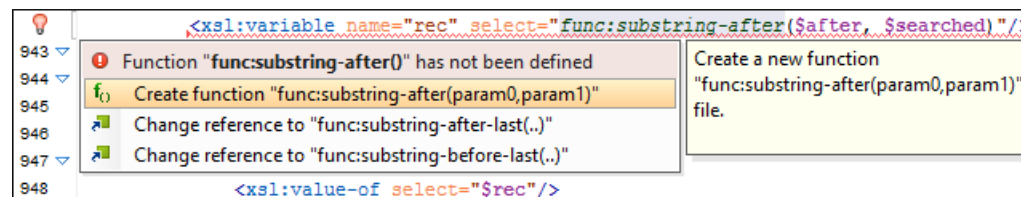


Figure 143: Example of an Undefined XSLT Functions Quick Fix

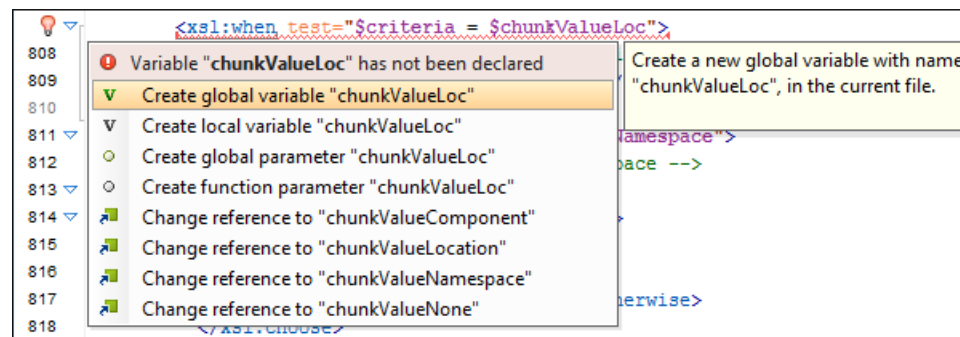


Figure 144: Example of an Undeclared XSLT Variables/Parameters Quick Fix

Oxygen XML Developer 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.

XSLT Refactoring Actions

Oxygen XML Developer plugin offers a set of actions that allow you to change the structure of an XSLT stylesheet without changing the results of running it in an XSLT transformation. Depending on the selected text, the following XSLT refactoring actions are available from the **Refactoring** submenu of the contextual menu:

-  **Extract template** (Active only when the selection contains well-formed elements) - Extracts the selected XSLT instructions sequence into a new template. It 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: The newly created template is indented and its name is highlighted in the `<xsl:call-template>` element.

- **Extract function** - Extracts the selected XSLT instructions sequence into a new function. It opens a dialog box that allows you to specify the name of the new function. It then moves the selected lines to a newly created XSLT function and inserts a function call in the place of the selected lines.
- **Create local variable** - Creates an XSLT variable, wrapped around the selection. It opens a dialog box that allows you to specify the name of the new variable. It then wraps the selection in the variable and you can reference it at anytime in the code.
- **Move to another stylesheet** (Active only when entire components are selected) - Allows you to move one or more XSLT global components (templates, functions, or parameters) to another stylesheet. It opens a dialog box that allows you to specify where the selected components will be moved to. Follow these steps when using the dialog box:
 1. Choose whether you want to move the selected components to a new stylesheet or an existing one.
 2. If you choose to move the components to an existing one, select the destination stylesheet. Press the **Choose** button to select the destination stylesheet file. Oxygen XML Developer plugin will automatically check if the destination stylesheet is already contained by the hierarchy of the current stylesheet. If it is not contained, choose 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. 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, the following element:

```
<person id="Big{test}Boss"/>
```

is converted to:

```
<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** (Active on a selection made inside an attribute that contains an XPath expression) - Allows you to create a new local variable by extracting the selected XPath expression. After creating the new local variable before the current element, Oxygen XML Developer plugin allows you to edit the name of the variable.

- ▼ **Extract global variable** (Active on a selection made inside an attribute that contains an XPath expression) - Allows you to create a new global variable by extracting the selected XPath expression. After creating the new global variable, Oxygen XML Developer plugin allows you to edit the name of the variable.

 **Note:** Oxygen XML Developer plugin checks if the selected expression depends on local variables or parameters that are not available in the global context where the new variable is created.
- ⊙ **Extract template parameter** (Active on a selection made inside an attribute that contains an XPath expression) - Allows you to create a new template parameter by extracting the selected XPath expression. After creating the new parameter, Oxygen XML Developer plugin allows you to edit the name of the parameter.
- **Extract global parameter** (Active on a selection made inside an attribute that contains an XPath expression) - Allows you to create a new global parameter by extracting the selected XPath expression. After creating the new parameter, Oxygen XML Developer plugin allows you to edit the name of the parameter.

 **Note:** Oxygen XML Developer plugin checks if the selected expression depends on local variables or parameters that are not available in the global context where the new parameter is created.
- Rename Component** - Allows you to rename the current component (in-place). The component and all its references in the document are highlighted with a thin border and the changes you make to the component at the cursor position are updated in real time to all occurrences of the component. To exit the in-place editing, press the **Esc** or **Enter** key on your keyboard.
- 🔍 **Rename Component in** - Opens a 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 to be affected by the action.

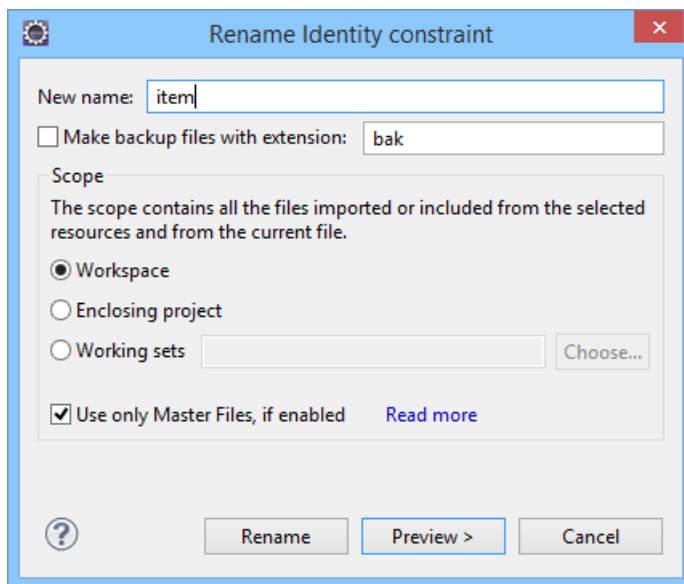


Figure 145: Rename Identity Constraint Dialog Box

 **Note:** Many of these refactoring actions are also proposed by the [Quick Assist support](#).

To watch our video demonstration about XSLT refactoring, go to https://www.oxygenxml.com/demo/XSL_Refactoring.html.

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 **Navigator** view. Oxygen XML Developer 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 Developer plugin uses information from the validation and transformation scenarios associated with the XSL file. From the transformation scenario Oxygen XML Developer plugin uses extensions and properties of Saxon 9.6.0.7, improving the Ant scenario associated with the XSpec document.

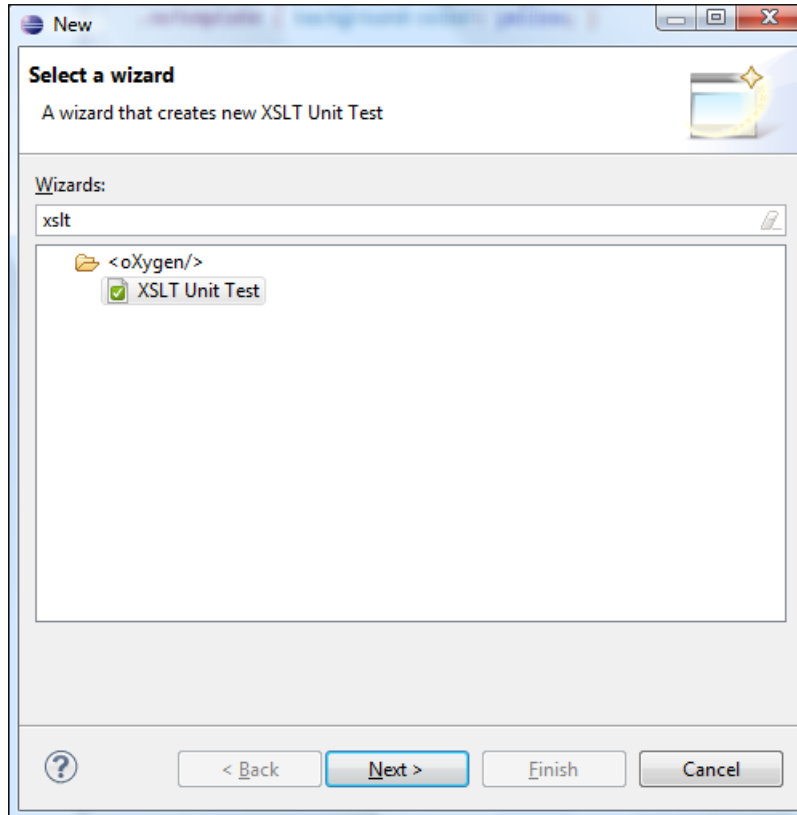


Figure 146: New XSLT Unit Test Wizard

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:call>
</x:scenario>
```

```

    <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 can 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.

Oxygen XML Developer plugin offers support for both XML Schema 1.0 and 1.1 and you can edit XML Schema files in the following editing modes:

- **Text editing mode** - Allows you to edit XML Schema files in a source editing mode.
- **Grid editing mode** - Displays XML Schema files in a structured spreadsheet-like grid.
- **Design editing mode** - Visual schema designer that helps you understand the structure and develop complex schemas.

For information about applying and detecting schemas, see [Associating a Schema to XML Documents](#) on page 211.

Related information

[Associating a Schema to XML Documents](#) on page 211

Design Editing Mode (XML Schema Diagram Editor)

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

Oxygen XML Developer plugin provides a simple and expressive 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.

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 another position and from one schema to another using the **Cut/Copy** and **Paste/Paste as Reference** actions.
- Go to the definition of an element or attribute with the **Show Definition** action.
- You can expand and see the contents of the imports/includes/redefines in the diagram. In order to edit components from other schemas the schema for each component will be opened as a separate file in Oxygen XML Developer plugin.

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

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

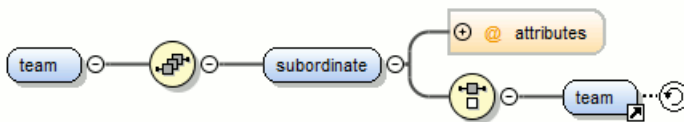


Figure 148: Recursive Reference

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, various 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 that 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 that was not yet mapped, you are prompted to add prefix mappings for the inserted component namespace in the current edited schema.

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

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

Related information

[Searching and Refactoring Actions in XML Schemas](#) on page 313

[Component Dependencies View for XML Schema](#) on page 311

[XML Schema Resource Hierarchy / Dependencies View](#) on page 309

[Generating Sample XML Files](#) on page 315

[Schema Design Preferences](#) on page 58

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 **X** **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 such as schemas or redefines.
- **Specify an ID for the component annotation.** An optional identifier for the annotation.

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 that 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.

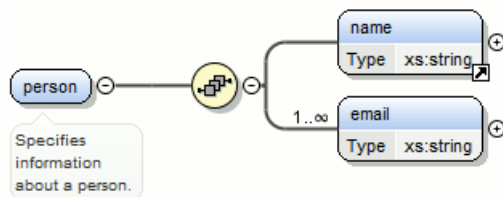


Figure 149: Extracting a Global Element

If you use the **Extract Global Element** action on a `name` element, the result is:

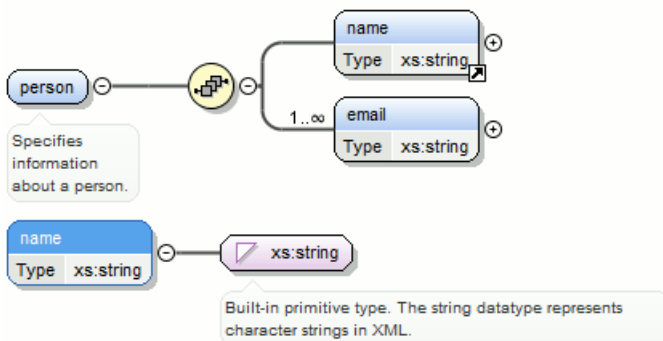


Figure 150: Extracting a Global Element on a name Element

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.

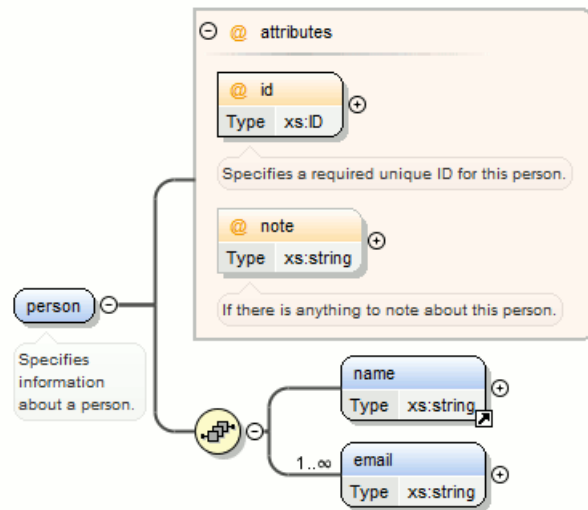


Figure 151: Extracting a Global Attribute

If you use the **Extract Global Attribute** action on a **note** attribute, the result is:

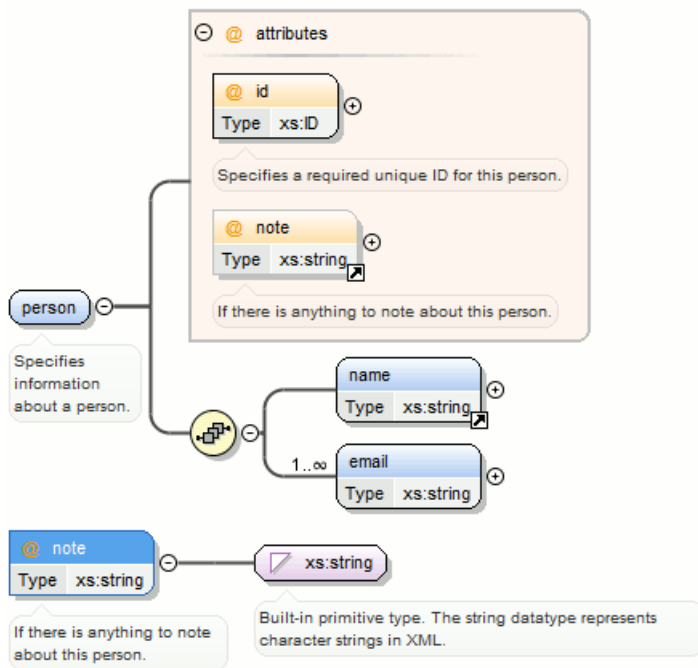
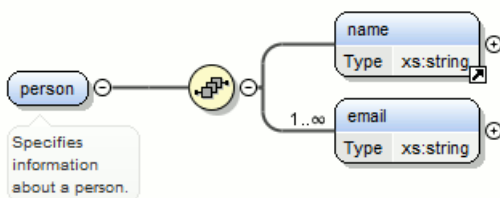


Figure 152: Extracting a Global Attribute on a note Attribute

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 153: Extracting a Global Group

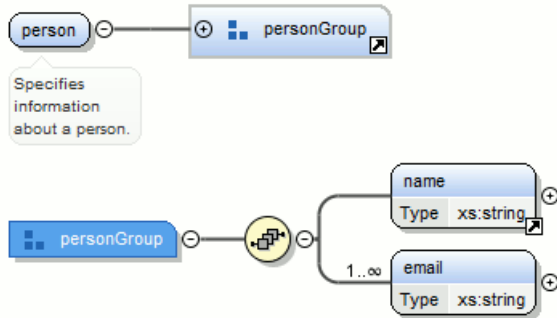


Figure 154: Extracting a Global Group on a sequence Element

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.

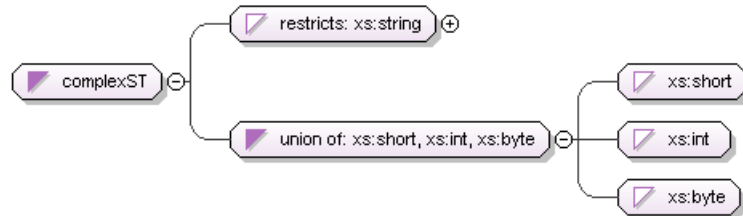


Figure 155: Extracting a Global Simple Type

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

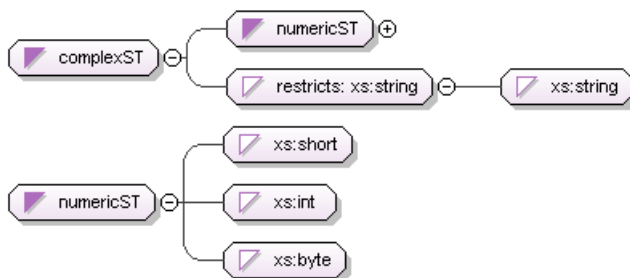


Figure 156: Extracting a Global Simple Type on a union Component

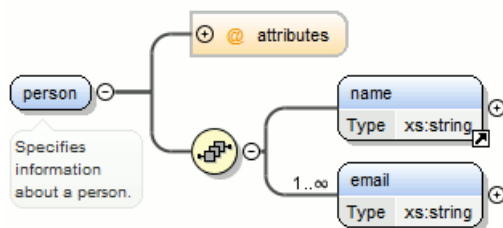


Figure 157: Extracting a Global Complex Type

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

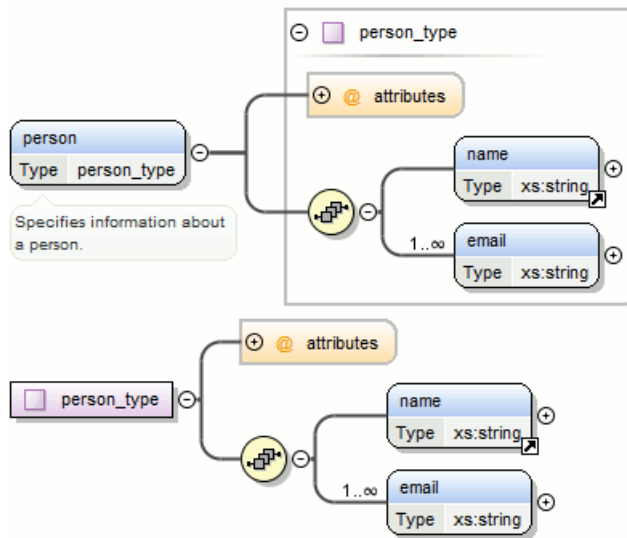


Figure 158: Extracting a Global Complex Type on a person Element



Rename Component in

Rename the selected component.



Cut Ctrl + X (Command + X on OS X)

Cut the selected component(s).



Copy Ctrl + C (Command + C on OS X)

Copy the selected component(s).



Copy XPath

This action copies an XPath expression that identifies the selected element or attribute in an instance XML document of the edited schema and places it in the clipboard.



Paste Ctrl + V (Command + V on OS X)

Paste the component(s) from the clipboard as children of the selected component.

Paste as Reference

Create references to the copied component(s). If not possible a warning message is displayed.

Remove (Delete)

Remove the selected component(s).

Override component

Copies the overridden component in the current XML Schema. This option is available for *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).

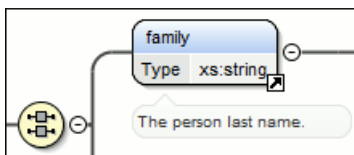


Figure 159: Example: Required Component

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

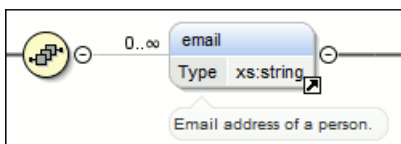


Figure 160: Example: Optional Component

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

xs:schema

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

Figure 161: The xs:schema Component

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.

Table 4: xs:schema Properties

Property Name	Description	Possible Values
Target Namespace	The schema target namespace.	Any URI
Element Form Default	Determining whether or not local element declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Attribute Form Default	Determining whether or not local attribute declarations will be namespace-qualified by default.	qualified, unqualified, [Empty]. Default value is unqualified.
Block Default	Default value of the <code>block</code> attribute of <code>xs:element</code> and <code>xs:complexType</code> .	#all, extension, restriction, substitution, restriction extension, restriction substitution, extension substitution, restriction extension substitution, [Empty].
Final Default	Default value of the <code>final</code> attribute of <code>xs:element</code> and <code>xs:complexType</code> .	#all, restriction, extension, restriction extension, [Empty].
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



Figure 162: The xs:element Component

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.

Table 5: 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 that 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 that 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 that this element belongs to.	All declared elements. For XML Schema 1.1 this property supports multiple values.	For global and reference elements
Abstract	Controls whether or not 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" (belongs to the target namespace) or "unqualified" (doesn't belong to any namespace).	unqualified/qualified	Only for local elements
Nilable	When this attribute is set to true, the element can be declared as nil using an <code>xsi:nil</code> attribute in the instance documents.	true/false	For global elements and element references
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 if the element can be subject to a type or substitution group substitution. '#all' blocks any substitution, 'substitution' blocks any substitution through substitution groups and 'extension'/'restriction' block any substitution (both through <code>xsi:type</code> and substitution groups) by elements or types, derived respectively by extension or restriction from the type of the element. Its default value is defined by the <code>blockDefault</code> attribute of the parent <code>xs:schema</code> .	#all, restriction, extension,substitution, extension restriction, extension substitution, restriction substitution, restriction extension substitution	For global elements and element references

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 <code>finalDefault</code> attribute of the parent <code>xs:schema</code> .	#all, restriction, extension, restriction extension, [Empty]	For global elements and element references
ID	The component id.	Any id	For all elements.
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

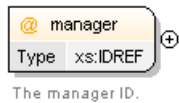


Figure 163: The xs:attribute Component

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.

Table 6: 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.

Property Name	Description	Possible Value	Mentions
Default	Default value. When specified, an attribute is added by the schema processor (if it is missing from the instance XML document) and it is given this value. The default and fixed attributes are mutually exclusive.	Any string	For all local or global attributes. For references the value is from the referenced attribute.
Fixed	When specified, the value of the attribute is fixed and must be equal to this value. The default and fixed attributes are mutually exclusive.	Any string	For all local or global attributes. For references the value is from the 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 whether or not the attribute is qualified (must have a namespace prefix in the instance XML document). The default value for this attribute is specified by the <code>attributeFormDefault</code> attribute of the <code>xs:schema</code> document element.	unqualified/qualified	For local attributes.
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 a 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



Figure 164: The xs:attributeGroup Component

Defines an attribute group to be used in complex type definitions. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-attributeGroup>.

Table 7: 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

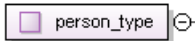


Figure 165: The xs:complexType Component

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 info: set contributions for elements.
- Limiting the ability to derive additional types from a given complex type.
- Controlling the permission to substitute, in an instance, elements of a derived type for elements declared in a content model to be of a given complex type.

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

Table 8: 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.

Property Name	Description	Possible Values	Mentions
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 that extend/restrict a base type. It is automatically detected.
Content Mixed	Specifies if the complex content model will be mixed.	true/false	For complex contents.
Mixed	Specifies if the complex type content model will be mixed.	true/false	For global and anonymous complex types.
Abstract	When set to <code>true</code> , this complex type cannot be used directly in the instance documents and needs to be substituted using an <code>xsi:type</code> attribute.	true/false	For global and anonymous complex types.
Block	Controls if a substitution (either through a <code>xsi:type</code> or substitution groups) can be performed for a complex type, which is an extension or a restriction of the current complex type. This attribute can only block such substitutions (it cannot "unblock" them), which can also be blocked in the element definition. The default value is defined by the <code>blockDefault</code> attribute of <code>xs:schema</code> .	all, extension, restriction, extension restriction, [Empty]	For global complex types.
Final	Controls whether the complex type can be further derived by extension or restriction to create new complex types.	all, extension, restriction, extension restriction, [Empty]	For global complex types.
Default Attributes Apply	The schema element can carry a <code>defaultAttributes</code> attribute, which identifies an attribute group. Each <code>complexType</code> defined in the schema document then automatically includes that attribute group, unless this is overridden by the <code>defaultAttributesApply</code> attribute on the <code>complexType</code> element.	true/false	This property is available only for XML Schema 1.1.
ID	The component id.	Any id	For all complex types.

Property Name	Description	Possible Values	Mentions
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



Figure 166: The xs:simpleType Component

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 that is a base type to another type will be rendered with yellow background.

Table 9: 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).

Name	Description	Possible Values	Scope
Member Types	Category for grouping union members.	Not editable property.	For global and anonymous simple types with the derivation method set to union.
Member	A simple type definition component. Required if derivation method is set to union.	All global simple types and built-in simple types(from schema for schema). In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for easily create anonymous simple types.	For global and anonymous simple types with the derivation method set to union. Deriving a simple datatype by union merges the lexical spaces of several simple datatypes (called member types) to create a new simple datatype. The member types can be defined either by reference (through the memberTypes attribute) or embedded as simple datatype local definitions in the xs:union element. Both styles can be mixed.
Final	Blocks any further derivations of this datatype (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.

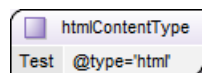


Figure 167: The xs:alternative Component

Table 10: 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].

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	Specifies the type of XML schema component.	Not editable property.
System ID	Points to the document location of the schema.	Not editable property.

xs:group



Figure 168: The xs : group Component

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.

Table 11: 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



Figure 169: The xs:include Component

Adds multiple schemas with the same target namespace to a document. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-include>.

Table 12: 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



Figure 170: The xs:import Component

Adds multiple schemas with a different target namespace to a document. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-import>.

Table 13: 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



Figure 171: The xs:redefine Component

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>.

Table 14: 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



Figure 172: The xs:override Component

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>.

Table 15: xs:override Properties

Property Name	Description	Possible Values
Schema Location	Redefine schema location.	Any URI
ID	Redefine ID	Any ID

xs:notation



Figure 173: The xs:notation Component

Describes the format of non-XML data within an XML document. See more info at <http://www.w3.org/TR/xmlschema11-1/#element-notation>.

Table 16: xs:notation Properties

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

xs:sequence / xs:choice / xs:all



Figure 174: xs:sequence

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 175: xs:choice

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 176: xs:all

`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.

Table 17: `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`



Figure 177: The `xs:any` Component

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.

Table 18: `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

Property Name	Description	Possible Values
Min Occurs	Minimum occurrences of any	A numeric positive value. Default is 1.
Max Occurs	Maximum occurrences of any	A numeric positive value. Default is 1.
ID	The component id.	Any ID.
Component	The name of the edited component.	Not editable property.
System ID	The component system id.	Not editable property.

xs:anyAttribute

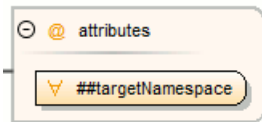


Figure 178: The xs:anyAttribute Component

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>.

Table 19: xs:anyAttribute Properties

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

xs:unique

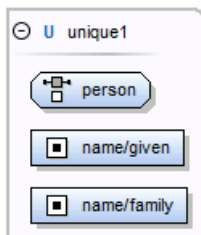


Figure 179: The xs:unique Component

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>.

Table 20: 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

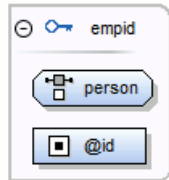


Figure 180: The xs:key Component

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>.

Table 21: 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

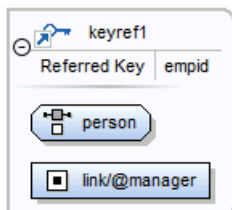


Figure 181: The xs:keyRef Component

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.

Table 22: 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.

Property Name	Description	Possible Values
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



Figure 182: The xs:selector Component

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>.

Table 23: xs:selector Properties

Property Name	Description	Possible Values
XPath	Relative XPath expression identifying the element that the constraint applies to.	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



Figure 183: The xs:field Component

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>.

Table 24: xs:field Properties

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

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.

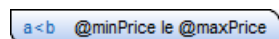


Figure 184: The xs:assert Component

Table 25: 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



Figure 185: The xs:openContent Component

The `openContent` element enables instance documents to contain extension elements to be inserted amongst 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>.

Table 26: 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.
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

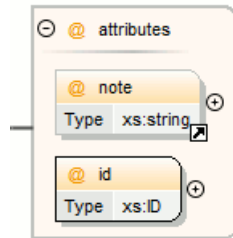


Figure 186: Attributes Construct

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

Table 27: 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

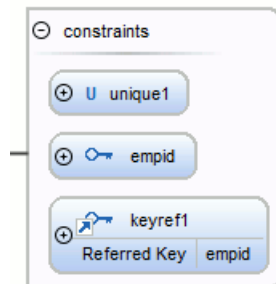


Figure 187: Constraints Construct

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

Table 28: constraints 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

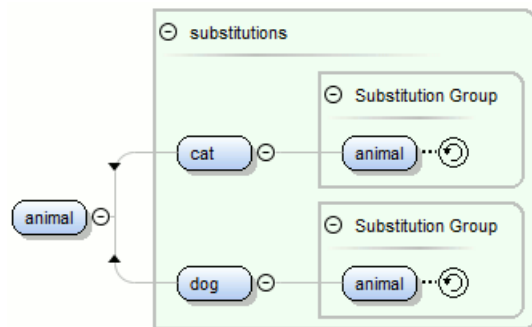


Figure 188: Substitutions Construct

Groups all elements that can substitute the current element.

Table 29: substitutions 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 Developer plugin [XML documents validation](#) capability.

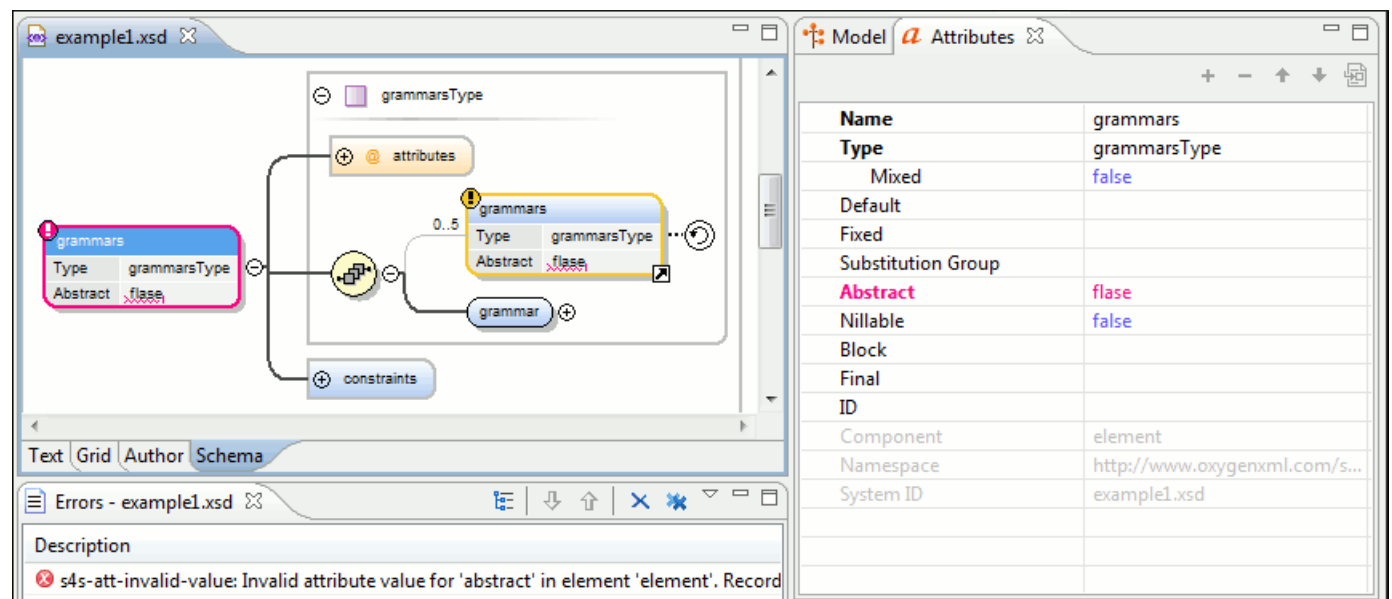


Figure 189: XML Schema Validation

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 that 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 an 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. You can also do this 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.

XML Schema 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](#). If the view is not displayed, it can be opened from the **Window > Show View** menu.

Figure 190: Palette View

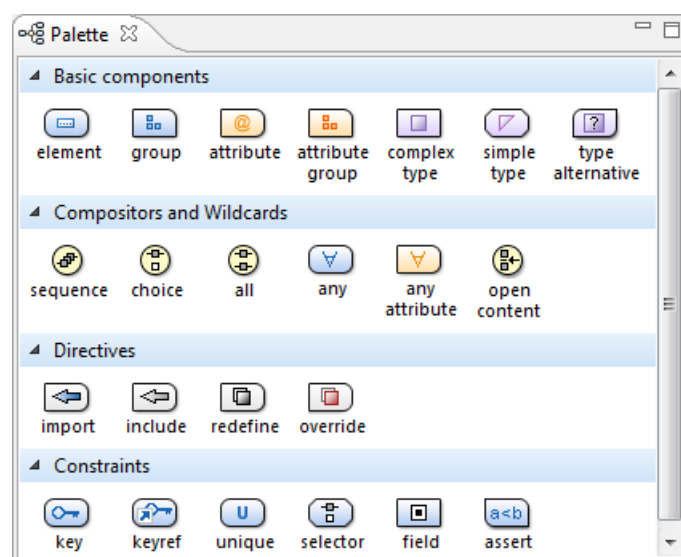


Figure 191: Palette View

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

https://www.oxygenxml.com/demo/Schema_Palette.html and https://www.oxygenxml.com/demo/Developing_XML_Schemas.html respectively.

XML Schema Facets View

The **Facets** view for XML Schemas presents the facets for the selected component, if available. If the view is not displayed, it can be opened from the **Window > Show View** menu.

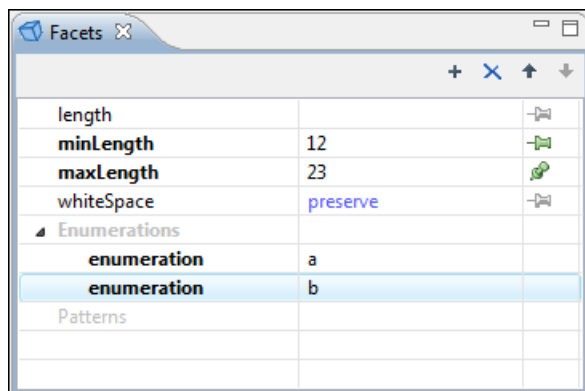


Figure 192: Facets View

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 that 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.

Editing XML Schema in Text Editing Mode

The Oxygen XML Developer plugin **Text** editing mode can be used for editing XML Schema in a source editing 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* checkbox from the [Diagram](#) preferences page.

Editing XML Schema in the Master Files Context

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

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

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

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

- Correct validation of a module in the context of a larger schema structure.
- **Content Completion Assistant** displays all the referable components valid in the current context. This includes components defined in modules other than the currently edited one.
- The **Outline** displays the components collected from the entire schema structure.

Validating XML Schema Documents

By default, XML Schema files are 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 an XML Schema document manually, select the  **Validate** action from the  **Validation** toolbar drop-down menu or the **XML** menu. When Oxygen XML Developer plugin validates an XML Schema file, 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.

Related concepts

[Validation Scenario](#) on page 198

Related information

[Validating XML Documents Against a Schema](#) on page 194

[Associating a Schema to XML Documents](#) on page 211

[Presenting Validation Errors in Text Mode](#) on page 125

Content Completion in XML Schema

The intelligent **Content Completion Assistant** available in Oxygen XML Developer plugin enables rapid, in-line identification and insertion of elements, attributes and attribute values valid in the editing context. All available proposals are listed in a pop-up menu displayed at the current cursor position.

The **Content Completion Assistant** is enabled by default. To disable it, [open the Preferences dialog box](#), go to **Editor > Content Completion**, and disable the [Enable content completion option](#).

When active, the **Content Completion Assistant** displays a list of context-sensitive proposals valid at the current cursor position. You can navigate through the list of proposals by using the **Up** and **Down** 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 in **Text** mode, do one of the following:

- Press **Enter** or **Tab** to insert both the start and end tags and position the cursor inside the start tag in a position suitable for inserting attributes.
- Press **Ctrl + Enter (Command + Enter on OS X)** to insert both the start and end tags and positions the cursor between the tags in a position where you can start typing content.

Depending on the [selected schema version](#), Oxygen XML Developer plugin populates the proposals list with information taken either from XML Schema 1.0 or 1.1.

Oxygen XML Developer plugin helps you to easily reference a component by providing the list of proposals (complex types, simple types, elements, attributes, groups, attribute groups, or notations) valid in the current context. The components are collected from the current file or from the imported/included schemas.

When editing `xs:annotation/xs:appinfo` elements of an XML Schema, the Content Completion Assistant proposes elements and attributes from a custom schema (by default ISO Schematron). This feature can be configured from the [XSD Content Completion](#) preferences page.

XML Schema Outline View

The **Outline** view for XML Schemas presents all the global components grouped by their location, namespace, or type. By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

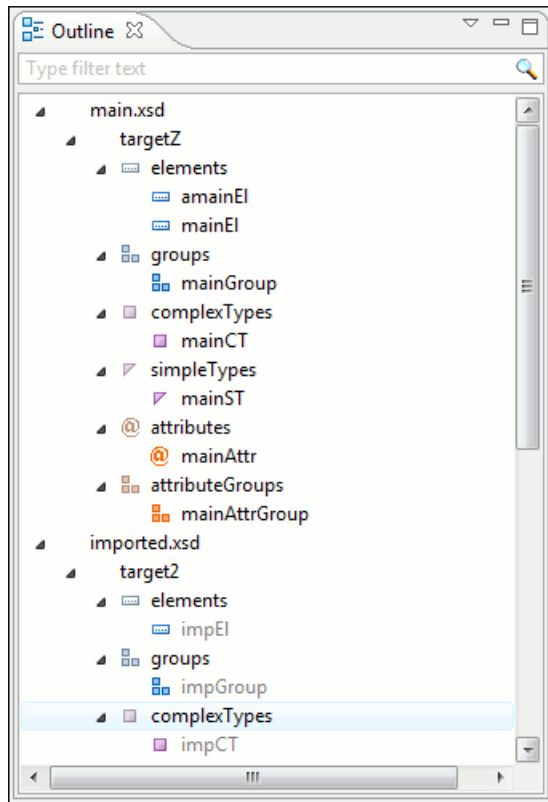


Figure 193: Outline View for XML Schema

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.

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.

Related information

[Searching and Refactoring Actions in XML Schemas](#) on page 313

[Component Dependencies View for XML Schema](#) on page 311

[XML Schema Resource Hierarchy / Dependencies View](#) on page 309

[Generating Sample XML Files](#) on page 315

[Editing Relax NG Schema in the Master Files Context](#) on page 358

XML Schema Attributes View

The **Attributes** view for XML Schemas presents the properties for the selected component in the schema diagram. By default, it is displayed on the right side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

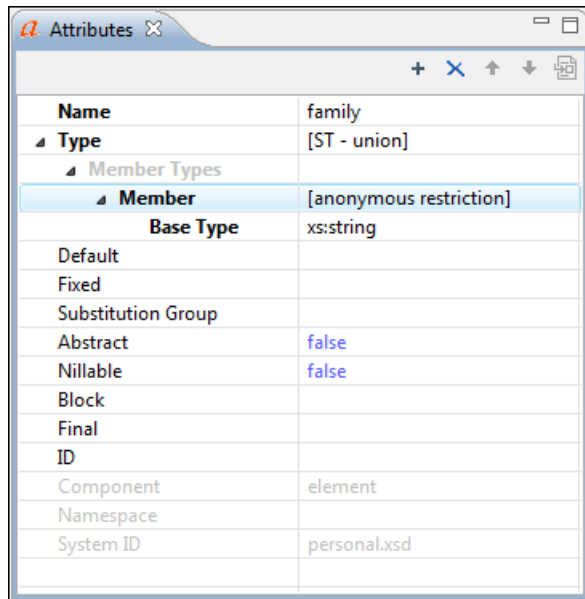


Figure 194: Attributes View

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 that contains at least one child is rendered with bold. Bold properties are properties with values set explicitly to them.

Properties for components that 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 Definition Ctrl (Meta on MAC OS) + Click

Shows the definition for the selected type.

Show Facets

Allows you to edit the facets for a simple type.

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](#). If the view is not displayed, it can be opened from the **Window > Show View** menu.

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.

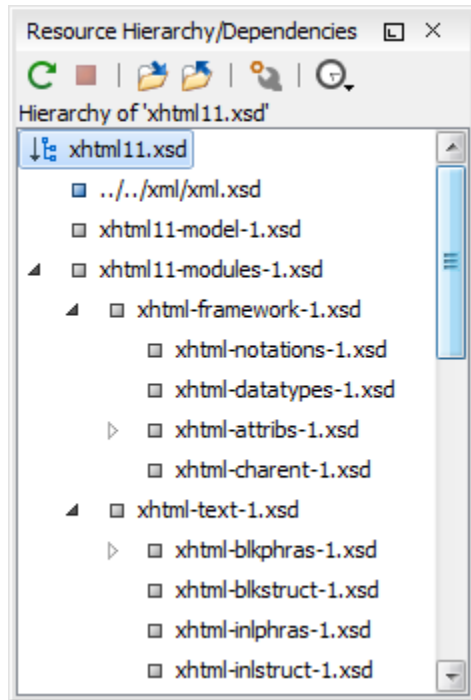


Figure 195: Resource Hierarchy/Dependencies View - Hierarchy for xhtml11.xsd

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

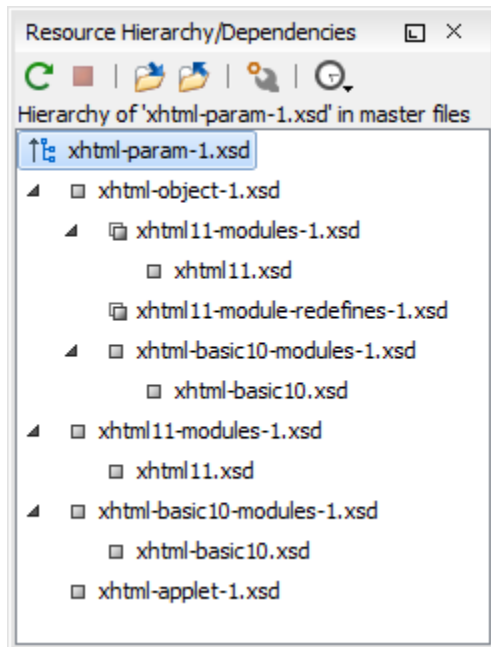


Figure 196: Resource Hierarchy/Dependencies View - Dependencies for xhtml-param-1.xsd

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*.

Related information

[Search and Refactor Operations Scope](#) on page 210

Moving/Renaming XML Schema Resources

You can 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 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*. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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).

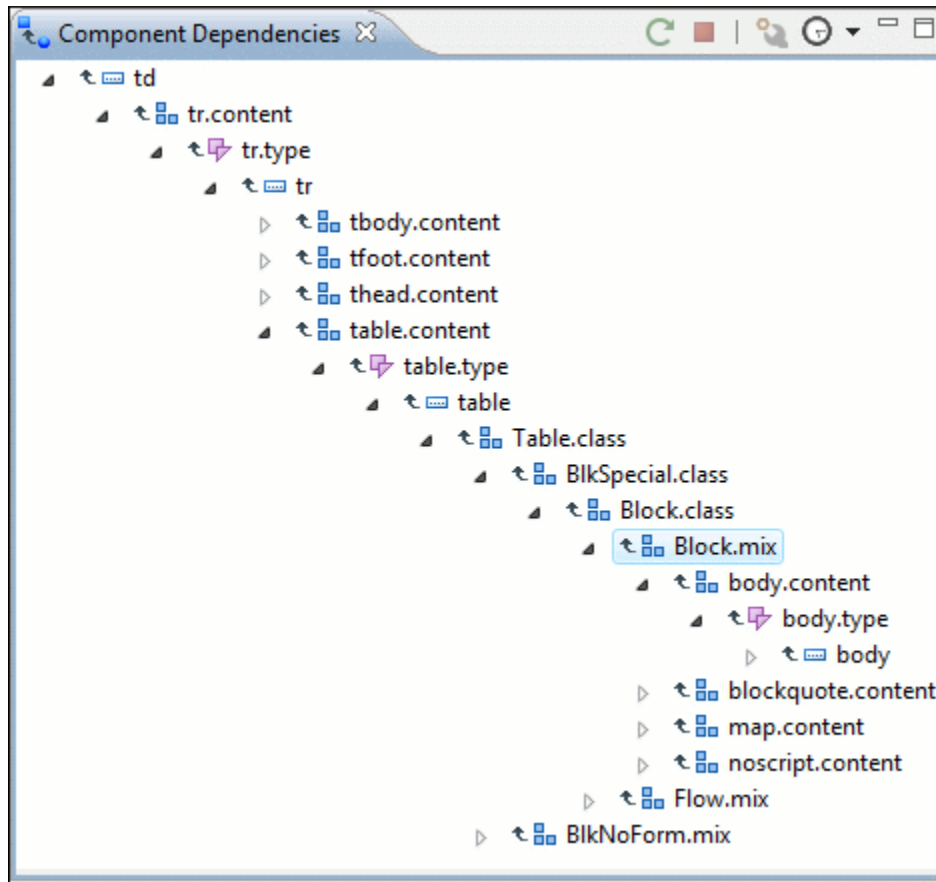


Figure 197: Component Dependencies View - Hierarchy for xhtml11.xsd

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 .

Related information

[Search and Refactor Operations Scope](#) on page 210

Highlight Component Occurrences

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

This feature is on by default. To configure it, [open the Preferences dialog box](#) and go to **Editor > Mark Occurrences**. A search can also be triggered with the **Search > Search Occurrences in File ()** contextual menu action. All matches are displayed in a separate tab of the **Results** view.

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 you define a scope in the **Search References** dialog box.
-  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
-  **Search Occurrences in File** - Searches all occurrences of the item at the 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 the in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in** - Opens a 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 to be affected by the action.

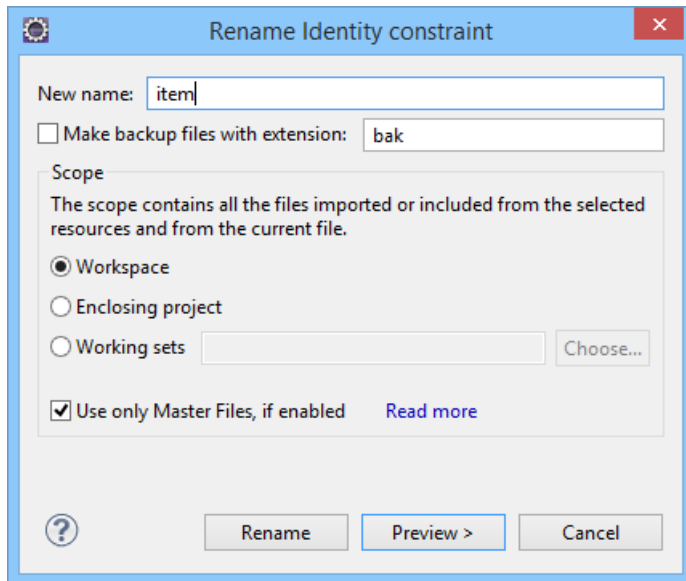


Figure 198: Rename Identity Constraint Dialog Box

Related information

[Search and Refactor Operations Scope](#) on page 210

Quick Fixes for DTD, XSD, and Relax NG Errors

Oxygen XML Developer plugin offers quick fixes for common errors that appear in XML documents that are validated against DTD, XSD, or Relax NG schemas.



Note: For XML documents validated against XSD schemas, the quick fixes are only available if you use the default Xerces validation engine.

Quick fixes are available in **Text** mode .

Oxygen XML Developer plugin provides quick fixes for numerous 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

Related information

[Schematron Quick Fixes \(SQF\)](#) on page 227

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 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 icon (💡) placed at the current line in the stripe on the left side of the editor. Also, you can invoke the quick assist menu by using the **Ctrl + 1** (**Meta 1** on Mac OS X) keyboard shortcuts.

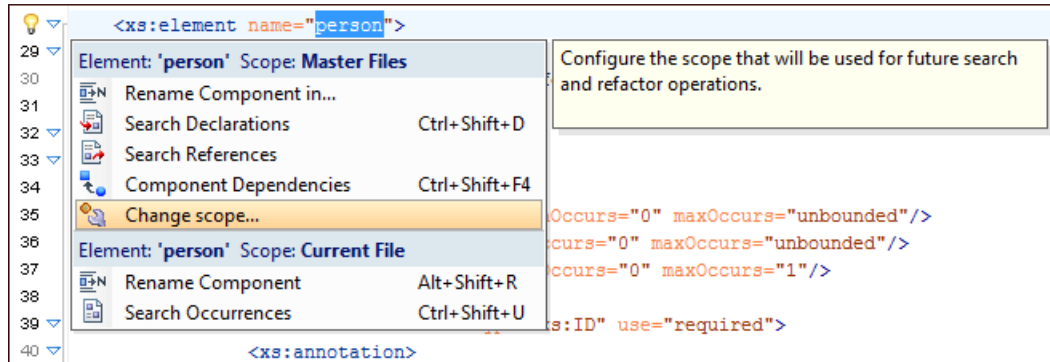


Figure 199: Quick Assist Support

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 https://www.oxygenxml.com/demo/Quick_Assist.html.

Related information

[Resource Hierarchy / Dependencies View](#) on page 309

[Component Dependencies View](#) on page 311

[Searching and Refactoring Actions](#) on page 313

Generating Sample XML Files

Oxygen XML Developer plugin offers support to generate sample XML files both from XML schema 1.0 and XML schema 1.1, depending on the XML schema version set in [XML Schema preferences page](#).

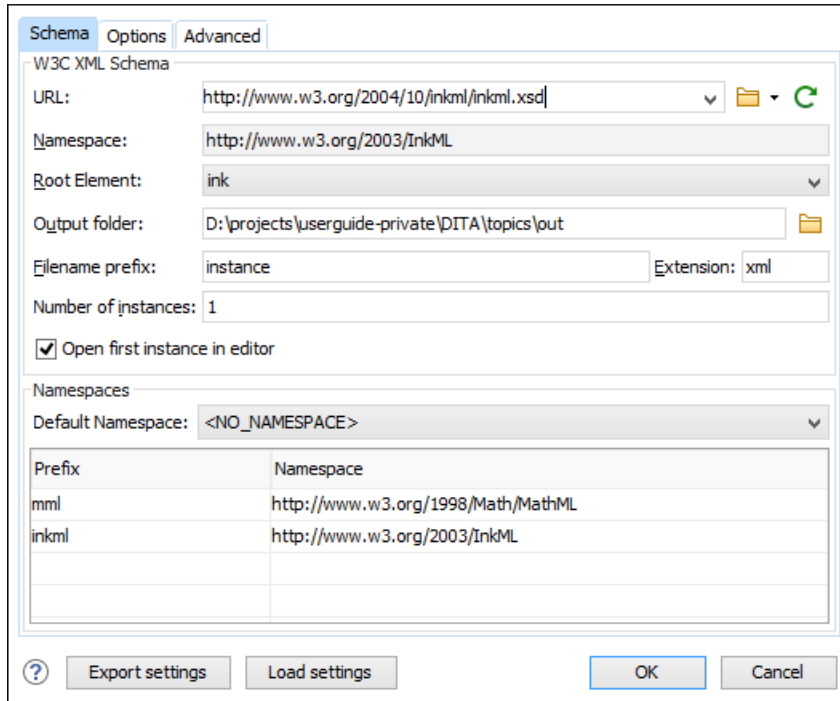
To generate sample XML files from an XML Schema, use the  **Generate Sample XML Files** action from the **XML Tools** menu. This action is also available in the contextual menu of the schema *Design mode*. The action opens the **Generate Sample XML Files** dialog box that allows you to configure a variety of options for generating the files.

For more information about this tool, watch our video demonstration about generating sample XML files at https://www.oxygenxml.com/demo/Generate_Sample_XML_Files.html.

The **Generate Sample XML Files** dialog box contains three tabs with various configurable options. Default values for these options can be set in the [Sample XML Files Generator preferences page](#).

Schema Tab (Generate Sample XML Files Tool)

The  **Generate Sample XML Files** tool includes a dialog box that allows you to configure a variety of options for generating the XML files. The first set of options are found in the **Schema** tab.



The screenshot shows the 'Generate Sample XML Files' dialog box with the 'Schema' tab selected. The 'W3C XML Schema' section includes a 'URL' field with a text input and a 'Browse' button, a 'Namespace' field, a 'Root Element' dropdown menu, an 'Output folder' field with a 'Browse' button, a 'Filename prefix' field, an 'Extension' field, and a 'Number of instances' field. There is a checkbox for 'Open first instance in editor'. The 'Namespaces' section includes a 'Default Namespace' dropdown menu and a table with two columns: 'Prefix' and 'Namespace'.

Prefix	Namespace
mml	http://www.w3.org/1998/Math/MathML
inkml	http://www.w3.org/2003/InkML

Figure 200: Generate Sample XML Files Dialog Box (Schema Tab)

This tab includes the following options:

URL

Specifies the URL of the Schema location. You can specify the path by using the text field, the history drop-down menu, or the browsing tools in the  **Browse** drop-down list.

Namespace

Displays the namespace of the selected schema.

Root Element

After the schema is selected, this drop-down menu 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

You can specify the prefix and extension for the file name that will be generated. Generated file names have the following format: **prefixN.extension**, where N represents an incremental number from 0 up to the specified **Number of instances**.

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 the editor.

Namespaces section

You can specify the **Default Namespace**, as well as the prefixes for the namespaces.

Load settings

Use this button to load previously exported settings.

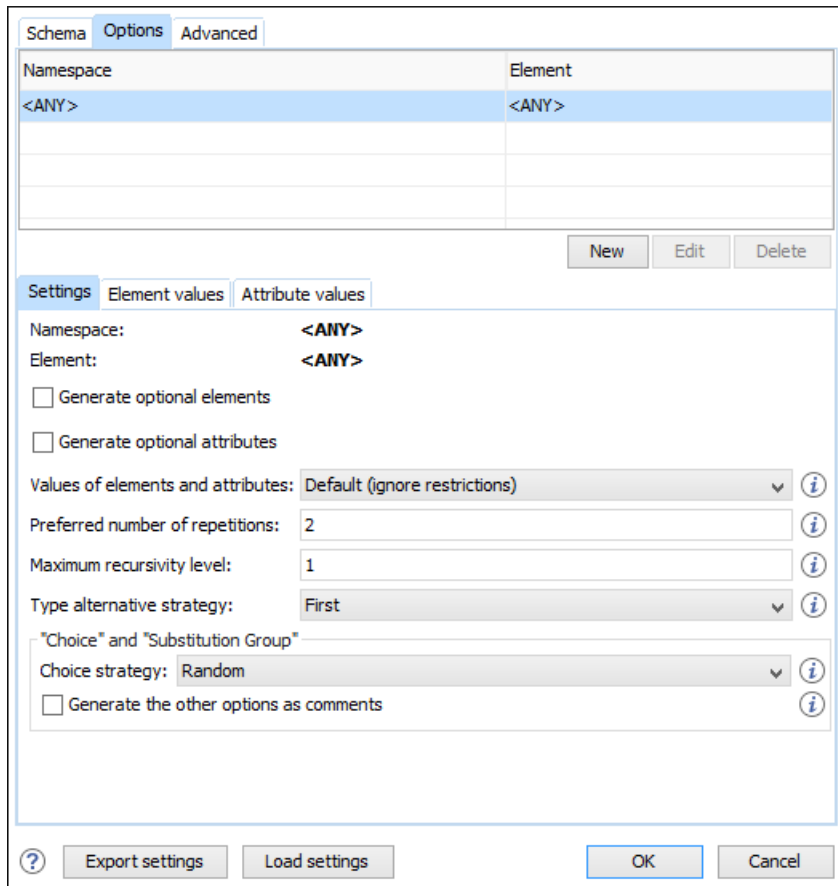
Export settings

Use this button to save the current settings for future use.

You can click **OK** at any point to generate the sample XML files.

Options Tab (Generate Sample XML Files Tool)

The  **Generate Sample XML Files** tool includes a dialog box that allows you to configure a variety of options for generating the XML files. The **Options** tab allows you to set specific options for namespaces and elements.



Namespace	Element
<ANY>	<ANY>

New Edit Delete

Settings Element values Attribute values

Namespace: <ANY>
Element: <ANY>

☐ Generate optional elements
☐ Generate optional attributes

Values of elements and attributes: Default (ignore restrictions) ⓘ

Preferred number of repetitions: 2 ⓘ

Maximum recursivity level: 1 ⓘ

Type alternative strategy: First ⓘ

"Choice" and "Substitution Group"

Choice strategy: Random ⓘ

☐ Generate the other options as comments ⓘ

? Export settings Load settings OK Cancel

Figure 201: Generate Sample XML Files Dialog Box (Options Tab)

This tab includes the following options:

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.
- All elements from a specific namespace.
- A specific element from a specific namespace.

Settings subtab

Namespace

Displays the namespace specified in the table at the top of the dialog box.

Element

Displays the element specified in the table at the top of the dialog box.

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. The following 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 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.

Preferred number of repetitions

Allows you to set the preferred number of repeating elements related to `minOccurs` and `maxOccurs` facets defined in the 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.

Type alternative strategy

Used for the `xs:alternative` element from XML Schema 1.1. The possible strategies are:

- **First** - The first valid alternative type is always used.
- **Random** - A random alternative type is used.

Choice strategy

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

If enabled, generates 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.

Element values subtab

Allows you to add values that are used to generate the content of elements. If there are multiple values, then the values are used in a random order.

Attribute values subtab

Allows you to add values that are used to generate the content of attributes. If there are multiple values, then the values are used in a random order.

Load settings

Use this button to load previously exported settings.

Export settings

Use this button to save the current settings for future use.

You can click **OK** at any point to generate the sample XML files.

Advanced Tab (Generate Sample XML Files Tool)

The  **Generate Sample XML Files** tool includes a dialog box that allows you to configure a variety of options for generating the XML files. The **Advanced** tab allows you to set some options in regards to output values and performance.

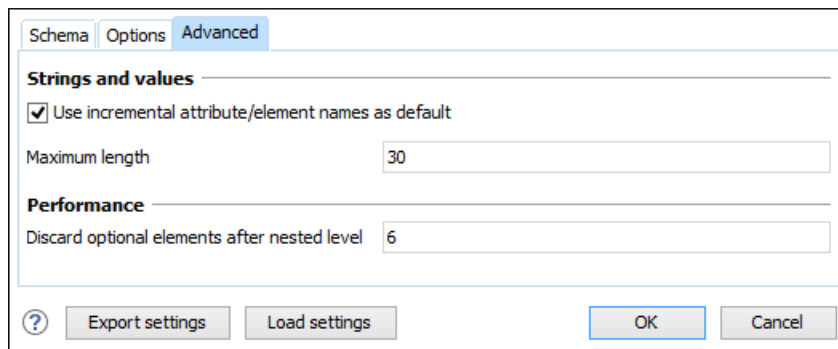


Figure 202: Generate Sample XML Files Dialog Box (Advanced Tab)

This tab includes the following options:

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 Developer 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.

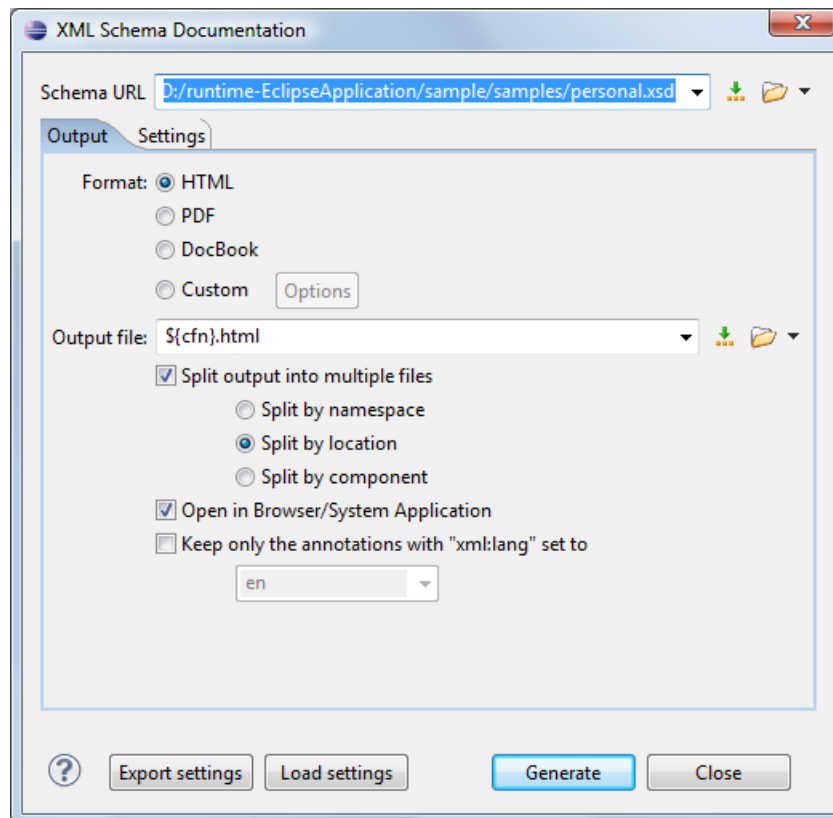


Figure 203: XML Schema Documentation Dialog Box

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.

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, go to **Window > Preferences > General > Web Browser** and specify it there. 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.).

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.

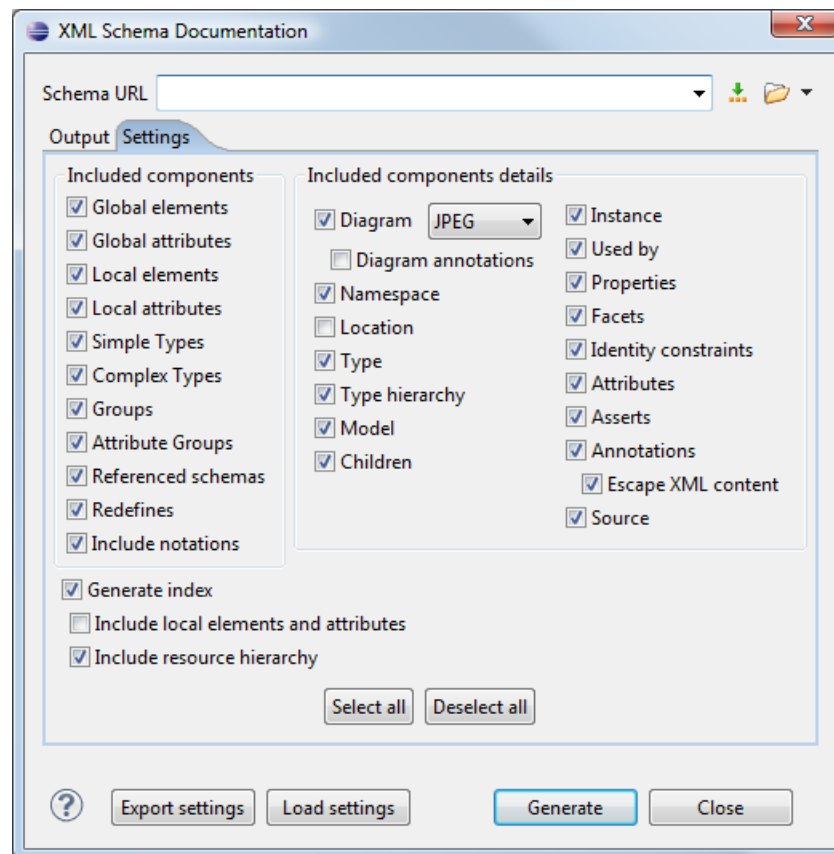


Figure 204: Settings Tab of the XML Schema Documentation Dialog Box

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 or not 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. The separator characters that are used depend upon 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 or not the resource hierarchy for an XML Schema documentation is generated. It is disabled by default.

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.

Related information

[Customizing the PDF Output of Generated XML Schema Documentation](#) on page 325

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 Developer plugin XSD editor panel](#).

Table of Contents

Group by:

Location

personal.xsd

Elements

email

family

given

link

name

person

personnel

url

Notations

gif

XML Schema documentation generated by XML Editor.

Element name

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

Showing:

☒ Annotations

☒ Attributes

☒ Diagrams

☒ Facets

☒ Identity Constraints

☒ Instances

☒ Model

☒ Properties

☒ Source

☒ Used by

Close

Figure 205: 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.

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

Figure 206: 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.

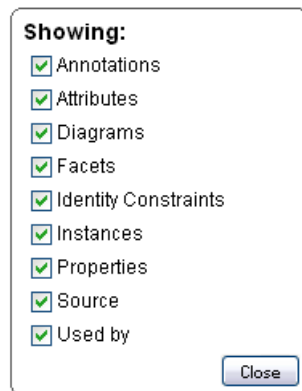


Figure 207: Showing Options

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.

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

Figure 208: Documentation for a Schema Component

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

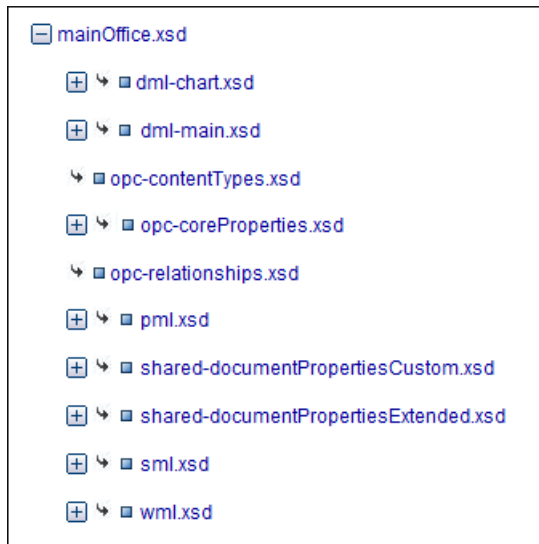


Figure 209: Example: Generated Documentation

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 [Customizing the PDF Output of Generated XML Schema Documentation](#) on page 325.

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_INSTALL_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_INSTALL_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_INSTALL_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 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:

https://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_INSTALL_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_INSTALL_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 Developer plugin installation folder. The script can be integrated in an external batch process launched from the command-line interface. The script accepts a variety of 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
```

Converting Schema to Another Schema Language

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

To use this tool, select the  **Generate/Convert Schema (Ctrl + Shift + BackSlash (Command + Shift + BackSlash on OS X))** action from the **XML Tools** menu. This action opens the **Generate/Convert Schema** dialog box that allows you to configure various options for conversion.

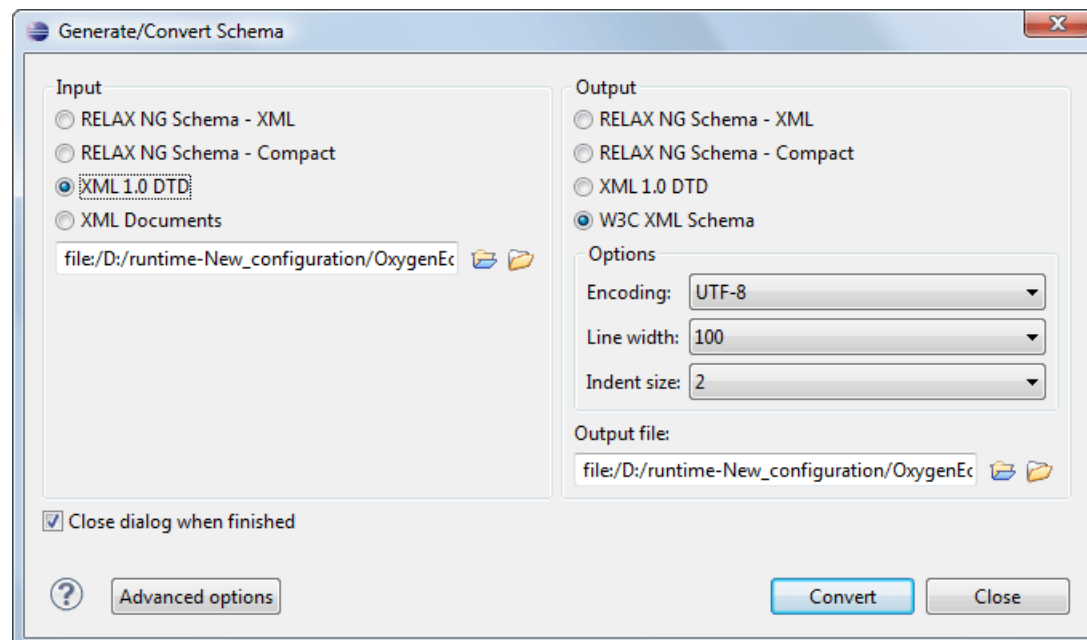


Figure 210: Generate/Convert Schema Dialog Box

The **Generate/Convert Schema** dialog box includes the following options:

Input section

Allows you to select the language of the source schema. If the conversion is based on a set of XML files, rather than just a single XML file, select the **XML Documents** option and use the file selector to add the XML files involved in the conversion.

Output section

Allows you to select the language of the target schema.

Options

You can choose the **Encoding**, the maximum **Line width**, and the **Indent size** (in number of spaces) for one level of indentation.

Output file

Specifies the path for the output file that will be generated.

Close dialog when finished

If you deselect this option, the dialog box will remain opened after the conversion so that you can easily continue to convert more files.

Advanced options

If you select **XML 1.0 DTD** for the input, you can click this button to access more advanced options to further fine-tune the conversion. The following advanced options are available:

XML 1.0 DTD Input section

These options apply to the source DTD:

- **xmlns** - Specifies the default namespace, that is the namespace used for unqualified element names.
- **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.
- **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.
- **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.
- **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.
- **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.
- **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.
- **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 or not 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.
- **xmlns mappings** table - Each row specifies the prefix used for a namespace in the input schema.

W3C XML Schema Output section

This section is available if you select **W3C XML Schema** for the output.

- **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).

Converting Database to XML Schema

Oxygen XML Developer 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.

Result: 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**.

Result: 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**.

Result: 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 linked by `xs:include` and `xs:import` statements on several levels. In some cases, working on such a schema as if it were a single file is more convenient than working on multiple files separately. The **Flatten Schema** operation allows you to flatten an entire hierarchy of XML schemas. Starting with the main XML schema, Oxygen XML Developer plugin calculates its hierarchy by processing the `xs:include` and `xs:import` statements.

The **Flatten Schema** action is available from the **Refactoring** submenu in the contextual menu in **Text** mode. The action opens the **Flatten Schema** dialog box that allows you to configure the operation.

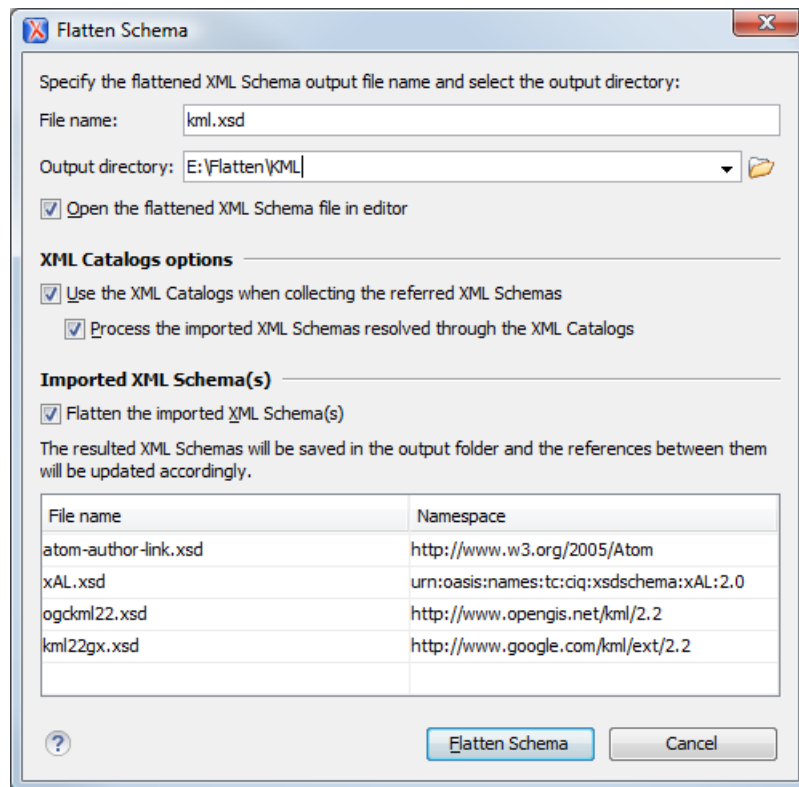


Figure 211: Flatten Schema Dialog Box

For the main schema file and for each imported schema, a new flattened schema is generated in the specified 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.

A flattened XML schema is obtained by recursively adding the components of the included schemas into the main one. This means Oxygen XML Developer plugin replaces the `xs:include`, `xs:redefine`, and `xs:override` elements with the ones coming from the included files.

Options in the Flatten Schema Dialog Box

The following options are available in the **Flatten Schema** dialog box:

File name

The name of the output file.

Output directory

The path of the output directory where the flattened schema file will be saved.

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 the imported and included schemas to be resolved 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 or not the imported schemas that were resolved through an XML Catalog are also processed.

Flatten the imported XML Schema(s)

Specifies whether or not 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.

Flatten Schema from the Command Line

The **Flatten Schema** tool can be also ran from command line by using 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 whether or not the imported XML Schemas should be flattened. 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 whether or not the imported schemas that were resolved through the XML Catalogs should be flattened. The default value is `true`.



Note: This option is used only when `-useCatalogs` is set to `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 you to test regular expressions on a fragment of text as they are applied to an XML instance document. Start the tool by selecting **XML Schema Regular Expressions Builder** from the **XML Tools** menu.

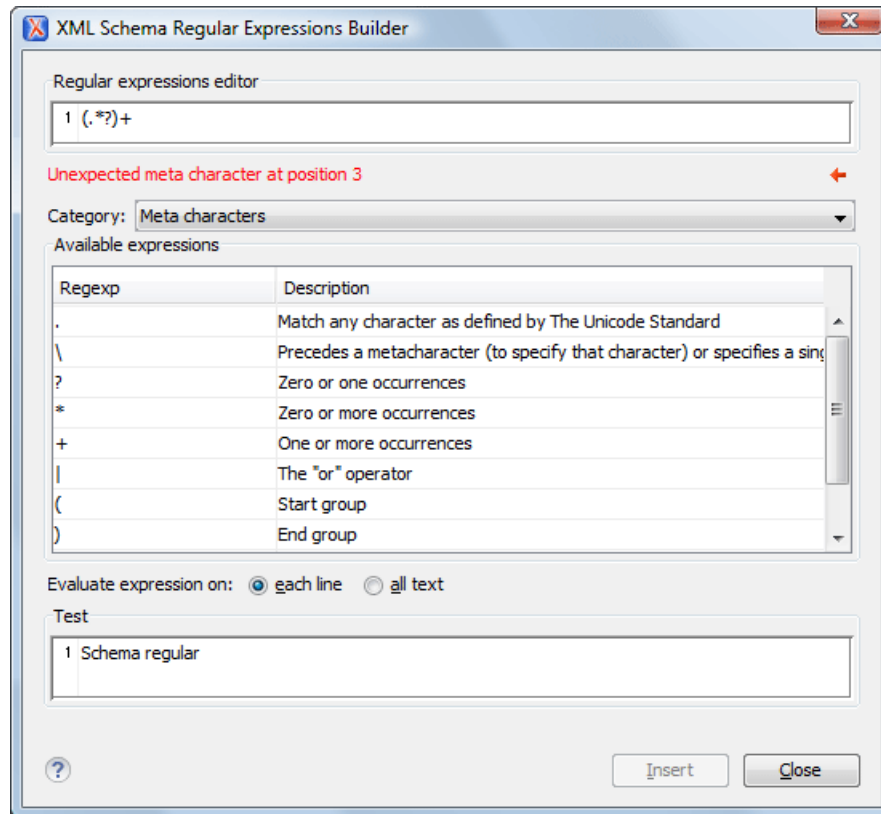


Figure 212: XML Schema Regular Expressions Builder Dialog Box

The dialog box contains the following:

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

You can choose from several categories of predefined expressions. The selected category influences the displayed expressions in the **Available expressions** table.

Available expressions

This table includes the available regular expressions and a short description for each of them. The set of expressions depends on the category selected in the previous **Category** 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.

Evaluate expression on

You can choose between 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

A text editor that allows you to enter a text sample for 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 mode* or *schema Design mode*. Accordingly, the **Insert** button will be disabled if the current document is edited in these modes.



Note: Some regular expressions may indefinitely block 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 Developer plugin offers full support for XML Schema 1.1, including:

- XML Documents Validation and Content Completion Based on XML Schema 1.1.
- XML Schema 1.1 Validation and Content Completion.
- Editing XML Schema 1.1 files in the Schema **Design** mode.
- The Flatten Schema action.
- Resource Hierarchy/Dependencies and Refactoring Actions.
- Master Files.
- Generating Documentation for XML Schema 1.1.
- Support for generating XML instances based on XML Schema.
- 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).



Tip: 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.

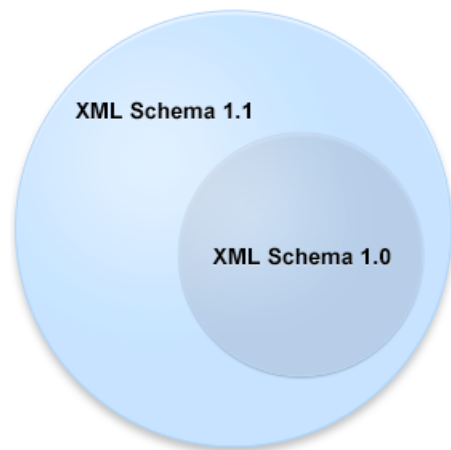


Figure 213: XML Schema 1.1

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 of schema an element is validated against, based on the values of the attribute of the element.
- **Open content** - Content models can 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 default may be set that 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 that 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 Assistant** presents XML Schema 1.1 elements that you can insert accidentally in an 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 https://www.oxygenxml.com/demo/XML_Schema_11.html.

Related information

[Setting the XML Schema Version](#) on page 334

Setting the XML Schema Version

Oxygen XML Developer plugin lets you set the version of the XML Schema you are editing either in the **XML Schema** preferences page, or through the versioning attributes. 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 30: Using the `minVersion` and `maxVersion` Attributes to Set the XML Schema Version

Versioning Attributes	XML Schema Version
<code>minVersion = "1.0" maxVersion = "1.1"</code>	1.0
<code>minVersion = "1.1"</code>	1.1
<code>minVersion = "1.0" maxVersion = greater than "1.1"</code>	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 Developer plugin to its default options.

Oxygen XML Developer 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 Developer plugin takes into account the `minVersion` and `maxVersion` attributes defined in the XML Schema.

Related information

[XML Schema 1.1](#) on page 333

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 **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 and it allows quick access to components. By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

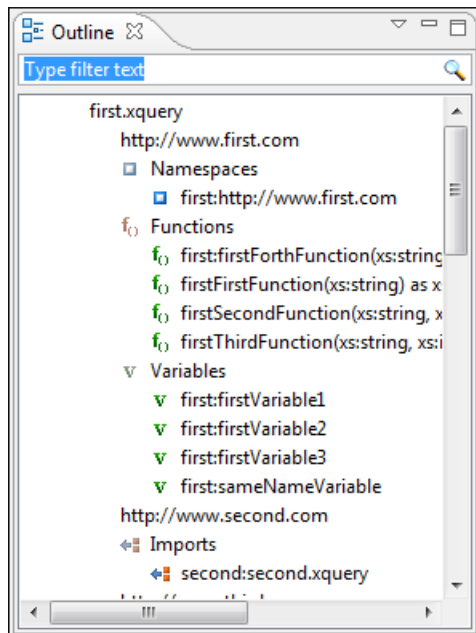


Figure 214: XQuery Outline View

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.

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.

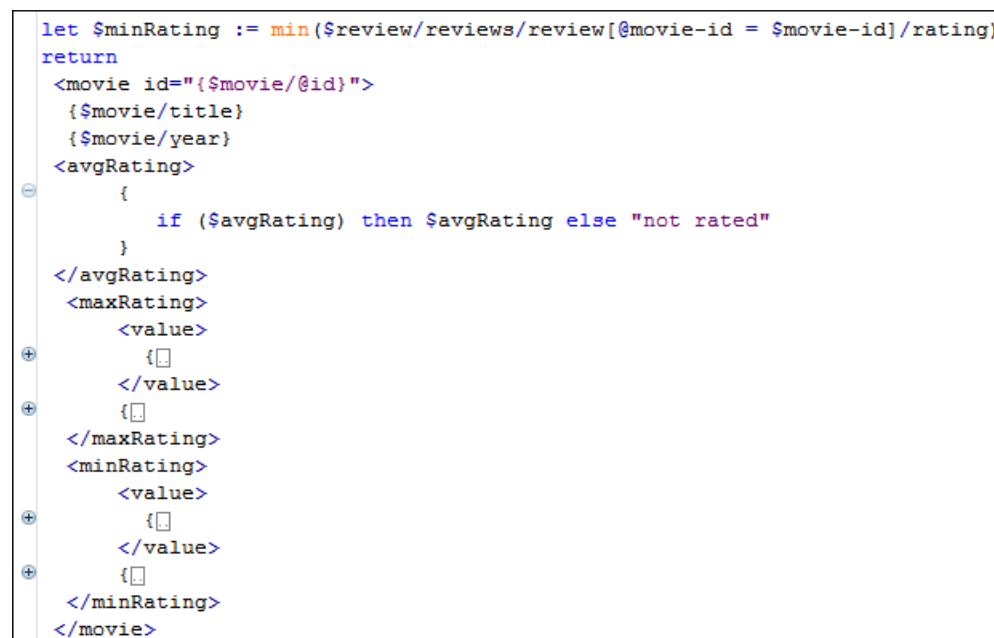


Figure 215: Folding in XQuery Documents

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 Developer 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.

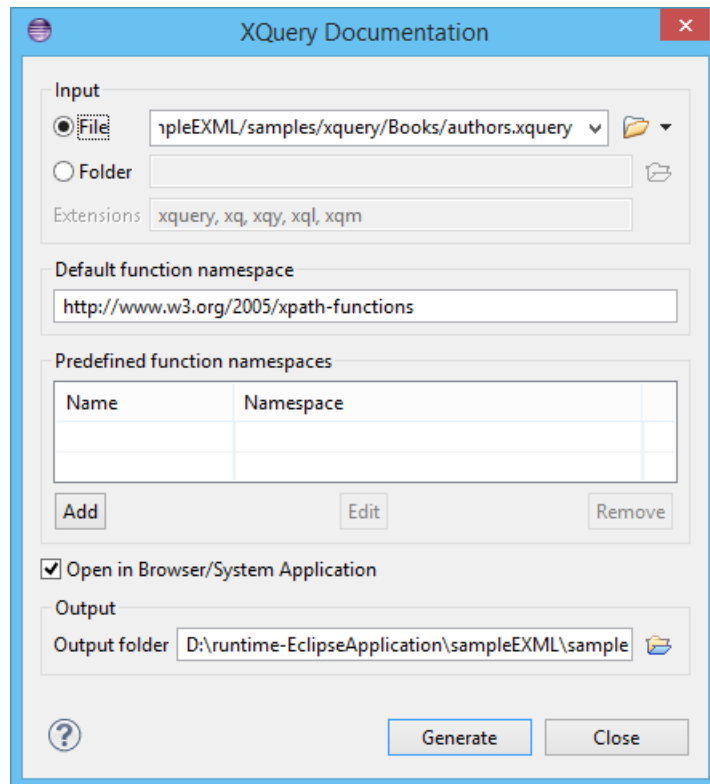


Figure 216: XQuery Documentation Dialog Box

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, go to **Window > Preferences > General > Web Browser** and specify it there. 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 Developer plugin provides a special type of editor dedicated to WSDL documents. The WSDL editor offers support for validation, a specialized **Content Completion Assistant**, a component oriented **Outline** view, searching and refactoring operations, and support to generate documentation.

Both WSDL version 1.1 and 2.0 are supported and SOAP versions 1.1 and 1.2. That means that in the location where a SOAP extension can be inserted the **Content Completion Assistant** offers elements from both SOAP 1.1 and SOAP 1.2. Validation of SOAP requests is executed first against a SOAP 1.1 schema and then against a SOAP 1.2 schema. In addition to validation against the XSD schemas, Oxygen XML Developer plugin also checks if the WSDL file conforms with the WSDL specification (available only for WSDL 1.1 and SOAP 1.1).

In the following example you can see how the errors are reported.

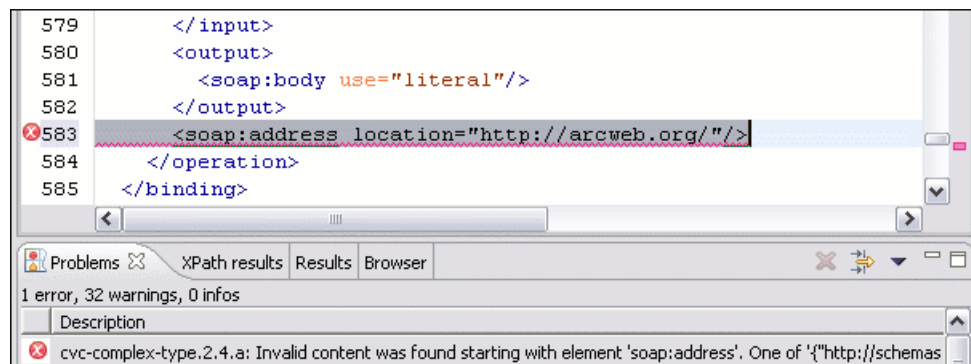


Figure 217: Validating a WSDL file

To watch our video demonstration about the WSDL editing support in Oxygen XML Developer plugin, go to https://www.oxygenxml.com/demo/Create_New_WSDL.html.

Related information

[Editing XML Documents in Text Mode](#) on page 161

Editing WSDL Documents in the Master Files Context

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

You can set a main WSDL document either using the [master files support from the Navigator view](#), or using a validation scenario.

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

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

- Correct validation of a module in the context of a larger WSDL structure.
- **Content Completion Assistant** displays all components valid in the current context.
- The **Outline** displays the components collected from the entire WSDL structure.



Note: When you edit an XML schema document that has a WSDL document set as master, the validation operation is performed over the master WSDL document.

To watch our video demonstration about editing WSDL documents in the master files context, go to https://www.oxygenxml.com/demo/WSDL_Working_Modules.html.

Validating WSDL Documents

By default, WSDL files are 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 a WSDL document 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 Assistance 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.

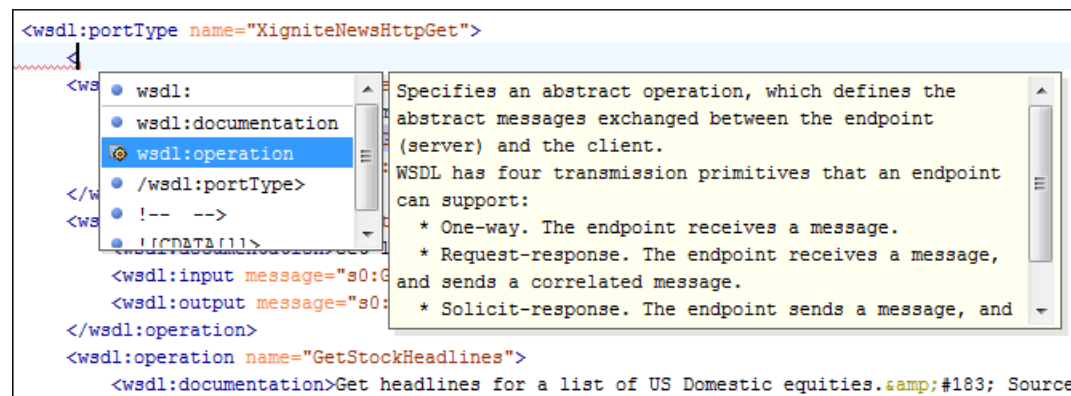


Figure 218: WSDL Content Completion Assistant

 **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 assistance window to speed up the insertion into the document of prefixed elements.

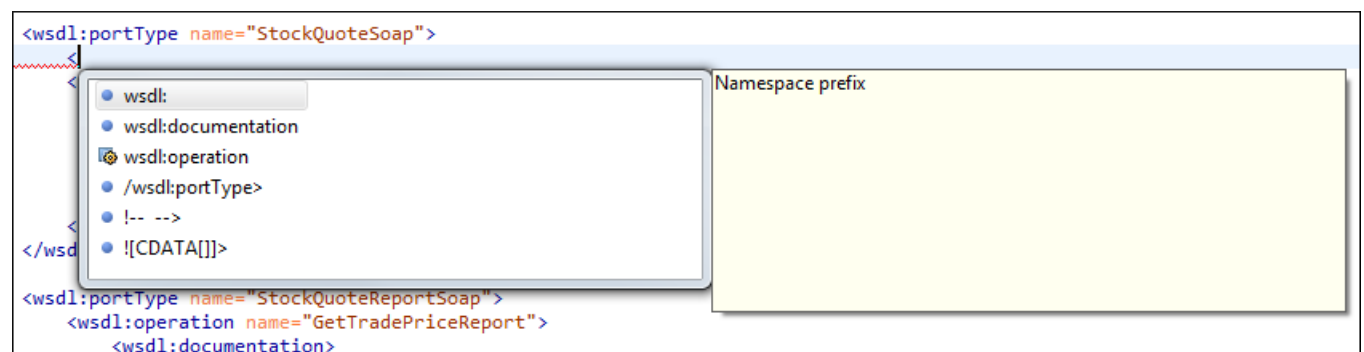


Figure 219: Namespace Prefixes in the Content Completion Assistant

For the common namespaces, such as XML Schema namespace (<http://www.w3.org/2001/XMLSchema>) or SOAP namespace (<http://schemas.xmlsoap.org/wsdl/soap/>), Oxygen XML Developer plugin provides an easy mode to declare them by proposing a prefix for these namespaces.

WSDL Outline View

The **Outline** view for WSDL documents 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.

By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

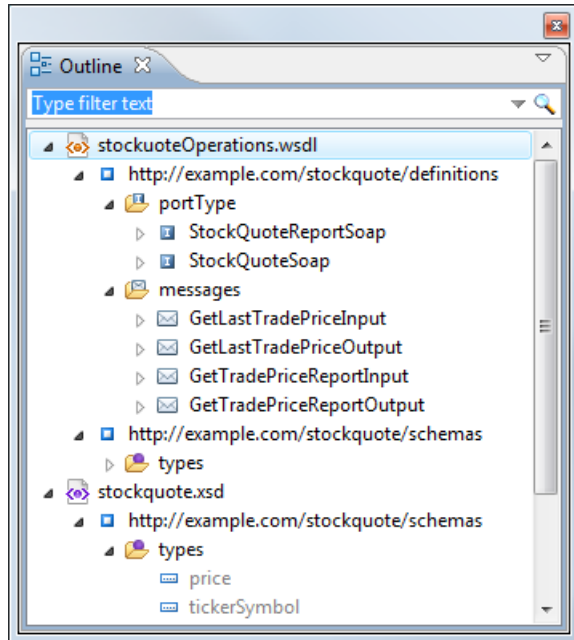


Figure 220: WSDL Outline View

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.

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.

WSDL Resource Hierarchy/Dependencies View in WSDL Documents

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a WSDL resource. If the view is not displayed, it can be opened from the **Window > Show View** menu.



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 **Navigator** view and choose **Resource Hierarchy** from the contextual menu.

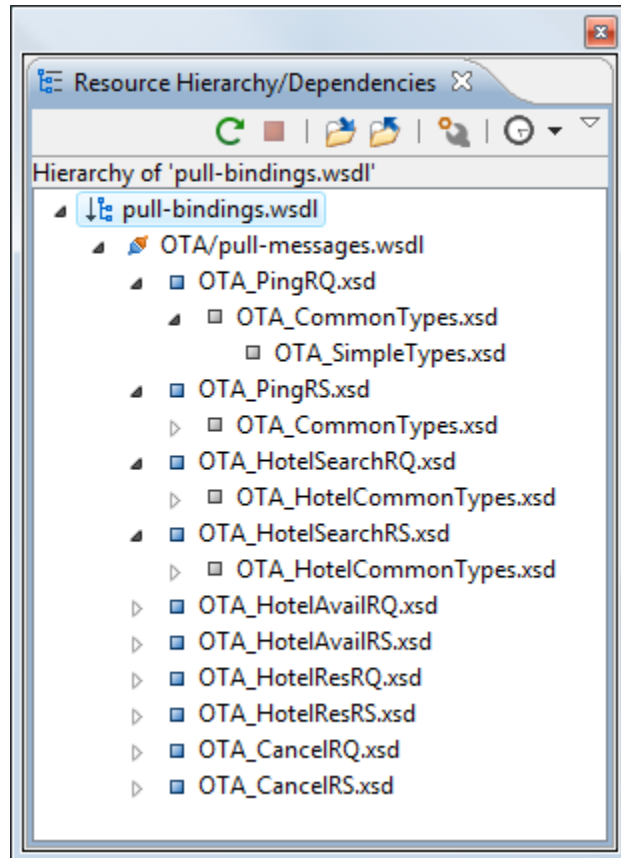


Figure 221: Resource Hierarchy/Dependencies View

If you want to see the dependencies of a WSDL document, select the document in the **Navigator** view and choose **Resource Dependencies** from the contextual menu.

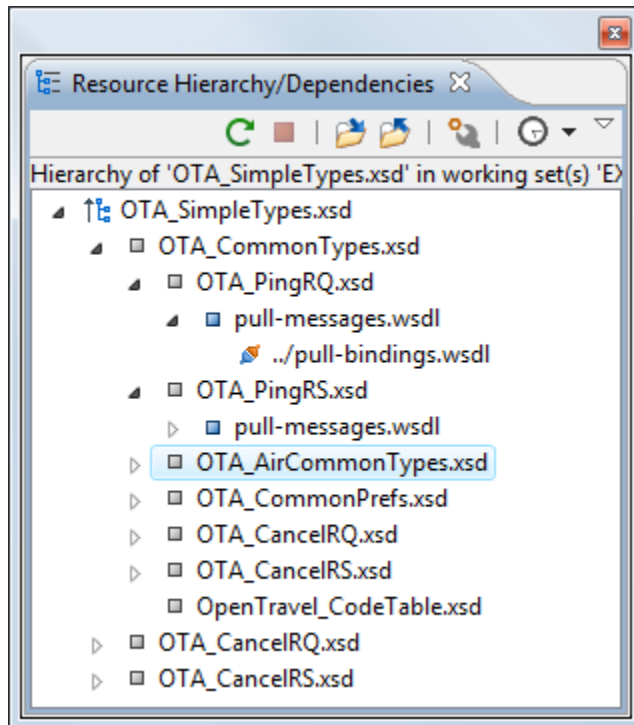


Figure 222: Resource Hierarchy/Dependencies View

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*.

Related information

Search and Refactor Operations Scope on page 210

Moving/Renaming WSDL Resources

You can 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. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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.

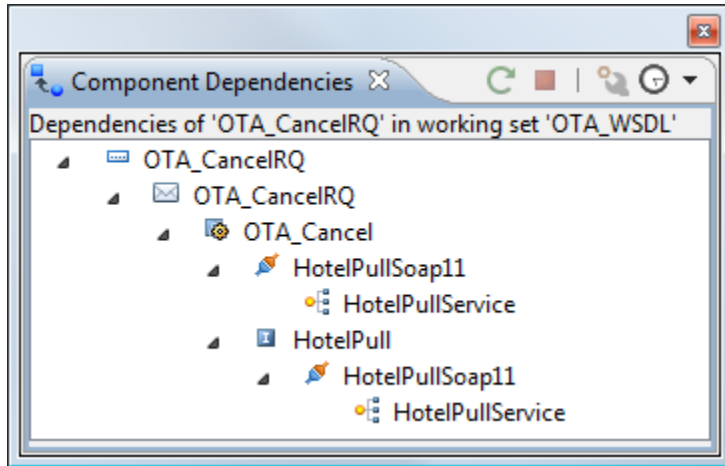


Figure 223: Component Dependencies View

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



Refresh

Refreshes the dependencies structure.



Stop

Stops the dependencies computing.



Configure

Allows you to configure a [search scope](#) to compute the dependencies structure. You can decide to use the defined scope for future operations automatically, by checking the corresponding check box.



History

Allows you to repeat a previous dependencies computation.

The following actions are available in the contextual menu of the **Component Dependencies** view:

Go to First Reference

Selects the first reference of the referenced component from the current selected component in the dependencies tree.

Go to Component

Displays the definition of the current selected component in the dependencies tree.



Tip: If a component contains multiple references to another, a small table is 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 Developer plugin searches for the component declaration and all its references and highlights them automatically.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected.

To change the default 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.

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 the in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in** - Opens a 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 to be affected by the action.

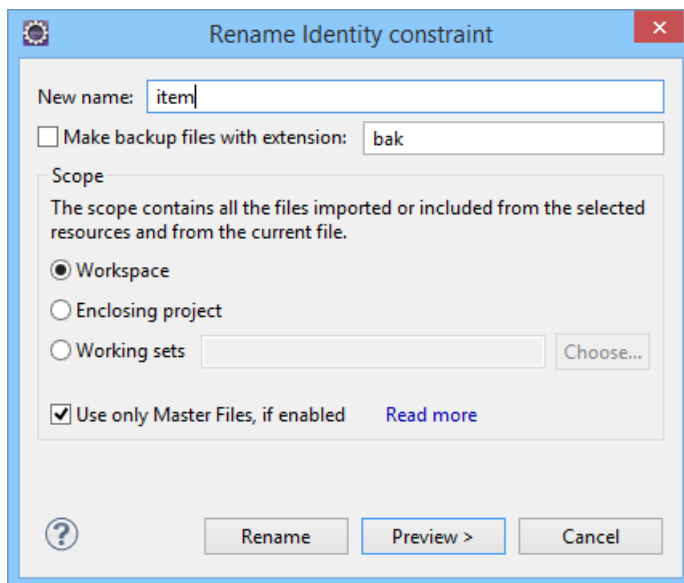


Figure 224: Rename Identity Constraint Dialog Box

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**

checkbox 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](#).

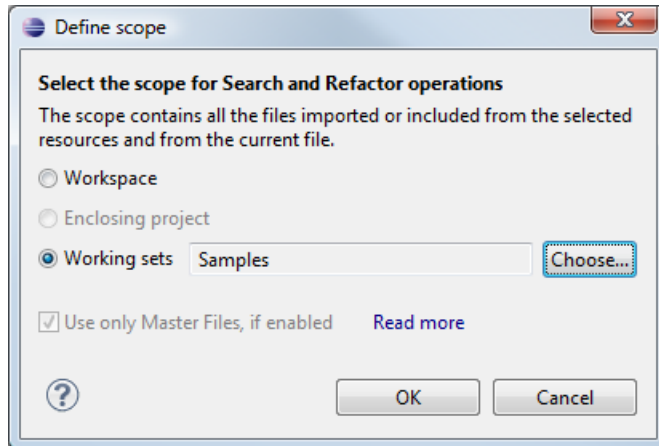


Figure 225: Change Scope Dialog Box

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.

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 icon (💡) placed at the current line in the stripe on the left side of the editor. Also, you can invoke the quick assist menu by using the **Ctrl + 1** (**Meta 1** on Mac OS X) keyboard shortcuts.

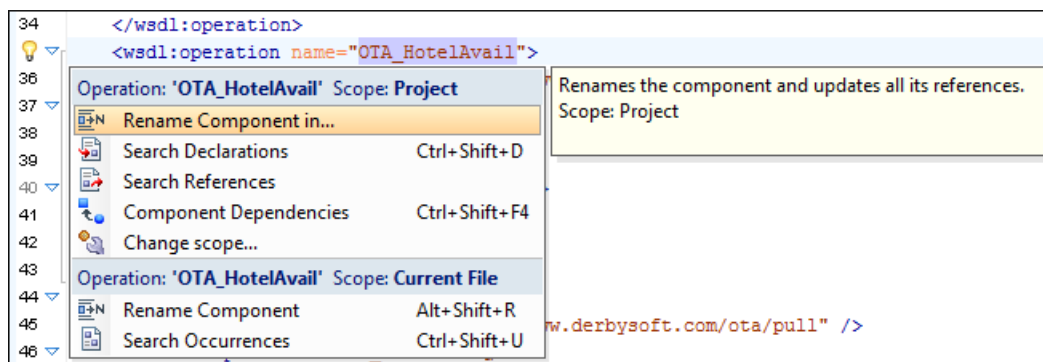


Figure 226: WSDL Quick Assist Support

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

Rename Component in

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

Generating Documentation for WSDL Documents

You can use Oxygen XML Developer plugin to generate detailed documentation for the components of a WSDL document in HTML format. You can select the WSDL components to include in your output and the level of details to present for each of them. Also, the components are hyperlinked. 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.

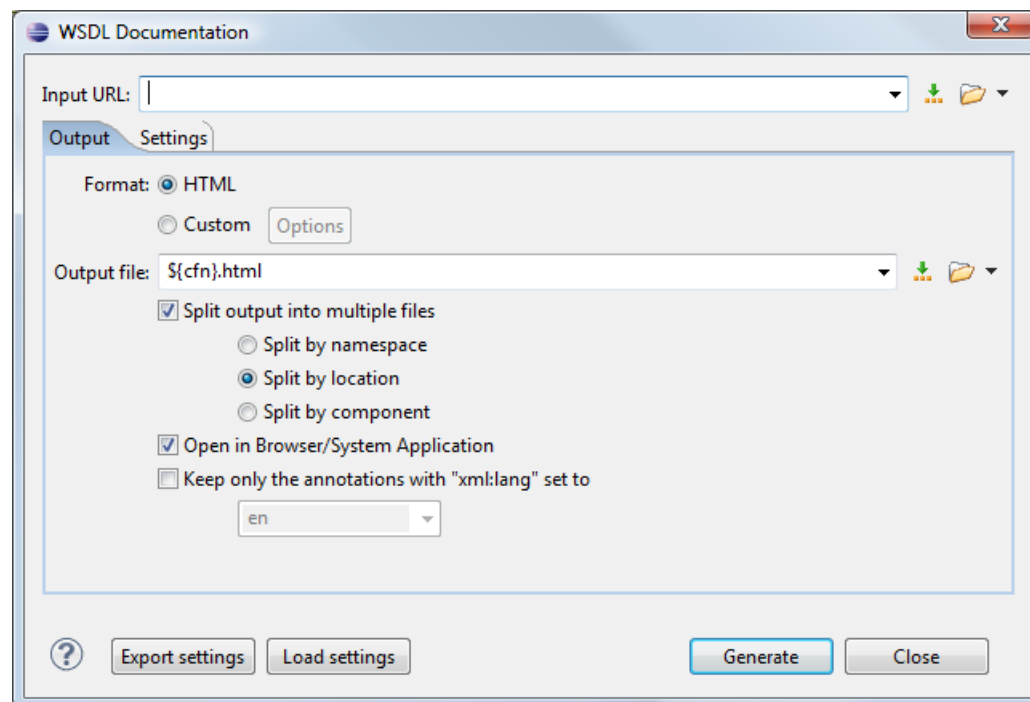


Figure 227: WSDL Documentation Dialog Box

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.

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, go to **Window > Preferences > General > Web Browser** and specify it there. 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.).

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.

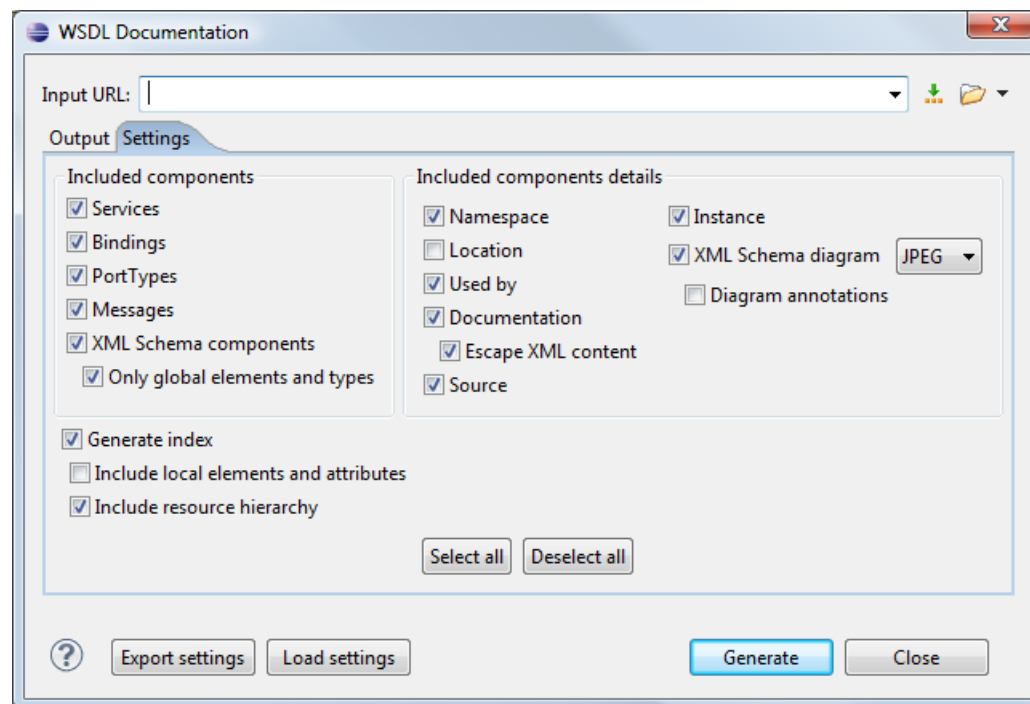


Figure 228: Settings Tab of the WSDL Documentation Dialog Box

The **Settings** tab allows you to choose whether or not to include the following:

- **Components**
 - **Services** - Specifies whether or not the generated documentation includes the WSDL services.
 - **Bindings** - Specifies whether or not the generated documentation includes the WSDL bindings.
 - **Port Types** - Specifies whether or not the generated documentation includes the WSDL port types.
 - **Messages** - Specifies whether or not the generated documentation includes the WSDL messages.
 - **XML Schema Components** - Specifies whether or not the generated documentation includes the XML Schema components.
 - **Only global elements and types** - Specifies whether or not 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 or not 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 or not the resource hierarchy for an XML Schema documentation is generated. It is disabled by default.

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.

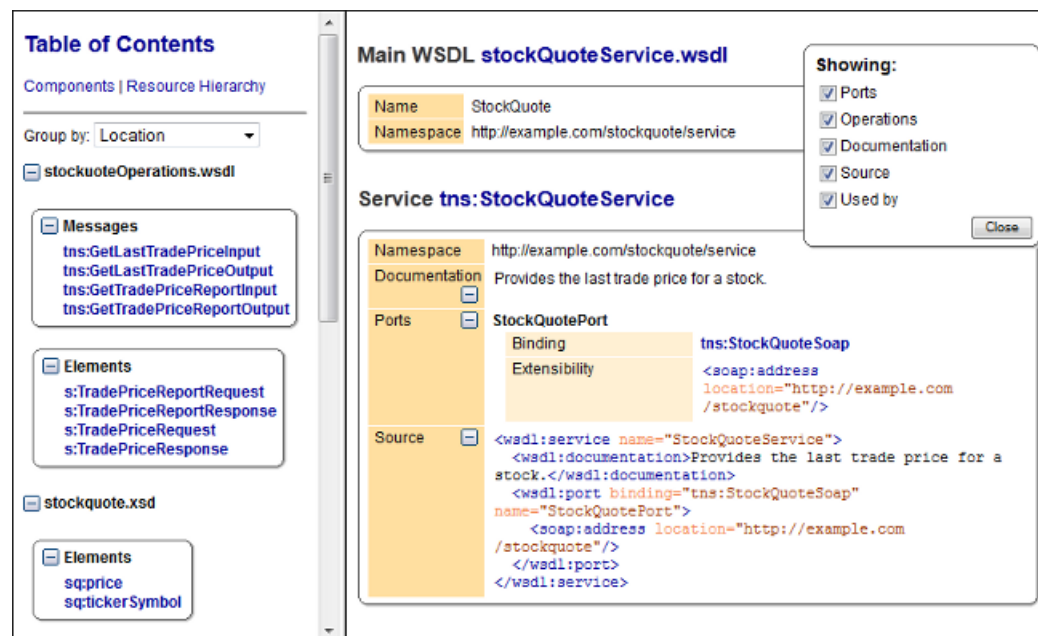


Figure 229: WSDL Documentation in HTML Format

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.

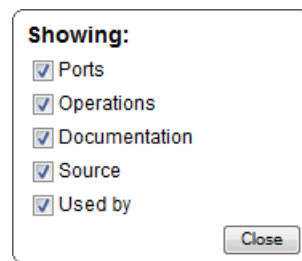


Figure 230: Showing Options

Generating WSDL Documentation in a Custom Format

To obtain the default HTML documentation output from a WSDL document, Oxygen XML Developer plugin uses an intermediary XML document to which it applies an XSLT stylesheet. To create a custom output from your WSDL document, edit 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_INSTALL_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_INSTALL_DIR]/frameworks/wsdl_documentation` folder.

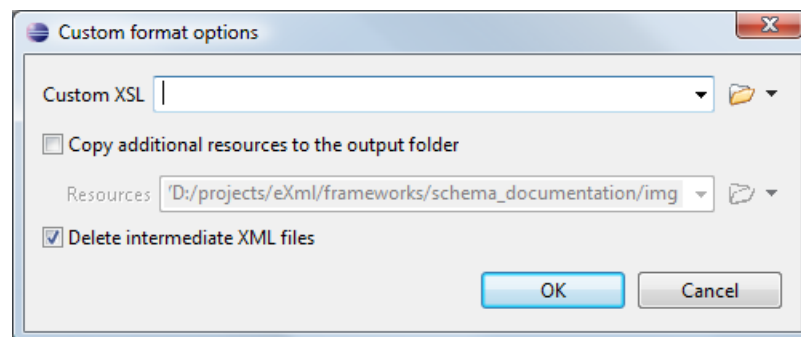


Figure 231: Custom Format Options Dialog Box

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 Developer plugin. You can integrate the scripts in an external batch process launched from the command-line interface.

WSDL SOAP Analyzer

After you edit and validate your Web service descriptor against a mix of the XML Schemas for WSDL and SOAP, it is easy to check if the defined SOAP messages are accepted by the remote Web Services server by using the integrated **WSDL SOAP Analyzer** tool (available from the toolbar or **WSDL** menu).

Composing a SOAP Request

WSDL SOAP Analyzer is a tool that helps you test if the messages defined in a Web Service Descriptor (WSDL) are accepted by a Web Services server.

Oxygen XML Developer plugin provides two ways of testing, one for the currently edited WSDL document and another for the remote WSDL documents that are published on a web server. To open the **WSDL SOAP 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.

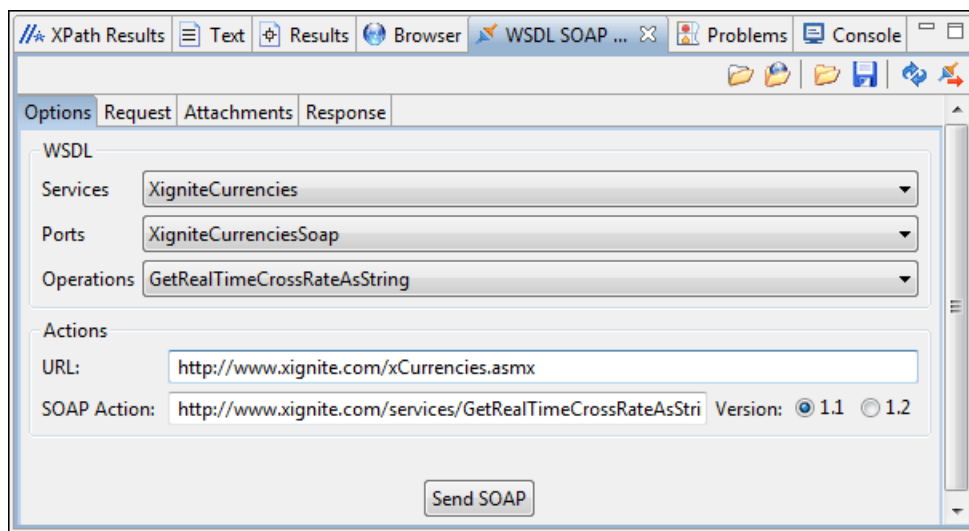


Figure 232: WSDL SOAP Analyzer View

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 Developer plugin tries to generate as much content as possible for the SOAP request. The envelope of the SOAP request has the correct namespace for the selected SOAP version, that is *http://schemas.xmlsoap.org/soap/envelope/* for SOAP 1.1 or *http://www.w3.org/2003/05/soap-envelope* for SOAP 1.2. Usually you just have to change a few values 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 various operations, Oxygen XML Developer plugin remembers the modified request for each one. You can press the **Regenerate** button 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 Developer plugin prompts you to save them, then tries to open them with the associated system application.

- **Errors List** - There may be situations where the WSDL file is respecting the WSDL XML Schema, but it fails to be valid (for example, in the case of a message that is defined by means of an element that is not found in the types section of the WSDL). In such a case, the errors are listed here. This list is presented only when there are errors.
- **Send Button** - Executes the request. A status dialog box is displayed when Oxygen XML Developer plugin is connecting to the server.

The testing of a WSDL file is straight-forward: click the WSDL analysis button, then select the service, the port, and the operation. The editor generates the skeleton for the SOAP request. You can edit the request, eventually attach files to it and send it to the server. Watch the server response in the response area. You can find more details in the [Testing Remote WSDL Files](#) section.



Note: SOAP requests and responses are automatically validated in the **WSDL SOAP 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 Developer 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.

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.

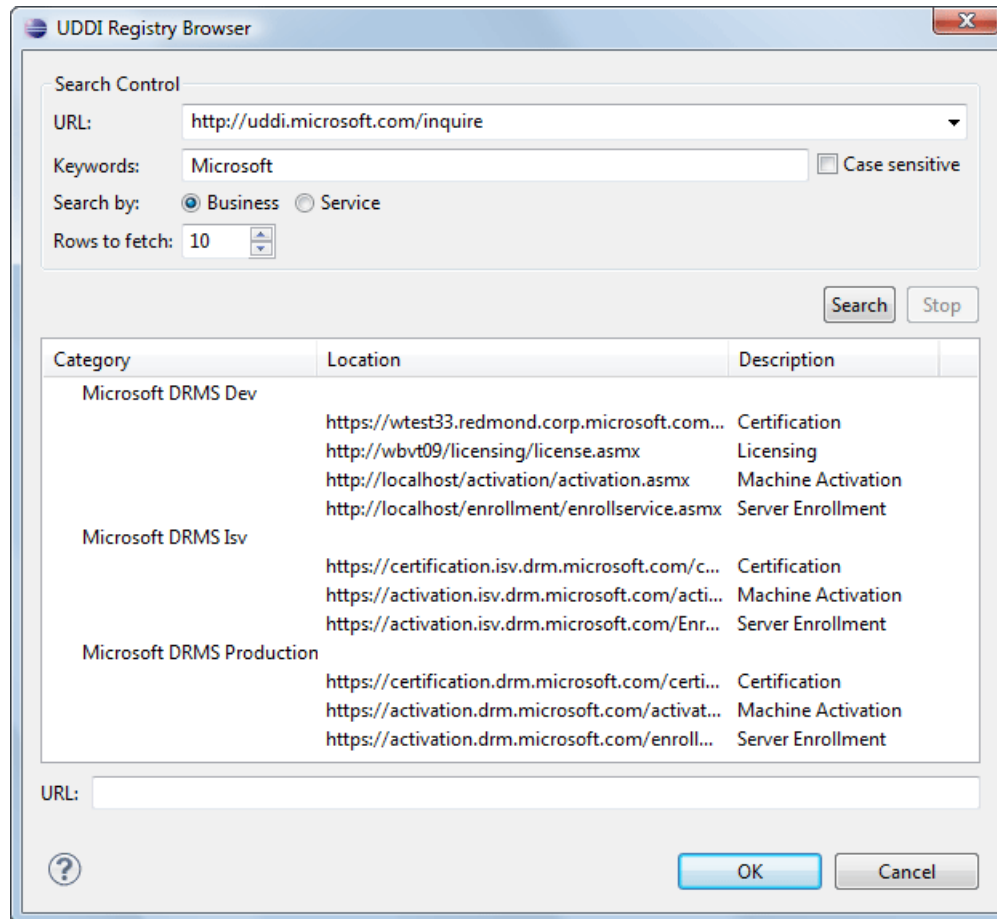


Figure 233: UDDI Registry Browser Dialog Box

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

Oxygen XML Developer plugin includes a built-in editor for CSS stylesheets. This section presents the features of the CSS editor and how these features should be used. The features of the CSS editor include:

- **Create new CSS files and templates** - You can use the built-in new file wizards to [create new CSS documents or templates](#).
- **Open and Edit CSS files** - CSS files can be opened and edited in a source editing mode.
- **Validation** - Presents validation errors in CSS files.
- **Content completion** - Offers proposals for properties and the values that are available for each property.
- **Syntax highlighting** - The syntax highlighting in Oxygen XML Developer plugin makes CSS files more readable.
- **Shortcut to open resources** - You can use **Ctrl + Single-Click (Command + Single-Click on OS X)** to open imported stylesheets or other resources (such as images) in the default system application for the particular type of resource.

Validating CSS Stylesheets

Oxygen XML Developer plugin includes a built-in *CSS Validator*, integrated with general validation support. This makes the *usual validation features* for presenting errors also available for CSS stylesheets.

The CSS properties accepted by the validator are those included in the current CSS profile that is selected in *the CSS validation preferences*. The **CSS 3 with Oxygen extensions** profile includes all the CSS 3 standard properties and the CSS extensions specific for Oxygen. That means all Oxygen specific extensions are accepted in a CSS stylesheet by *the built-in CSS validator* when this profile is selected.

Specify Custom CSS Properties

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_INSTALL_DIR]/lib` and create the `endorsed` folder. Copy the `custom_props.jar` file to `[OXYGEN_INSTALL_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.

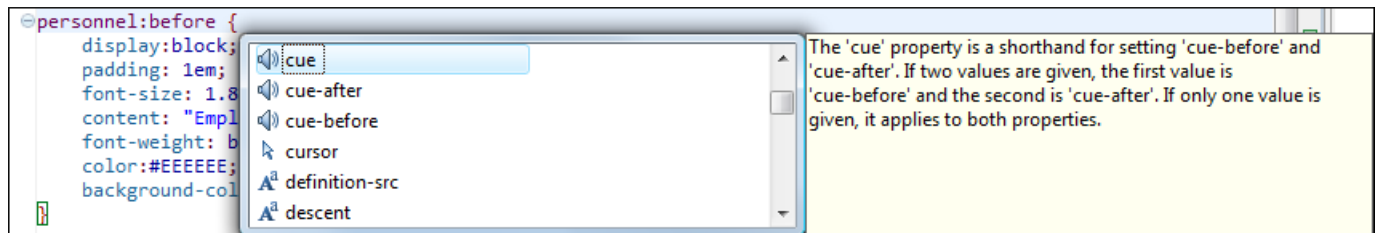


Figure 234: Content Completion in CSS Stylesheets

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.

Proposals for CSS Selectors - After inserting a *CSS selector*, the content completion assistance will propose a list of pseudo-elements and pseudo-classes that are available for the selected CSS profile.

Proposals for @media and @import Rules - After inserting `@media` or `@import` `<url>` rules, the content completion assistance will propose a list of supported media types.

Related tasks

Specify Custom CSS Properties on page 356

CSS Outline View

The **Outline** view for CSS stylesheets 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 Developer plugin highlights the corresponding import or selector in the CSS editor.

By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

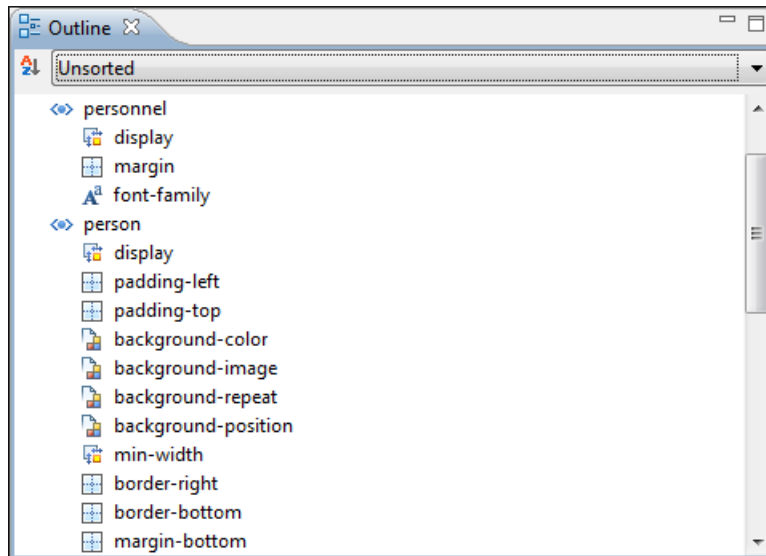


Figure 235: CSS Outline View

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 Developer plugin opens a dialog box that allows you to:

- Set the location of the resulting CSS.

- Place each style rule on a new line.

After pressing **OK**, Oxygen XML Developer plugin performs the following actions:

- All spaces are normalized (all leading and trailing spaces are removed, while sequences of white spaces are replaced with single space characters).
- All comments are removed.



Note: The CSS minifier relies heavily upon the W3C CSS specification. If the content of the CSS file you are trying to minify does not conform with the specifications, an error dialog box will be displayed, listing all errors encountered during the processing.

The resulting CSS stylesheet gains a lot in terms of execution performance, but loses in terms of readability. The source CSS document is left unaffected.



Note: To restore the readability of a minified CSS, invoke the **Format and Indent** action from the **XML** menu, the **Source** submenu from the contextual menu, or **Source** toolbar. However, this action will not recover any of the deleted comments.

Editing Relax NG 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.

Oxygen XML Developer plugin offers support for editing Relax NG schema files in the following editing modes:

- **Text editing mode** - Allows you to edit Relax NG schema files in a source editing mode, along with a schema design pane with two tabs that offer a **Full Model View** and **Logical Model View**.
- **Grid editing mode** - Displays Relax NG schema files in a structured spreadsheet-like grid.

For information about applying and detecting schemas, see [Associating a Schema to XML Documents](#) on page 211.

Related information

[Associating a Schema to XML Documents](#) on page 211

Editing Relax NG Schema in the Master Files Context

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

You can set a main Relax NG document either using the [master files support from the Navigator view](#), or using a validation scenario.

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

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

- Correct validation of a module in the context of a larger schema structure.
- **Content Completion Assistant** displays all the referable components valid in the current context. This includes components defined in modules other than the currently edited one.
- The **Outline** displays the components collected from the entire schema structure.

Related tasks

[Creating a New Validation Scenario](#) on page 199

Related information

[XML Schema Outline View](#) on page 131

Relax NG Schema Diagram Editor

This section explains how to use the graphical diagram editor for Relax NG schemas.

Introduction to Relax NG Schema Diagram Editor

Oxygen XML Developer plugin provides a simple, expressive, and easy-to-read schema diagram editor for Relax NG schemas.

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

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

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

Full Model View

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

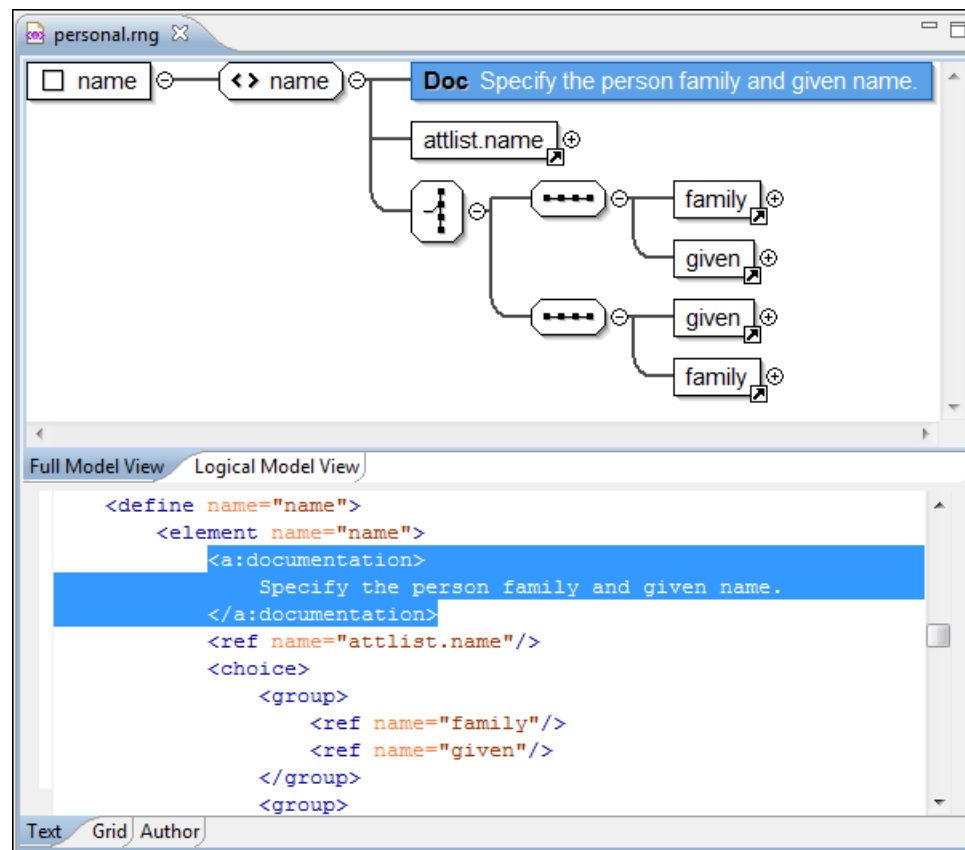


Figure 236: Relax NG Schema Editor - Full Model View

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 by double-clicking the names.

Logical Model View

The **Logical Model View** presents the compiled schema in the form of 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.

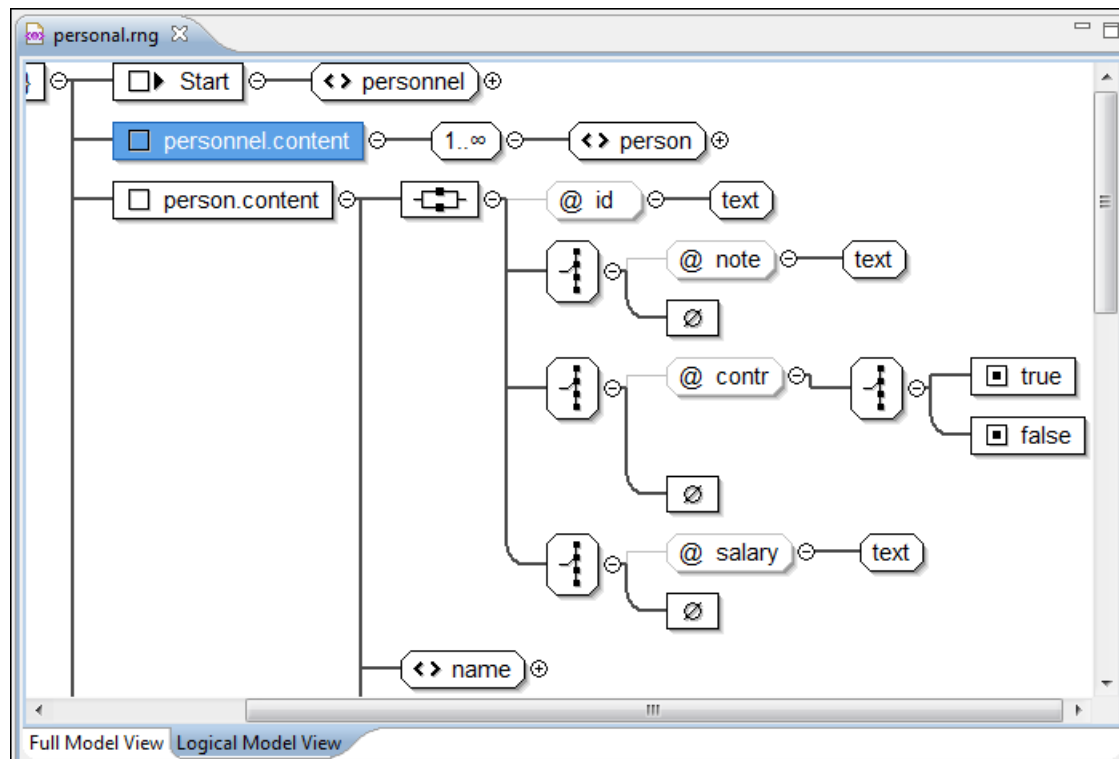
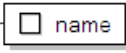
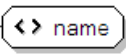
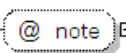
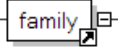
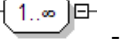
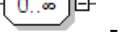
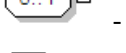
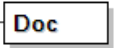


Figure 237: Logical Model View for a Relax NG Schema

Symbols Used in the Schema Diagram

The views in the schema diagram editor renders all the Relax NG schema patterns with the following intuitive symbols:

-  `name` - define pattern with the `name` attribute set to the value shown inside the rectangle (in this example `name`).
-  `attlist.person` - 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`).
-  `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`).
-  `name` - element pattern with the `name` attribute set to the value shown inside the rectangle (in this example `name`).
-  `@ note` - attribute pattern with the `name` attribute set to the value shown inside the rectangle (in this case `note`).
-  `family` - ref pattern with the `name` attribute set to the value shown inside the rectangle (in this case `family`).
-  `1..∞` - `oneOrMore` pattern.
-  `0..∞` - `zeroOrMore` pattern.
-  `0..1` - `optional` pattern.
-  - choice pattern.

-  - value pattern (for example, used inside a choice pattern).
-  - group pattern.
-  - A pattern from the Relax NG Annotations namespace (<http://relaxng.org/ns/compatibility/annotations/1.0>) that is treated as a documentation element in a Relax NG schema.
-  - text pattern.
-  - empty pattern.

Actions Available in the Schema Diagram Editor

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.

Validating Relax NG Schema Documents

By default, Relax NG schema files are 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 a Relax NG schema document manually, select the  **Validate** action from the  ▾ **Validation** toolbar drop-down menu or the **XML** menu. When Oxygen XML Developer plugin validates a Relax NG schema file, 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.

Related concepts

[Validation Scenario](#) on page 198

Related information

[Validating XML Documents Against a Schema](#) on page 194

[Associating a Schema to XML Documents](#) on page 211

[Presenting Validation Errors in Text Mode](#) on page 125

Relax NG Outline View

The **Outline** view for Relax NG schemas presents a list with the patterns that appear in the diagram in both the **Full Model View** and **Logical Model View** cases and it allows for quick access to a component by name. By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

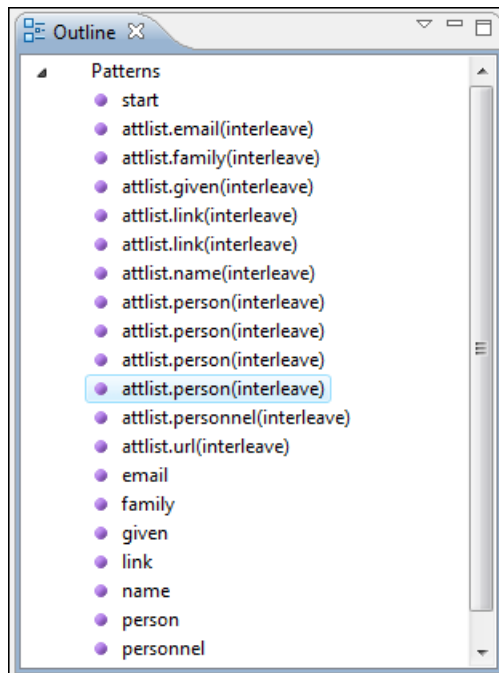


Figure 238: Relax NG Outline View

This view has two modes, with the tree showing either 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.

RNG Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view allows you to see the hierarchy/dependencies for a schema. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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

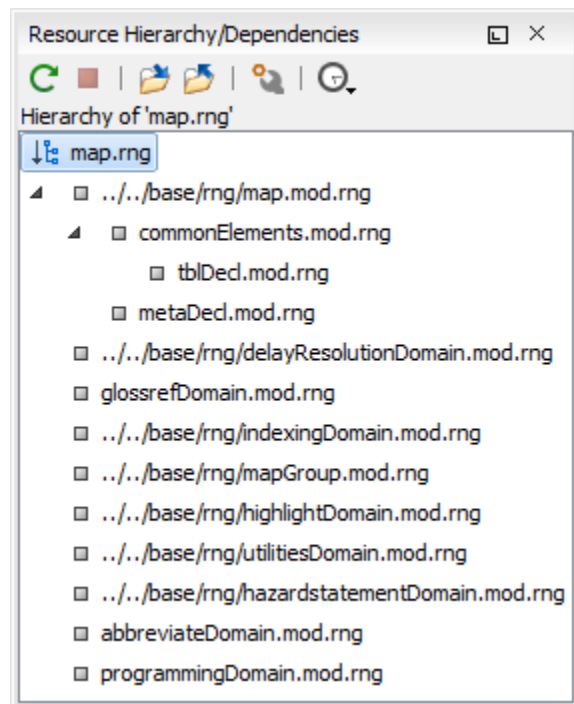


Figure 239: Resource Hierarchy/Dependencies View - hierarchy for map . rng

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

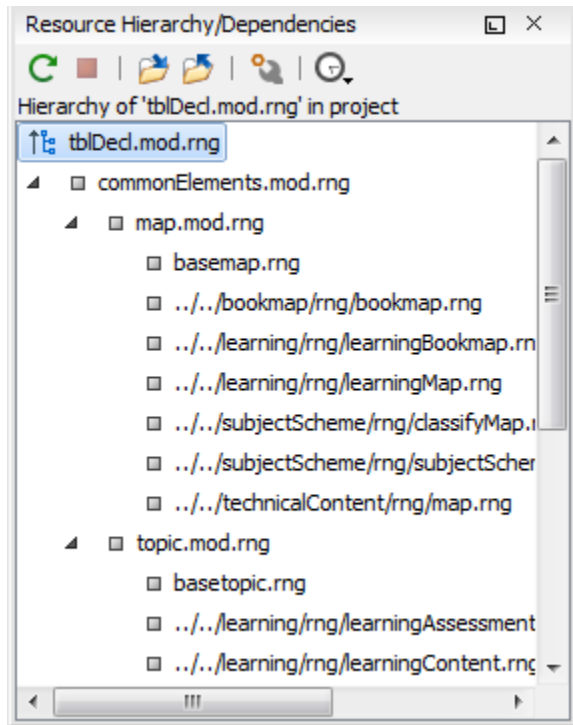


Figure 240: Resource Hierarchy/Dependencies View - Dependencies for tblDed.mod.rng

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*.

Related information

[Search and Refactor Operations Scope](#) on page 210

Moving/Renaming RNG Resources

You can 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. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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.

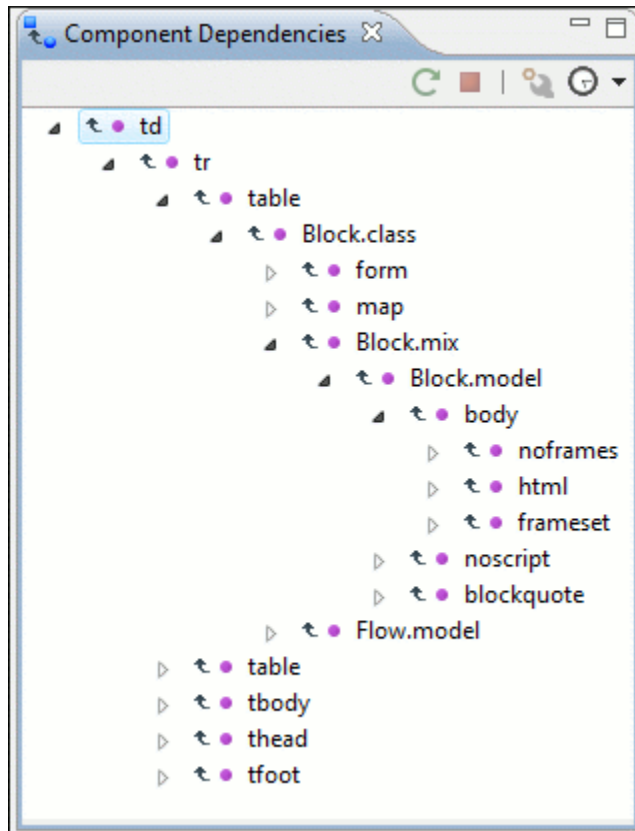


Figure 241: Component Dependencies View - Hierarchy for xhtml.rng

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 .

Related information

[Search and Refactor Operations Scope](#) on page 210

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 the in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in** - Opens a 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 to be affected by the action.

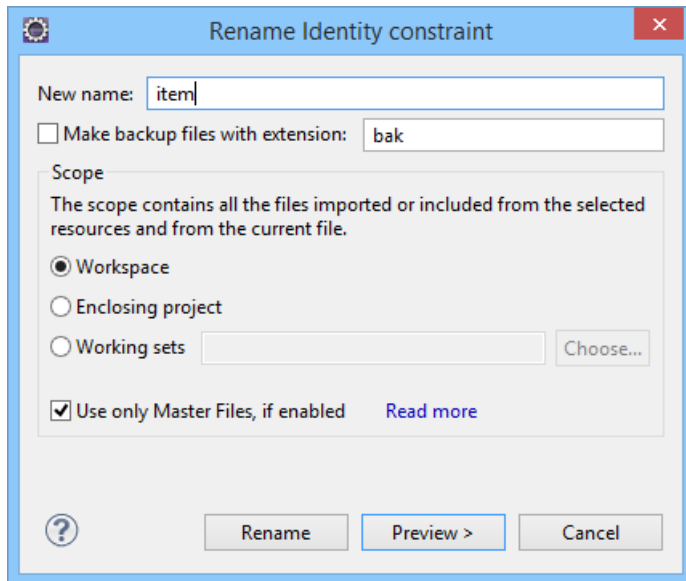


Figure 242: Rename Identity Constraint Dialog Box

Quick Fixes for DTD, XSD, and Relax NG Errors

Oxygen XML Developer plugin offers quick fixes for common errors that appear in XML documents that are validated against DTD, XSD, or Relax NG schemas.



Note: For XML documents validated against XSD schemas, the quick fixes are only available if you use the default Xerces validation engine.

Quick fixes are available in **Text** mode .

Oxygen XML Developer plugin provides quick fixes for numerous 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

Related information

[Schematron Quick Fixes \(SQF\)](#) on page 227

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 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 icon (💡) placed at the current line in the stripe on the left side of the editor. Also, you can invoke the quick assist menu by using the **Ctrl + 1** (**Meta 1** on Mac OS X) keyboard shortcuts.

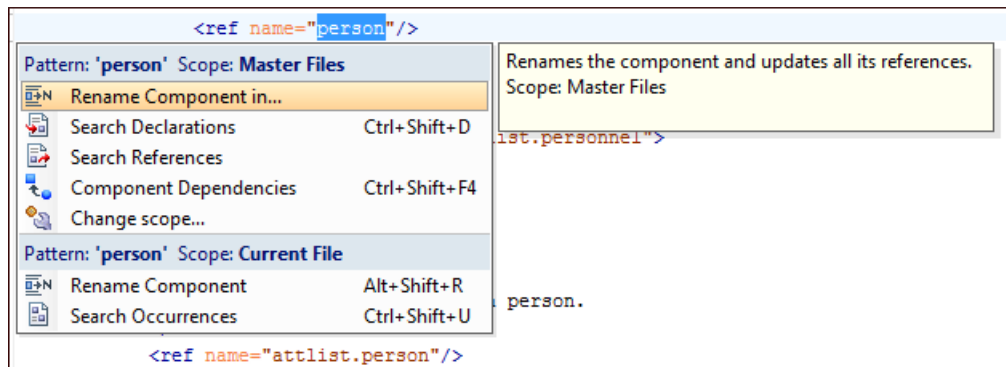


Figure 243: RNG Quick Assist Support

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.

Related information

[Component Dependencies View](#) on page 366

[Resource Hierarchy/Dependencies View](#) on page 364

[Searching and Refactoring Actions](#) on page 368

[Search and Refactor Operations Scope](#) on page 210

Configuring a Custom Datatype Library for a RELAX NG Schema

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

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

To load the custom library, restart the Eclipse platform.

Editing NVDL Schemas

Some complex XML documents are composed by combining elements and attributes from namespaces. Furthermore, 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 that must be taken into account for validation of the XML document or for content completion. An NVDL (Namespace Validation Definition Language) schema can be used. This schema allows the application to combine and interleave multiple schemas of different types (W3C XML Schema, RELAX NG schema, Schematron schema) in the same XML document.

Oxygen XML Developer plugin offers support for editing NVDL schema files in the following editing modes:

- **Text editing mode** - Allows you to edit NVDL schema files in a source editing mode, along with a schema design pane with two tabs that offer a **Full Model View** and **Logical Model View**.
- **Grid editing mode** - Displays NVDL schema files in a structured spreadsheet-like grid.

For information about applying and detecting schemas, see [Associating a Schema to XML Documents](#) on page 211.

Related information

[Associating a Schema to XML Documents](#) on page 211

NVDL Schema Diagram

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

Introduction to NVDL Schema Diagram Editor

Oxygen XML Developer plugin provides a simple, expressive, and easy-to-read schema diagram editor for NVDL schemas.

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

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

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

Full Model View

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

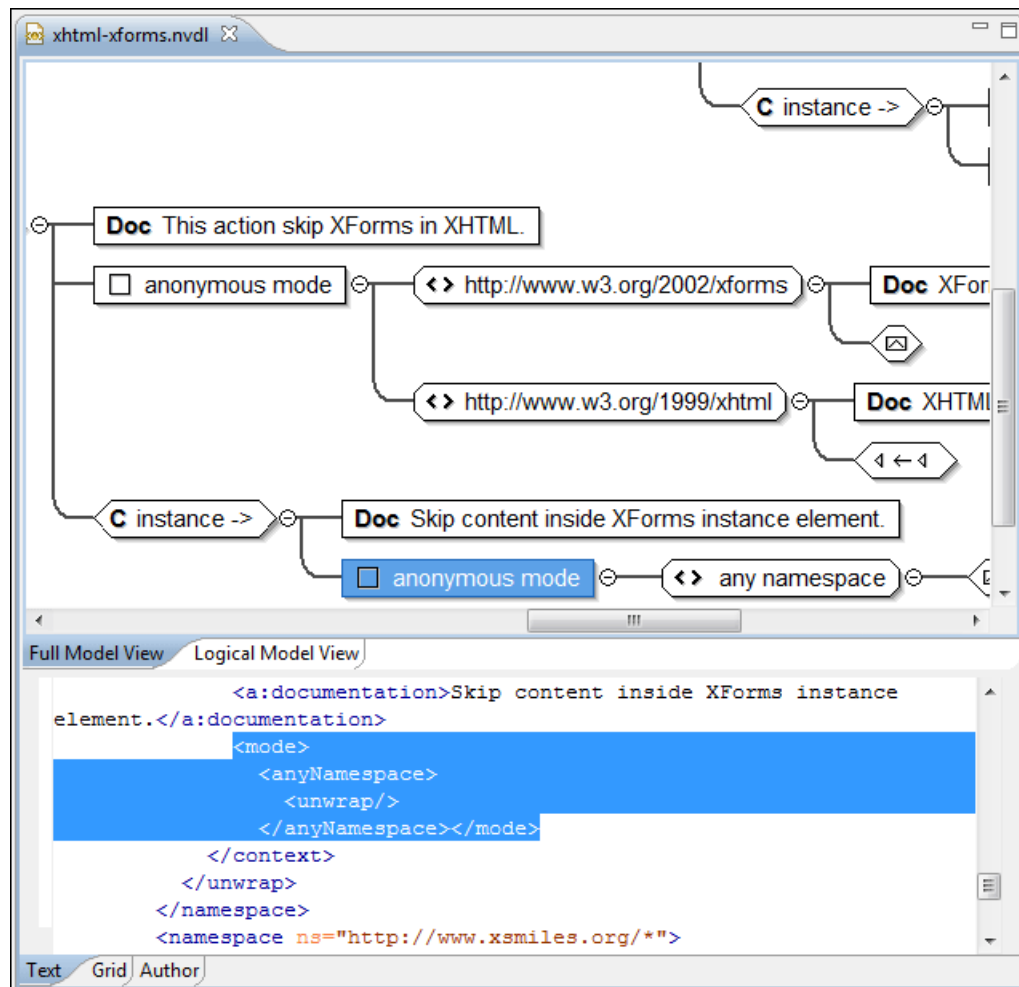


Figure 244: NVDL Schema Editor - Full Model View

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 to edit its properties.

Logical Model View

The **Logical Model View** presents the compiled schema in the form of 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.

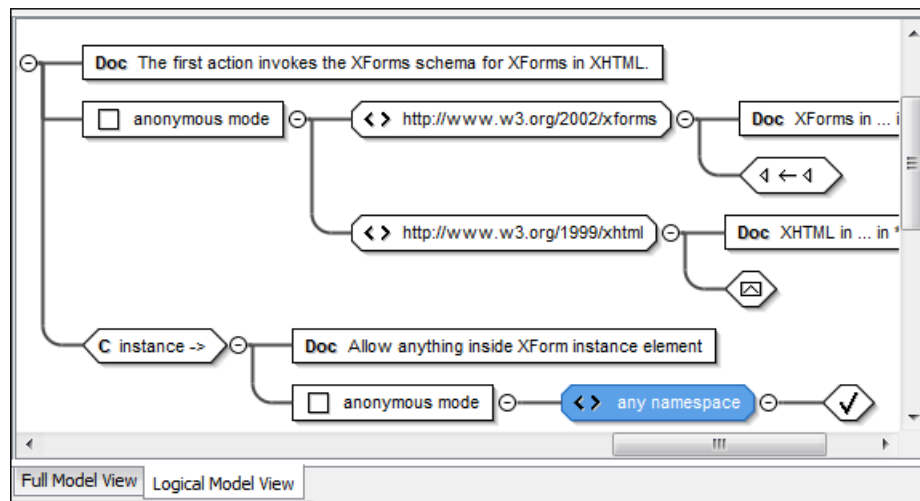


Figure 245: Logical Model View for an NVDL Schema

Actions Available in the Diagram Editor

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 **Outline** view for NVDL schemas presents a list with the named or anonymous rules that appear in the diagram and it allows for quick access to a rule by name. By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

Validating NVDL Schema Documents

By default, NVDL schema files are 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 an NVDL schema document manually, select the  **Validate** action from the  **Validation** toolbar drop-down menu or the **XML** menu. When Oxygen XML Developer plugin validates an NVDL schema file, 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.

Related information

[Validating XML Documents Against a Schema](#) on page 194

Component Dependencies View for NVDL Schemas

The **Component Dependencies** view allows you to see the dependencies for a selected NVDL named mode. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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.

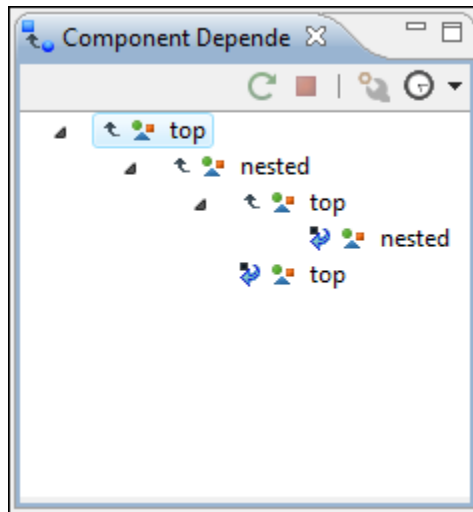


Figure 246: Component Dependencies View - Hierarchy for test.nvdl

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 .

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 the NVDL 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 the in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in** - Opens a 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 to be affected by the action.

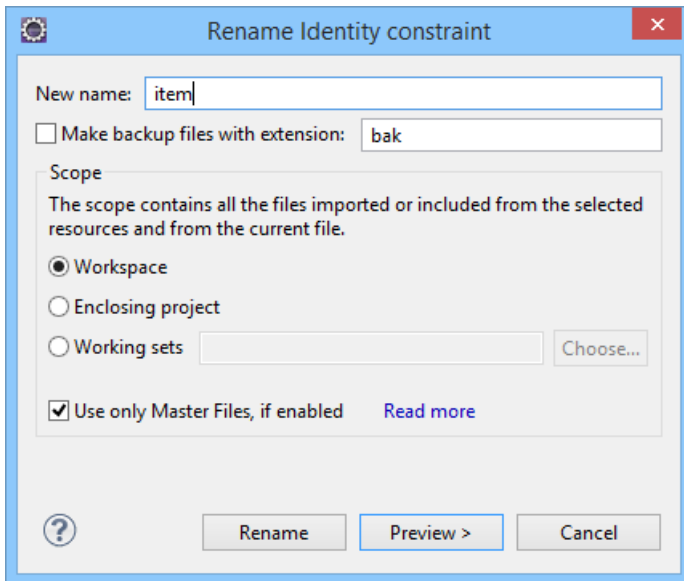


Figure 247: Rename Identity Constraint Dialog Box

Editing JSON Documents

This section explains the features of the Oxygen XML Developer plugin JSON Editor and how to use them.

Editing JSON Documents in Text Mode

When editing JSON documents in the **Text** editing mode, the *usual text editing actions* are available, along with other editor specific actions, including:

- *Find / Replace*
- *Drag and Drop*
- *Validation*
- *Format and Indent (Pretty Print)*



Note: You can run XPath expressions on opened JSON documents, but in **Text** mode the XPath results cannot be mapped in the document. However, they can be mapped in the **Grid** editing mode. You can use the **Grid** button at the bottom of the editor panel to switch to that editing mode.

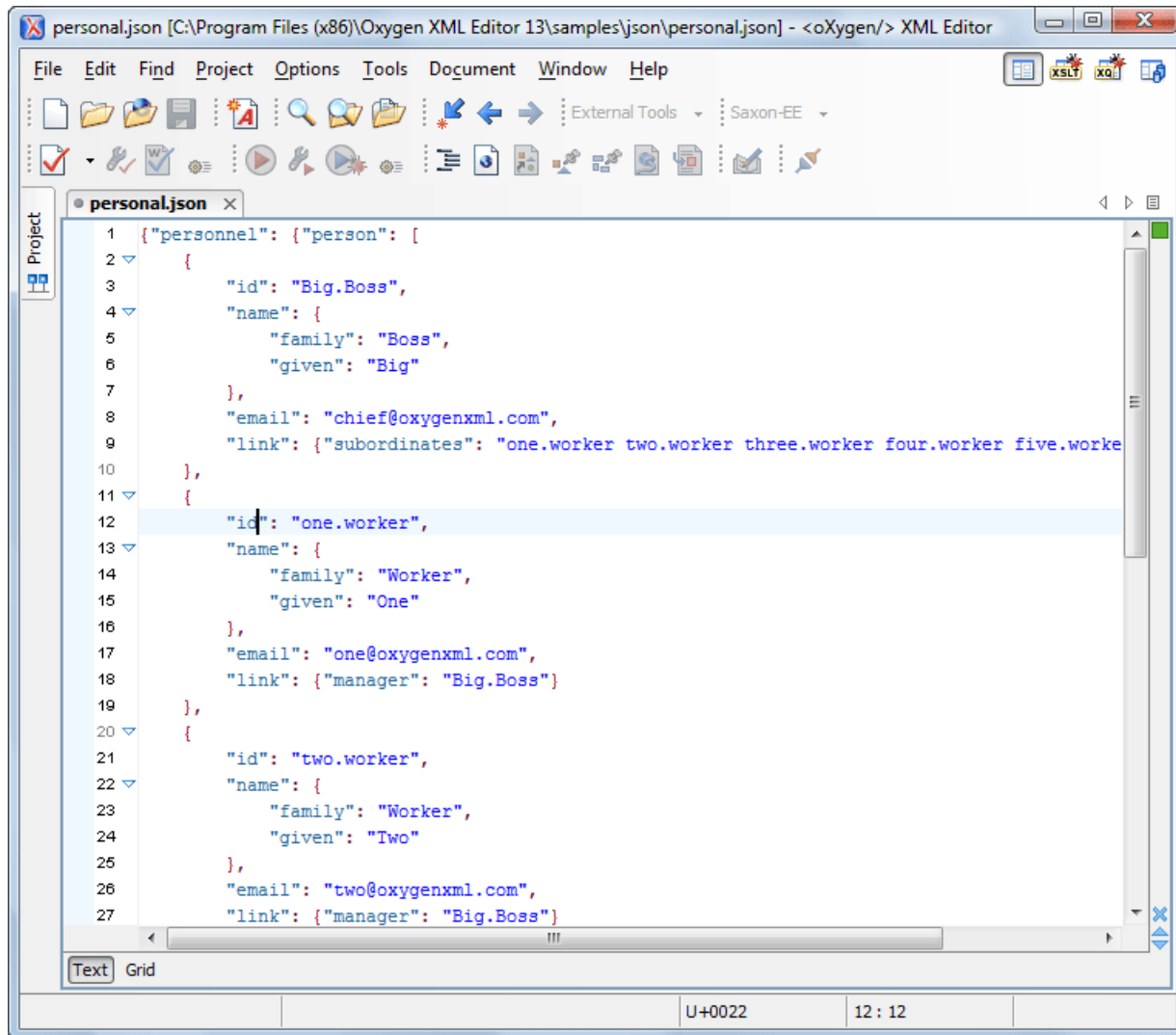


Figure 248: JSON Editor Text Mode

Related information

[Editing JSON Documents in Grid Mode](#) on page 378

Syntax Highlighting in JSON Documents

Oxygen XML Developer plugin supports syntax highlighting in JSON files to improve the readability of the content and reduce the time it takes to internalize the semantics of the structured content.

To customize the colors or styles used for the syntax highlighting colors for JSON files, follow these steps:

1. [Open the Preferences dialog box](#) .
2. Go to **Editor > Syntax Highlight**.
3. Select and expand the **JSON** section in the top pane.
4. Select the component you want to change and customize the colors or styles using the selectors to the right of the pane.

You can see the effects of your changes in the **Preview** pane.

Related information

[Syntax Highlight Preferences](#) on page 67

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

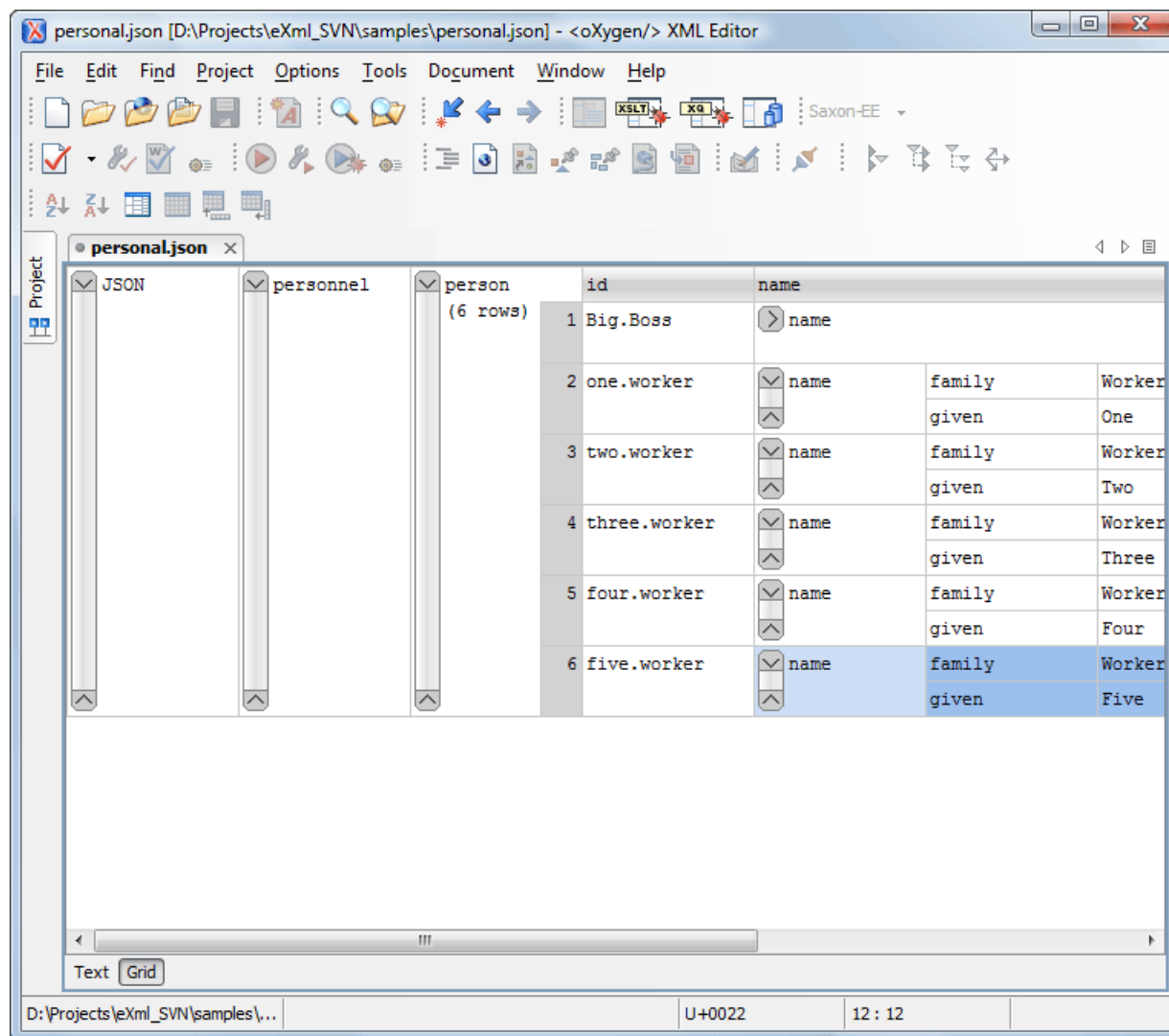


Figure 249: JSON Editor Grid Mode

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

Array

Useful when you want to convert a JSON *value* to *array*.

Insert value before

Inserts a value before the currently selected one.

Insert value after

Inserts a value after the currently selected one.

Append value as child

Appends a value as a child of the currently selected value.

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

Related information

[Grid Editing Mode](#) on page 126

[Grid Preferences](#) on page 57

Validating JSON Documents

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

Related information

[Presenting Validation Errors in Text Mode](#) on page 125

[Document Checking Preferences](#) on page 56

JSON Outline View

The **Outline** view for JSON documents displays the list of all the components of the JSON document you are editing. By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

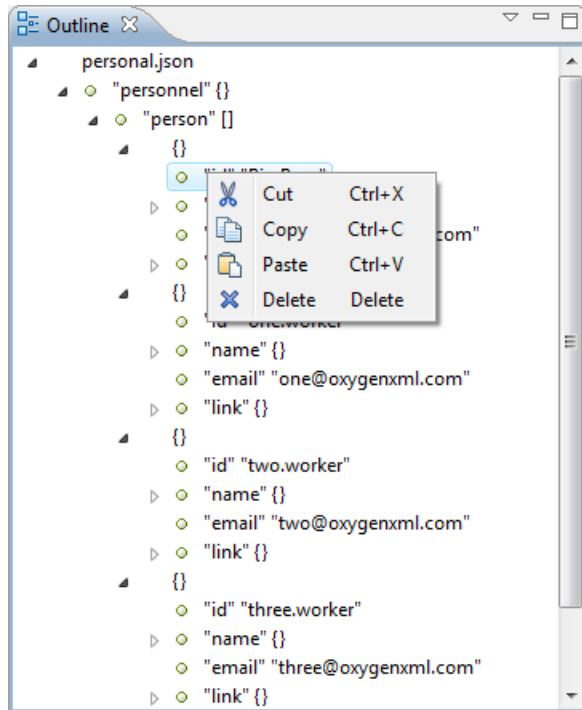


Figure 250: JSON Outline View

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 Developer 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.

XML to JSON Converter

Oxygen XML Developer plugin includes a useful and simple tool for converting XML files to JSON. It can be found in the **XML Tools** menu.

To convert an XML document to JSON, follow these steps:

1. Select the **XML to JSON** action from the **XML Tools** menu.

The **XML to JSON** dialog box is displayed:

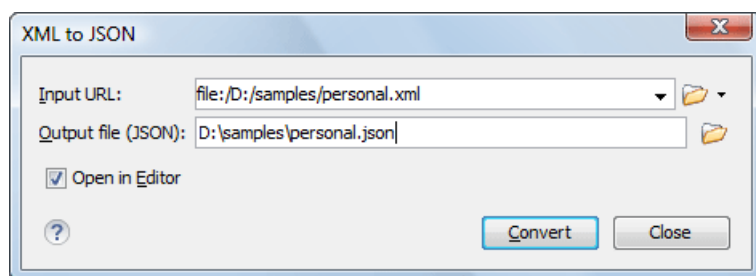


Figure 251: XML to JSON Dialog Box

2. Choose or enter the **Input URL** of the XML document.
3. Choose the path of the **Output file** that will contain the conversion JSON result.
4. Check the **Open in Editor** option to open the JSON result of the conversion in the Oxygen XML Developer plugin JSON Editor.
5. Click the **Convert** button.

The original XML document is now converted to a JSON document.

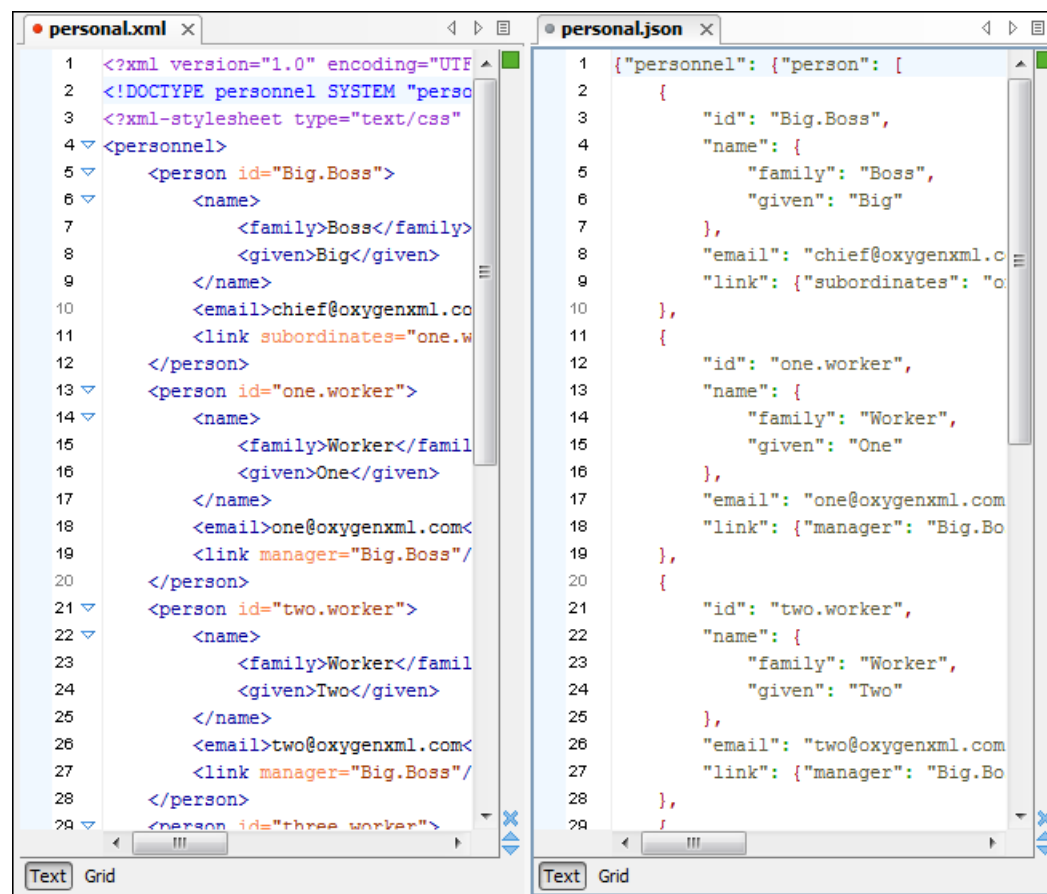


Figure 252: Example: XML to JSON Operation Result

Editing StratML Documents

Strategy Markup Language (StratML) is an XML vocabulary and schema for strategic plans. Oxygen XML Developer plugin supports StratML Part 1 (Strategic Plan) and StratML Part 2 (Performance Plans and Reports) and provides templates for the following documents:

- **Strategic Plan** (StratML Part 1)

- **Performance Plan** (StratML Part 2)
- **Performance Report** - (StratML Part 2)
- **Strategic Plan** - (StratML Part 2)

You can view the components of a StratML document in the **Outline** view. Oxygen XML Developer plugin implements a default XML with XSLT transformation scenario for this document type, called StratML to HTML.

Editing XLIFF Documents

XLIFF (*XML Localization 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 Developer 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_INSTALL_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_INSTALL_DIR]/frameworks/xliff/xsd/1.2`, and for version 2.0 in `[OXYGEN_INSTALL_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_INSTALL_DIR]/frameworks/xliff/xsd/2.0/modules`.

Editing JavaScript Documents

This section explains the features of the Oxygen XML Developer plugin JavaScript Editor and how you can use them.

JavaScript Editing Actions

Oxygen XML Developer plugin allows you to create and edit JavaScript files and assists you with useful features such as syntax highlight, content completion, and outline view. To enhance your editing experience, you can select entire blocks (parts of text delimited by brackets) by double-clicking somewhere inside the brackets.

```

12 function newPage(filename, overlay) {
13     divs = document.getElementsByTagName("div");
14
15     if (divs) {
16         var xdiv = divs[0];
17
18         if (xdiv) {
19             var xid = xdiv.getAttribute("id");
20
21             var mytoc = window.top.frames[0];
22             if (mytoc.lastUnderlined) {
23                 mytoc.lastUnderlined.style.textDecoration = "none";
24             }
25
26             var tdiv = xbGetElementById(xid, mytoc);
27
28             if (tdiv) {
29                 var ta = tdiv.getElementsByTagName("a").item(0);
30                 ta.style.textDecoration = "underline";
31                 mytoc.lastUnderlined = ta;
32             }
33         }
34     }
35
36     if (overlay != 0) {
37         overlaySetup('lc');
38     }

```

Figure 253: 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 Developer 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 that 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 **Compare Files** tool to compare the file you are editing with a file you choose in the dialog box.

Validating JavaScript Files

You have the possibility to validate the JavaScript document you are editing. Oxygen XML Developer plugin uses the Mozilla Rhino library for validation. For more information about this library, go to <http://www.mozilla.org/rhino/doc.html>. The JavaScript validation process checks for errors in the syntax. Calling a function that is not defined is not treated as an error by the validation process. The interpreter discovers this error when executing the faulted line. Oxygen XML Developer plugin can validate a JavaScript document both on-request and automatically.

Content Completion in JavaScript Documents

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:

-  - function
-  - variable
-  - object
-  - property
-  - method



Note: These icons decorate both the elements from the content completion list of proposals and from the **Outline** view.

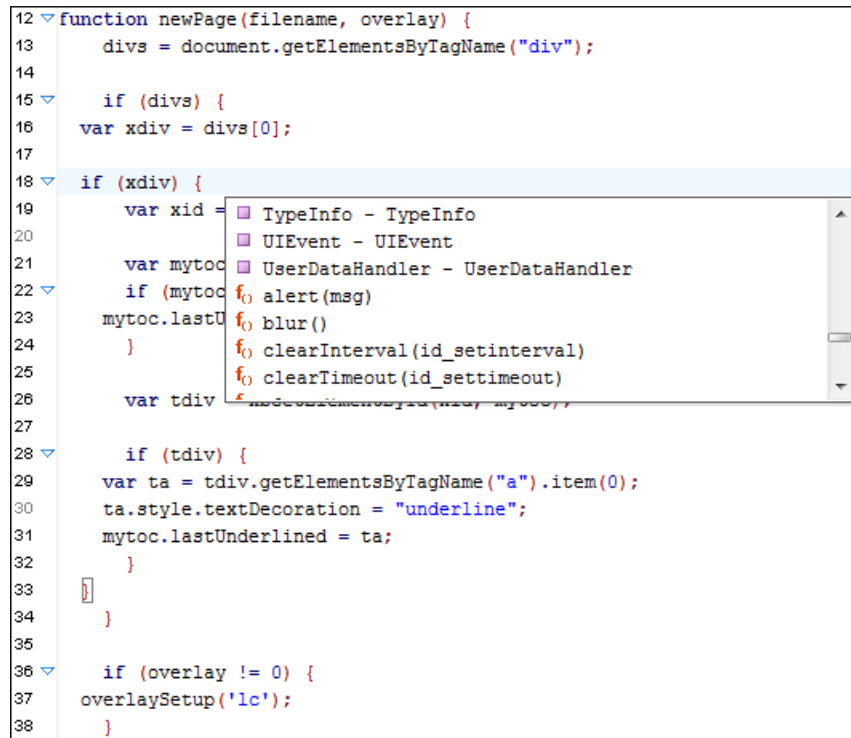


Figure 254: JavaScript Content Completion Assistant

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 Developer plugin present a list of all the components of the JavaScript document you are editing in the **Outline** view. By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

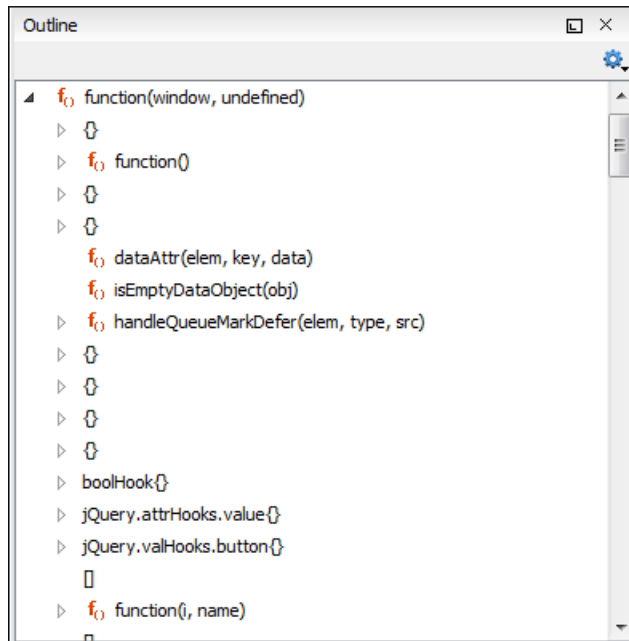


Figure 255: JavaScript Outline View

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.

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 that comes bundled with Oxygen XML Developer plugin version 18.1.

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 list of content completion proposals 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**.

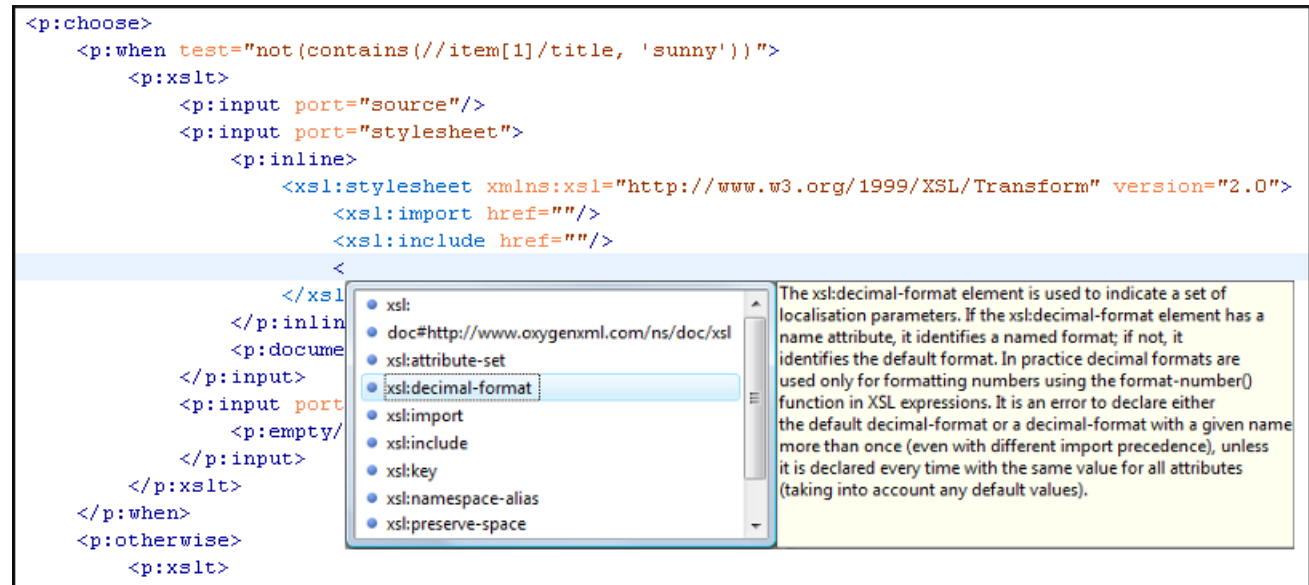


Figure 256: XProc Content Completion

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.

Oxygen XML Developer plugin assists you in editing Schematron documents with schema-based content completion, syntax highlight, search and refactor actions, and dedicated icons for the **Outline** view. You can create a new Schematron schema using one of the Schematron templates available in the [New from Templates wizard](#).

For information about applying and detecting Schematron schemas, see [Associating a Schema to XML Documents](#) on page 211.

Validating XML Documents Against Schematron

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.

How to Specify the Query Language Binding

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](#).

Multi-Lingual Support in Schematron Messages

You can specify the desired language for the validation messages in [the Schematron Preferences page](#). The Schematron validation messages can be presented in multiple languages by defining the language for each message using the Schematron `diagnostics`. For more information, see the [Use of Schematron for Multi-Lingual Schemas](#) specification.

How to Customize Color Schemes in Schematron

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**.

Schematron Transformation Scenario

When you create a Schematron document, Oxygen XML Developer plugin provides a built-in transformation scenario. You can 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.

Related information

[Editing XML Documents in Text Mode](#) on page 161

[Associating a Schema to XML Documents](#) on page 211

Editing Schematron Schema in the Master Files Context

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

You can set a main Schematron document either using the [master files support from the Navigator view](#), or using a validation scenario.

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

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

- Correct validation of a module in the context of a larger schema structure.
- **Content Completion Assistant** displays all the referable components valid in the current context. This includes components defined in modules other than the currently edited one.

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 a Schematron document manually, select the  **Validate** action from the  **Validation** toolbar drop-down menu or the **XML** menu. When Oxygen XML Developer plugin validates a Schematron schema, it expands all the included modules so the entire schema hierarchy is validated. The validation problems are highlighted directly in the editor, making it easy to locate and fix any issues.

Oxygen XML Developer plugin offers an error management mechanism capable of pinpointing errors in XPath expressions and in the included schema modules.

Related information

[Presenting Schematron Validation Issues](#)

Presenting Schematron Validation Issues

The possible issues that might occur during the validation process when validating XML documents against Schematron are presented with colored underlines in the editing pane, colored markers in the right vertical stripe, and details about the issues are presented in the **Problems** tab at the bottom area of the Oxygen XML Developer plugin window. Each error is flagged with a severity level that can be: *warning*, *error*, *fatal* or *info*.

To set a severity level, Oxygen XML Developer plugin looks for the following information:

- The **role** attribute, which can have one of the following values:
 - **warn** or **warning** - Sets the severity level to *warning*. By default, underlined with a yellow squiggly line in the editing pane and a yellow marker in the right vertical stripe.

- **error** - Sets the severity level to *error*. By default, underlined with a red squiggly line in the editing pane and a red marker in the right vertical stripe.
- **fatal** - Sets the severity level to *fatal*. By default, underlined with a red squiggly line in the editing pane and a red marker in the right vertical stripe.
- **info** or **information** - Sets the severity level to *info*. By default, underlined with a blue squiggly line in the editing pane and a blue marker in the right vertical stripe.
- The start of the message, after trimming leading white-spaces. Oxygen XML Developer plugin looks to match the following exact string of characters (case sensitive):
 - **Warning:** - Sets the severity level to *warning*. By default, underlined with a yellow squiggly line in the editing pane and a yellow marker in the right vertical stripe.
 - **Error:** - Sets the severity level to *error*. By default, underlined with a red squiggly line in the editing pane and a red marker in the right vertical stripe.
 - **Fatal:** - Sets the severity level to *fatal*. By default, underlined with a red squiggly line in the editing pane and a red marker in the right vertical stripe.
 - **Info:** - Sets the severity level to *info*. By default, underlined with a blue squiggly line in the editing pane and a blue marker in the right vertical stripe.
- If none of the previous rules match, Oxygen XML Developer plugin sets the severity level to *error*. By default, underlined with a red squiggly line in the editing pane and a red marker in the right vertical stripe.

Related concepts

[Validation Scenario](#) on page 198

Related information

[Validating XML Documents Against a Schema](#) on page 194

[Associating a Schema to XML Documents](#) on page 211

[Presenting Validation Errors in Text Mode](#) on page 125

Content Completion in Schematron Documents

Oxygen XML Developer plugin helps you edit a Schematron schema through the **Content Completion Assistant**, offering proposals 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 (such as `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:

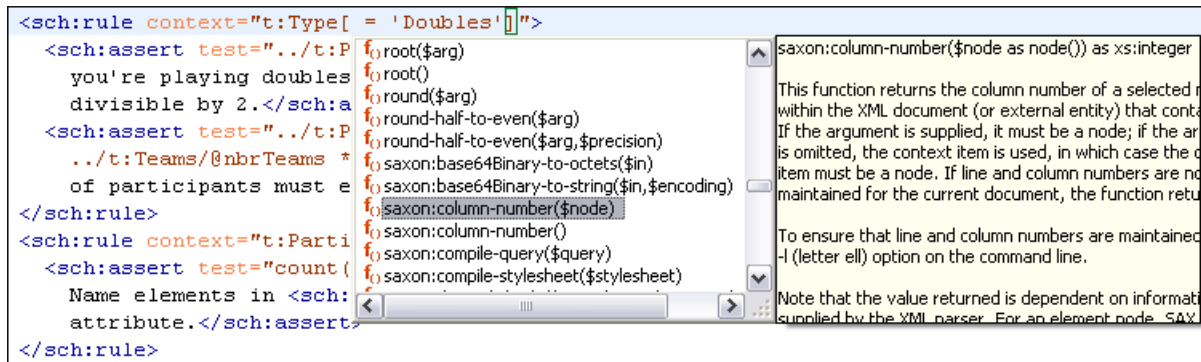


Figure 257: XSLT Extension Functions in Schematron Schema Content Completion

The **Content Completion Assistant** also includes *code templates that can be used to quickly insert code fragments* into Schematron documents.

XML Schema or RELAX NG with Embedded Schematron Rules

Schematron rules can be embedded into an XML Schema through annotations (using the `appinfo` element), or in any element on any level of a RELAX NG Schema (taking into account that the RELAX NG validator ignores all elements that are not in the RELAX NG namespace).

Oxygen XML Developer plugin accepts such documents as Schematron validation schemas and it is able to extract and use the embedded rules.

Validating XML Documents with Relax NG and Embedded Schematron

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.

Validating XML Documents with XML Schema and Embedded Schematron

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 Developer plugin provides two built-in validation scenarios: **Validate XML Schema with embedded Schematron** for XML schema, and **Validate Relax NG with embedded Schematron** for Relax NG. You can use one of these scenarios to validate the embedded Schematron rules.

Example: Embedded Schematron in Relax NG Schema

```
<sch:pattern>
  <sch:rule context="...">
    <sch:assert test="...">Message.</sch:assert>
  </sch:rule>
</sch:pattern>
```

Example: Embedded Schematron in XML Schema

```
<xsd:appinfo>
  <sch:pattern>
    <sch:rule context="...">
      <sch:assert test="...">Message.</sch:assert>
```

```
</sch:rule>
</sch:pattern>
</xsd:appinfo>
```

Related information

[Embed Schematron Quick Fixes in Relax NG or XML Schema](#) on page 403

Schematron Outline View

The **Outline** view for Schematron schemas presents a list of components in a tree-like structure and it allows for quick access to a component by name. By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

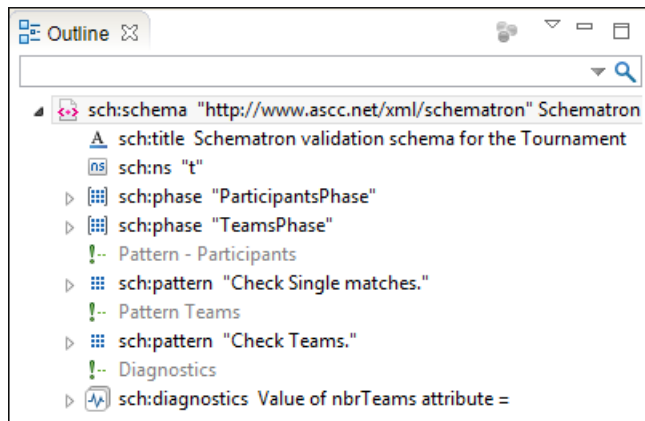


Figure 258: Schematron Outline View

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. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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

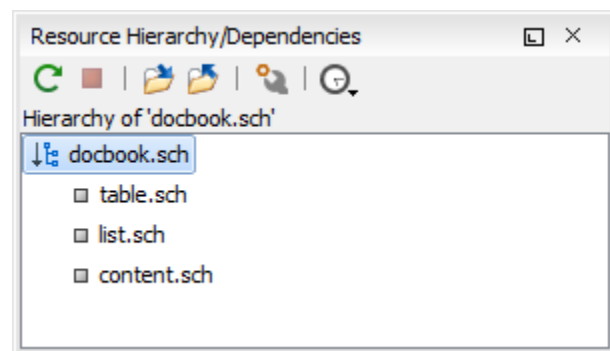


Figure 259: Resource Hierarchy/Dependencies View

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

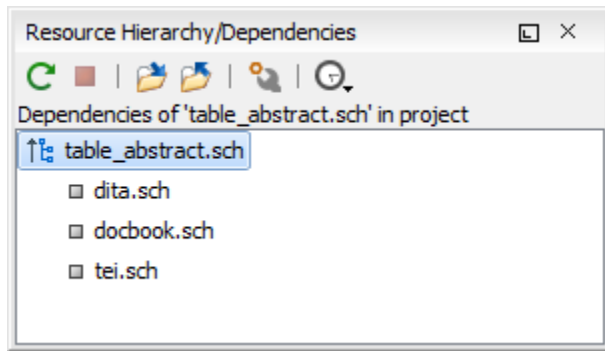


Figure 260: Resource Hierarchy/Dependencies View - Dependencies for table_abstract.sch

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 can 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 Developer plugin searches for the component declaration and all its references and highlights them automatically.

Customizable colors are used: one for the component definition and another one for component references. Occurrences are displayed until another component is selected.

To change the default 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 (Command + Shift + U on OS X)** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.

Searching and Refactoring Operations in Schematron Documents

Search Actions

The following search actions can be applied on **pattern**, **phase**, or **diagnostic** types and are available from the **Search** submenu in the contextual menu of the current editor:

-  **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
-  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
-  **Search Occurrences in File** - Searches all occurrences of the item at the 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 the in-place editing, press the **Esc** or **Enter** key on your keyboard.
- **Rename Component in** - Opens a 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 to be affected by the action.

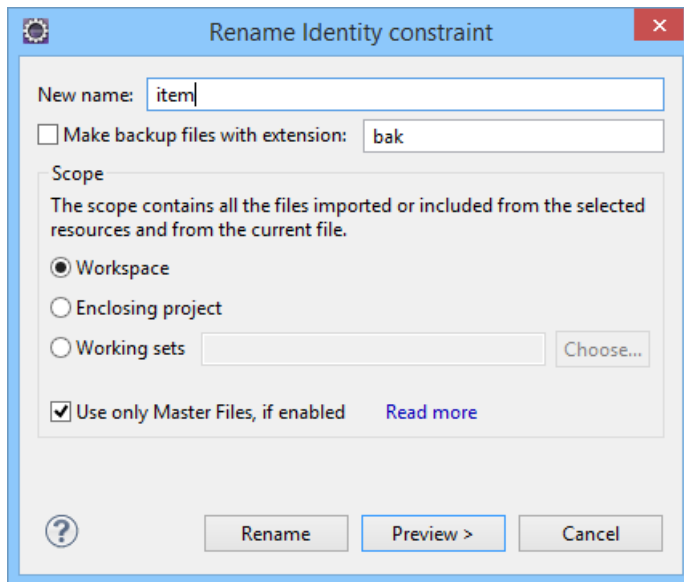


Figure 261: Rename Identity Constraint Dialog Box

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** checkbox 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](#).

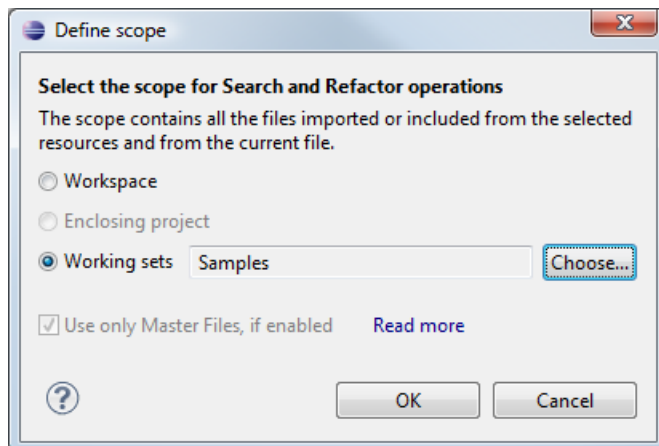


Figure 262: Define Scope Dialog Box

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* support improves the development work flow, offering fast access to the most commonly used actions when you edit 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 icon (💡) placed at the current line in the stripe on the left side of the editor. Also, you can invoke the quick assist menu by using the **Ctrl + 1** (**Meta 1** on Mac OS X) keyboard shortcuts.

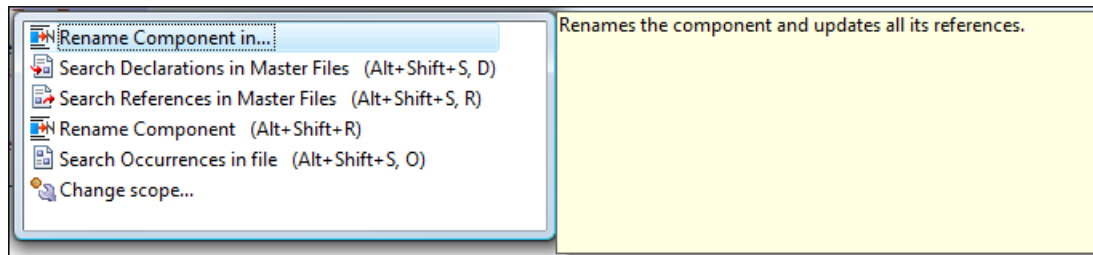


Figure 263: Schematron Quick Assist Support

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

Rename Component in

Renames the component and all its dependencies.

Search Declarations

Searches the declaration of the component in a predefined scope. It is available only when the context represents a component name reference.

Search References

Searches all references of the component in a predefined scope.

Component Dependencies

Searches the component dependencies in a predefined scope.

Change Scope

Configures the scope that will be used for future search or refactor operations.

Rename Component

Allows you to rename the current component in-place.

Search Occurrences

Searches all occurrences of the component within the current file.

Editing Schematron Quick Fixes

Oxygen XML Developer plugin provides support for editing the [Schematron Quick Fixes](#). You can define a library of quick fixes by editing them directly in the current Schematron file or in a separate file. Oxygen XML Developer plugin assists you in editing Schematron Quick Fixes with schema-based content completion, syntax highlighting, and validation as you type.

For information about applying and detecting the Schematron schemas that include SQF, see [Associating a Schema to XML Documents](#) on page 211.

Related information

[Oxygen XML Blog: Schematron Checks to Help Technical Writing](#)

Schematron Quick Fixes (SQF)

Oxygen XML Developer plugin provides support for Schematron Quick Fixes (SQF). They help you resolve issues 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 validation messages. Specifically, they are associated with *assert* or *report* messages.

A typical use case is using Schematron Quick Fixes to assist content authors 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 post: <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.

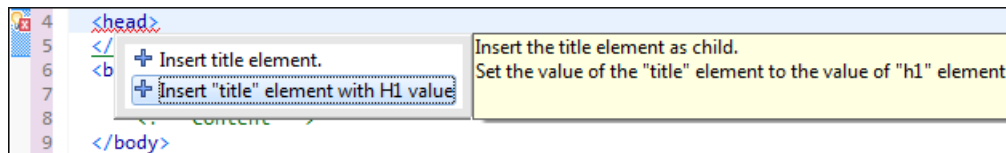


Figure 264: Example of a Schematron Quick Fix

Related information

[Editing Schematron Quick Fixes](#) on page 395

[Schematron Quick Fix Specifications](#)

[Presenting Schematron Validation Issues](#)

Defining Schematron Quick Fixes

You can define and customize Schematron Quick Fixes directly in the current Schematron file or in a separate Schematron 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 basic types of [operations \(elements\)](#) are supported:
 - `<sqf:add>` Element - To add a new node or fragment in the document.
 - `<sqf:delete>` Element - To remove a node from the document.
 - `<sqf:replace>` Element - To replace a node with another node or fragment.
 - `<sqf:stringReplace>` Element - To replace text content with other text or a fragment.

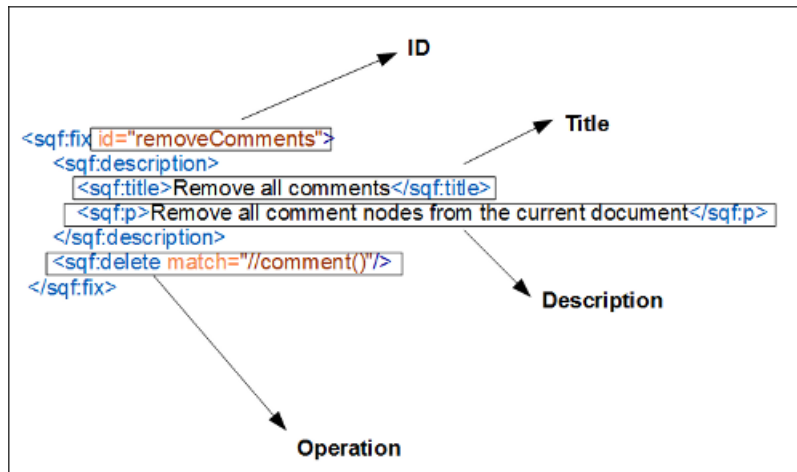


Figure 265: Schematron Quick Fix Components

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.

Additional Elements Supported in the Schematron Quick Fixes

<sqf:user-entry>

This element defines a value that must be set manually by the user. For more information, see [User Entry SQF Operation](#) on page 400.

<sqf:call-fix>

This element calls another quick fix within a quick fix. The called quick fix must be defined globally or in the same Schematron rule as the calling quick fix. A calling quick fix adopts the activity elements of the called quick fix and should not include other activity elements. You can also specify which parameters are sent by using the **<sqf:with-param>** child element.

<sqf:group>

Allows you to group multiple quick fixes and refer them from an **assert** or **report** element.

<sqf:fixes>

Is defined globally and contains global fixes and groups of fixes.

<sqf:keep>

Used to copy the selected nodes that are specified by the **select** attribute.



Note: In Oxygen XML Developer plugin the copied nodes cannot be manipulated by the current or other activity elements.

<sqf:param>

Defines a parameter for a quick fix. If the parameter is defined as **abstract** then the **type** and default value should not be specified and the fix can be called from an abstract pattern that defines this parameter.

Other SQF Notes



Note: The **sqf:default-fix** attribute is ignored in Oxygen XML Developer plugin.

For more details on editing Schematron Quick Fixes, go to: [Schematron Quick Fix Specifications](#)

Basic Schematron Quick Fix Operations

There are four basic operations that can be executed in a Schematron Quick Fix: *Add*, *Delete*, *Replace*, and *String Replace*.

Add

The `<sqf:add>` element allows you to add a node to the instance. An anchor node is required to select the position for the new node. The anchor node can be selected by the `match` attribute. Otherwise, it is selected by the `context` attribute of the rule.

The `target` attribute defines the name of the node to be added. It is required if the value of the `node-type` attribute is set to anything other than "comment".

The `<sqf:add>` element has a `position` attribute and it determines the position relative to the anchor node. The new node could be specified as the first child of the anchor node, the last child of the anchor node, before the anchor node, or after the anchor node (`first-child` is the default value). If you want to add an attribute to the anchor node, do not use the `position` attribute.



Note: If you insert an element and its content is empty, Oxygen XML Developer plugin will insert the required element content.

An Example of the `<sqf:add>` Element:

```
<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 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>
```

```

    </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: Consider the following information about using regular expressions in the `stringReplace` element:

- The regular expressions from this operation are compiled as Java regular expressions. For more information, see <https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>.
- The **j flag** allows you to use the standard Java regular expression engine, which allows native Java regular expression syntax. This allows, for example, the use of `\b` in a regular expression to match word boundaries. For more information, see <http://www.saxonica.com/html/documentation/functions/fn/matches.html>.
- Regular expressions in the `<sqf:stringReplace>` element always have 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(., '[o0][xX]ygen')">
        <sqf:fix id="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="[o0][xX]ygen"><ph keyref="product"/>
        </sqf:stringReplace>
      </sqf:fix>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

Related information

[User Entry SQF Operation](#) on page 400

[Restricting Quick Fix Operations](#) on page 400

User Entry SQF Operation

The <sqf:user-entry> element defines a value that must be set manually by the user. If multiple user-entry elements are defined, Oxygen XML Developer plugin will display a dialog box for each one, in which the user can specify values. Also, the <user-entry> element can be used as an XPath variable where the XPath variable is the name of the user-entry.

An Example of the <sqf:user-entry> Element:

```
<sqf:fix id="duplicate">
  <sqf:description>
    <sqf:title>Change the name of the element</sqf:title>
  </sqf:description>
  <sqf:user-entry name="newName" default="product">
    <sqf:description>
      <sqf:title>Enter the new name of the element</sqf:title>
    </sqf:description>
  </sqf:user-entry>
  <sqf:replace node-type="element" target="{ $newName }" select="child::node()"/>
</sqf:fix>
```

Related information

[Basic Schematron Quick Fix Operations](#) on page 397

Restricting Quick Fix Operations

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 $lastWidth"
  role="replace">
  <sqf:description>
    <sqf:title>Subtract excessive width from the last element.</sqf:title>
  </sqf:description>
  <let name="delta" value="$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>
```

Related information

[Basic Schematron Quick Fix Operations](#) on page 397

Formatting/Indenting Content Inserted by SQF Operations

Content that is inserted by the **Add**, **Replace**, or **String Replace** Schematron Quick Fix 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>
```

Related information

[Basic Schematron Quick Fix Operations](#) on page 397

Executing Schematron Quick Fixes in Other Documents

You can apply Schematron Quick Fixes over nodes from referenced documents (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">
```

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.

Related concepts

[Validation Scenario](#) on page 198

Related information

[Validating XML Documents Against a Schema](#) on page 194

[Presenting Validation Errors in Text Mode](#) on page 125

Content Completion in SQF

Oxygen XML Developer plugin helps you edit Schematron Quick Fixes embedded in a Schematron document by offering proposals that are valid at the cursor position in a *Content Completion Assistant*. 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 Developer plugin searches for the quick fix declaration and all its references and highlights them automatically.

Customizable colors are used: one for the quick fix definition and another one for its references. Occurrences are displayed until another quick fix is selected.

To change the default 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 (Command + Shift + U on OS X))** action from contextual menu. Matches are displayed in separate tabs of the **Results** view.

Searching and Refactoring Operations in SQF

Search Actions

The following search actions can be applied on quick fix ids and are available from the **Search** submenu in the contextual menu of the current editor:

-  **Search References** - Searches all references of the item found at current cursor position in the defined scope, if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box is displayed and you have the possibility to define another search scope.
- **Search References in** - Searches all references of the item found at current cursor position in the file or files that you specify when define a scope in the **Search References** dialog box.
-  **Search Declarations** - Searches all declarations of the item found at current cursor position in the defined scope if any. If a scope is defined, but the current edited resource is not part of the range of resources determined by this, a warning dialog box will be displayed and you have the possibility to define another search scope.
- **Search Declarations in** - Searches all declarations of the item found at current cursor position in the file or files that you specify when you define a scope for the search operation.
-  **Search Occurrences in File** - Searches all occurrences of the item at the 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 the in-place editing, press the **Esc** or **Enter** key on your keyboard.
-  **Rename Component in** - Opens a 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 to be affected by the action.

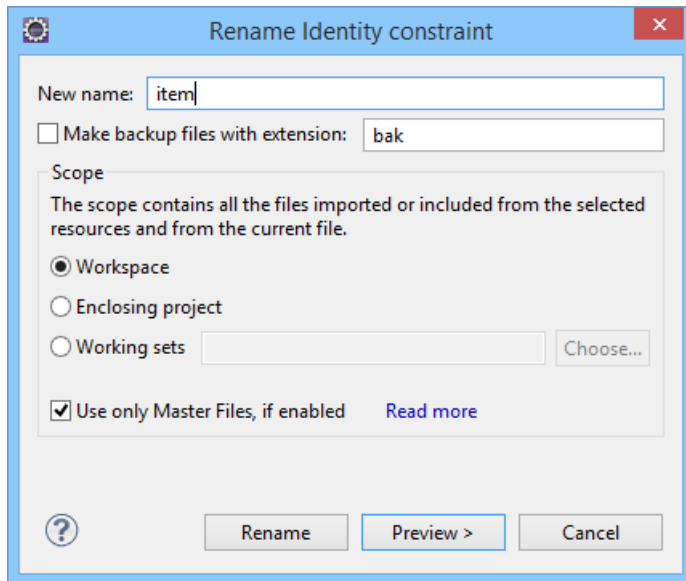


Figure 266: Rename Identity Constraint Dialog Box

Embed Schematron Quick Fixes in Relax NG or XML Schema

Schematron Quick Fixes can be embedded into an XML Schema through annotations (using the `appinfo` element), or in a schematron rule embedded in the RELAX NG Schema. For more information about embedding schematron in XML Schema or Relax NG, see [XML Schema or RELAX NG with Embedded Schematron Rules](#) on page 389.

Oxygen XML Developer plugin is able to extract and use the embedded Schematron Quick Fixes. To make the embedded Schematron Quick Fixes available, follow these steps:

1. Define a [validation against a schema](#).
2. For the **Schema type**, choose XML Schema or Relax NG.
3. Enable the **Embedded schematron rules** option.

Example: Embedded Schematron Quick Fix in XML Schema

```
<xsd:appinfo>
  <sch:pattern>
    <sch:rule context="...">
      <sch:assert test="..." sqf:fix="fixId">Message.</sch:assert>
      <sqf:fix id="fixId">
        .....
      </sqf:fix>
    </sch:rule>
  </sch:pattern>
</xsd:appinfo>
```

Example: Embedded Schematron Quick Fix in Relax NG

```
<sch:pattern>
  <sch:rule context="...">
    <sch:assert test="..." sqf:fix="fixId">Message.</sch:assert>
    <sqf:fix id="fixId">
      .....
    </sqf:fix>
  </sch:rule>
</sch:pattern>
```



Tip: For more extensive examples, see our samples in the `[OXYGEN_INSTALL_DIR]/samples/schematron` folder.

Related information

[XML Schema or RELAX NG with Embedded Schematron Rules](#) on page 389

[Defining Schematron Quick Fixes](#) on page 396

Integrating SQF in a Framework

You can use Schematron Quick Fixes to assist your content authors by imposing rules for an entire framework (document type) and offering fixes when a rule violation is detected.

For example, if you are using DITA, you may want your contributors to avoid inserting a figure (`fig` element) inside a paragraph (`p` element) that contains other content since it may result in undesirable placement or spacing in the output. The Schematron rule and its Quick Fix for this particular use-case could look like this:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
  xmlns:sqf="http://www.schematron-quickfix.com/validator/process"
  queryBinding="xslt2">
  <pattern id="check.figure.location">
    <rule context="p/fig">
      <report test="true()" role="warn" sqf:fix="moveAfter">
        A figure inside a paragraph doesn't transform well into PDF. </report>
      <sqf:fix id="moveAfter">
        <sqf:description>
          <sqf:title>Move after the paragraph.</sqf:title>
          </sqf:description>
          <let name="figToMove" value="."/>
          <sqf:add match="parent::p" select="$figToMove" position="after"/>
          <sqf:delete match="."/>
        </sqf:fix>
      </rule>
    </pattern>
  </schema>
```

The result of this example would be that the user will see a warning if they insert a `fig` element inside a `p` element and they are presented with the option of selecting the Quick Fix that would move the figure outside the paragraph.

How to Integrate SQF in a Framework

To integrate a Schematron Quick Fix in a framework, follow these steps:

1. *Define a Schematron Quick Fix for a rule* in an existing or new Schematron file.
2. Save it somewhere in your framework directory. For example, the default framework directory for DITA is located in: `[OXYGEN_INSTALL_DIR]/frameworks/dita/`.
3. Add a reference to the Schematron file that includes the SQF in your framework by following the procedure in *Associating a Schema in Validation Scenarios Defined in the Document Type* on page 214.
4. Open a document in your framework and test the new rule and quick fix.
5. You can continue to refine the rule and develop additional rules as needed.

Related information

Defining Schematron Quick Fixes on page 396

Basic Schematron Quick Fix Operations on page 397

Associating a Schema in Validation Scenarios Defined in the Document Type on page 214

Editing XHTML Documents

Oxygen XML Developer plugin provides support for editing XHTML files. Oxygen XML Developer plugin includes XHTML catalogs and document templates to help you get started. You can also find a sample file in the `[OXYGEN_INSTALL_DIR]/samples/xhtmll` folder.

The XHTML editing features include:

- **Source Editing** - You can edit XHTML files in the *Text editing mode* (XML source editor) using all of its *useful features*.
- **Validation** - Easily identify errors and their location with the Oxygen XML Developer plugin *XML validation features*.
- **Content Completion** - The **Content Completion Assistant** displays a list of context-sensitive proposals that are valid at the current cursor position.
- **Import HTML as XHTML** - Oxygen XML Developer plugin includes support for *importing HTML files as an XML document*.
- **Remote Editing** - Oxygen XML Developer plugin has built-in support for *editing documents that are stored on remote servers through FTP, SFTP, and WebDAV protocols*, allowing you to edit XHTML pages from your web server.
- **Syntax Highlighting** - 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**.

Editing Markdown Documents

Markdown was created as a lightweight markup language with plain text formatting syntax designed to provide a syntax that is very easy to read and write, and to convert it to HTML and other formats. It is often used by content contributors who want a quick and easy way to write content without having to take their fingers off the keyboard and without having to learn numerous codes or shortcuts, and it can easily be shared interchangeably between virtually any types of contributor and system.

Oxygen XML Developer plugin provides a built-in Markdown editor that allows you to convert Markdown syntax to HTML or DITA and it includes a preview panel to help you visualize the final output. Aside from the plain text syntax that is common amongst most Markdown applications, the editor in Oxygen XML Developer plugin also integrates many other powerful features that content authors are accustomed to using for other types of documents. Some of these additional unique features include:

- Additional toolbar and contextual menu actions.
- Automatic validation to help keep the syntax valid.
- Dedicated syntax highlighting to make Markdown documents even easier to read and write.
- Unique features for creating Markdown documents directly in DITA maps and converting Markdown documents to DITA topics.
- Specialized syntax rules to combine popular syntax features from several specifications.

Markdown Editor

Oxygen XML Developer plugin provides an intuitive, dynamic, and easy-to-use Markdown editor for writing and converting Markdown documents. It is a splitscreen editor with two panels that are synchronized in real-time. The left side is a simple text editor that is specially designed for writing Markdown syntax. The right side is a WYSIWYG style preview of how changes will look in the output.

Left-Side Markdown Text Editor

The left pane is a simple text editor that is refined to accept Markdown syntax. At the same time, you still have many of the actions, options, and features that you are used to when editing any other type of document in Oxygen XML Developer plugin.

The features of this special editor that are unique for Markdown documents include:

- **Unique Markdown Syntax Rules** - The Markdown editor in Oxygen XML Developer plugin uses [an integration of rules](#) that combine rules from common default Markdown syntax along with many of the rules used in the GitHub Flavored Markdown syntax.
- **Syntax Highlighting** - The Oxygen XML Developer plugin syntax highlighting feature is integrated into the Markdown text editor to make it easier to read and write Markdown syntax. You can even [customize the colors and styles for the syntax highlighting](#).
- **Automatic Spell Checking** - The Markdown editor supports the Oxygen XML Developer plugin [automatic spell checking feature](#) that reports possible misspelled words as you type. You simply need to enable the [Automatic spell check option in the Spell Check preferences page](#), then click the **Select editors** button and select **Markdown Editor**.
- **Helpful Toolbar and Contextual Menu Actions** - A [variety of unique actions](#) are available from the toolbar to help you insert proper Markdown syntax. The contextual menu also includes some common editing actions, as well as unique actions to export (convert) Markdown documents to HTML or DITA.

Right-Side WYSIWYG Preview Pane

The right pane is a WYSIWYG *Preview* pane that shows a visual representation of how changes made in the left-side text editor will be converted to HTML or DITA output. The changes you make in the text editor are parsed continually and they are immediately visible in the *Preview* pane. There are two tabs available in the *Preview* pane, one for visualizing DITA output and one for visualizing HTML output. You can switch between the two tabs at the bottom of the pane.

The *Preview* pane includes the following unique features:

- **WYSIWYG Visualization** - This pane presents the Markdown syntax from the left-side text editor in a visual WYSIWYG style interface that is automatically synchronized as you type.
- **Export Options** - The **DITA** tab includes a contextual menu action to *export (convert) the current Markdown document to a DITA topic*. Similarly, the **HTML** tab includes a contextual menu action to *export (convert) the Markdown document to HTML*.
- **Automatic Validation** - As you edit Markdown documents, they are *validated automatically*. The conversion engine constantly tries to parse your changes and if a change results in an error that prevents the parser from converting the syntax, an error message will be displayed in the *Preview* pane or **Results** view at the bottom of the editor.
- **Print Feature** - The Markdown editor includes a **Print** action that is available in the contextual menu and it allows you to configure options for printing the current document as you see it in the *Preview* pane.
- **Specialized DITA Features** - The Markdown editor includes some unique, *specialized features to integrate it with the powerful DITA support* in Oxygen XML Developer plugin.

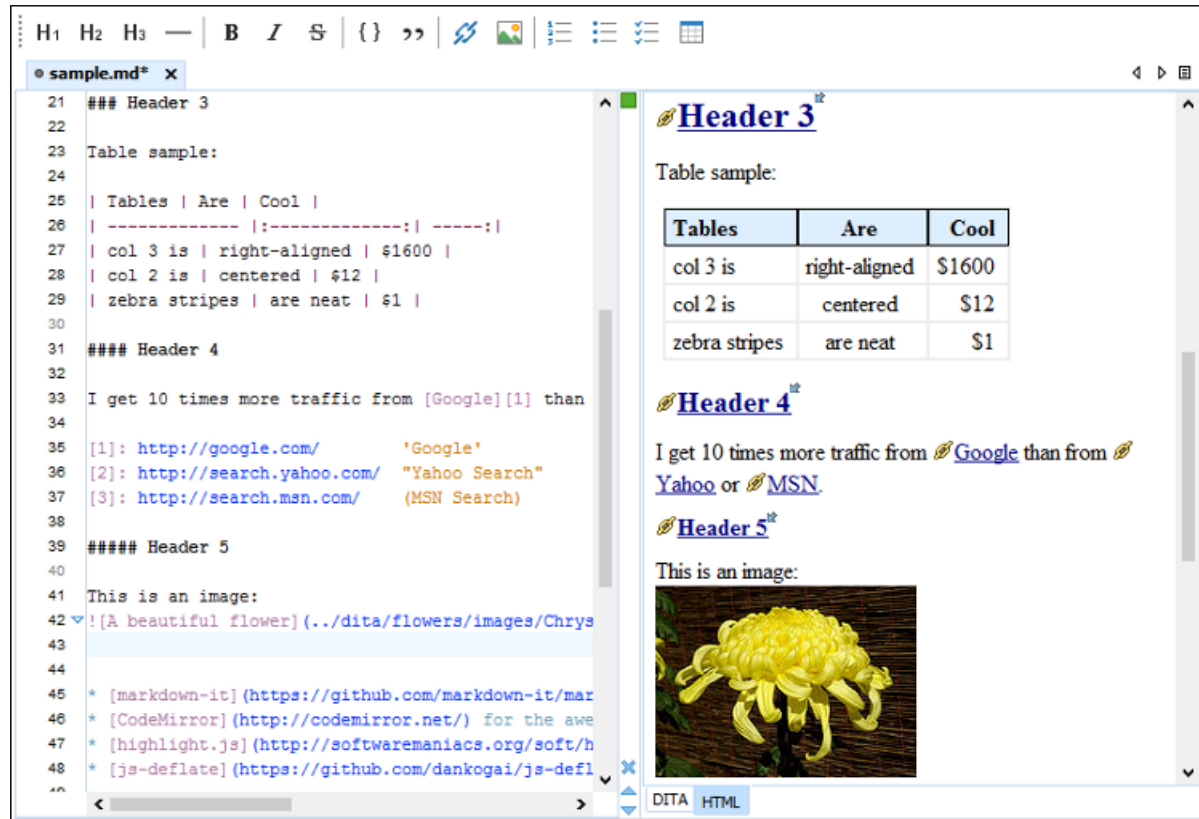


Figure 267: Markdown Editor in Oxygen XML Developer plugin

Related information

[Markdown Editor Syntax Rules and Specifications](#) on page 409

[Actions Available in the Markdown Editor](#) on page 407

[Working with Markdown Documents in DITA](#) on page 409

[Creating New Markdown Documents](#) on page 406

Creating New Markdown Documents

To create a new Markdown document in Oxygen XML Developer plugin, follow these steps:

1. Click the  **New** button on the toolbar or select **File > New**.
2. Select the  **Markdown** file template.
3. Click **Next**.
4. Choose the storage path and a file name for the new document.
5. Click the **Finish** button.

Result: A new Markdown document is created and it is opened in the specialized [Markdown Editor](#).

Related information

[Markdown Editor](#) on page 405

Actions Available in the Markdown Editor

Aside from the actions that are available in Oxygen XML Developer plugin for any type of document (such as the actions in the various menus and the common sections of the toolbar), a variety of unique actions are also available in the Markdown editor, from the toolbar and contextual menu.

Toolbar Actions

The following default actions are readily available on the toolbar when editing Markdown documents:

H1 Header (1st Level)

Inserts an *atx-style first level header* at the cursor position.

H2 Header (2st Level)

Inserts an *atx-style second level header* at the cursor position.

H3 Header (3rd Level)

Inserts an *atx-style third level header* at the cursor position.

Horizontal Rule

Inserts a *horizontal rule* at the cursor position.

B Bold (Strong)

Marks the selected text with *bold*.

I Italic (Emphasis)

Marks the selected text with *italics*.

Strikethrough

Marks the selected text with a *strikethrough*.

{ } Code Block

Inserts (or surrounds selected text in) a *codeblock*.

” Blockquote

Inserts a *blockquote* at the cursor position.

Insert Link

Opens the **Insert Link** dialog box that allows you to define a *link* to insert at the cursor position.

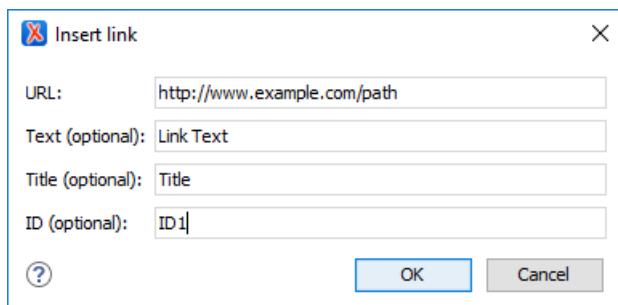


Figure 268: Insert Link Dialog Box

Insert Image

Opens the **Insert Image** dialog box that allows you to define an *image* to insert at the cursor position. You can type the URL of the image you want to insert or use browsing tools in the  **Browse** drop-down menu.

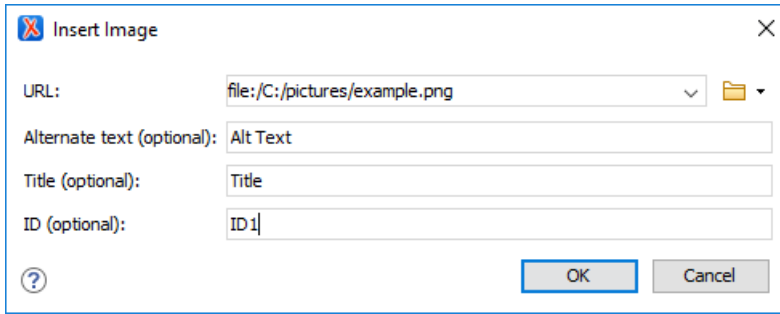


Figure 269: Insert Link Dialog Box

Insert Ordered List

Inserts an *ordered list* at the cursor position. Three child list items are also automatically inserted by default.

Insert Unordered List

Inserts an *unordered list* at the cursor position. Three child list items are also automatically inserted by default.

Insert Task List

Inserts a *task list* at the cursor position. Three child list items are also automatically inserted by default.

Insert Table

Inserts a *table* at the cursor position.

Contextual Menu Actions

The following default actions are available in the contextual menu when editing Markdown documents:

Cut, **Copy**, **Paste**

Use these actions to execute the typical editing actions on the currently selected content.

Show/Hide Preview

A toggle action that shows or hides the *Preview* pane.

Export as DITA Topic

Converts the current Markdown document into a DITA topic.

Export as HTML

Converts the current Markdown document into an HTML document.

Print (Available in the *Preview* pane)

Opens a page setup dialog box that allows you to configure printing options for the current document.

Related information

[Markdown Editor](#) on page 405

[Working with Markdown Documents in DITA](#) on page 409

[Markdown Editor Syntax Rules and Specifications](#) on page 409

Syntax Highlighting in the Markdown Editor

Oxygen XML Developer plugin supports syntax highlighting in the Markdown editor to improve the readability of the content and make it easier to internalize the semantics of the content.

To customize the colors or styles used for the syntax highlighting colors for Markdown documents, follow these steps:

1. [Open the Preferences dialog box](#) .
2. Go to **Editor > Syntax Highlight**.
3. Select and expand the **Markdown** section in the top pane.
4. Select the component you want to change and customize the colors or styles using the selectors to the right of the pane.

You can see the effects of your changes in the *Preview* pane.

Related information

[Syntax Highlight Preferences](#) on page 67

[Markdown Editor](#) on page 405

Automatic Validation in Markdown Documents

Markdown documents are validated automatically as you type. The conversion engine constantly tries to parse your changes to display the results in the *Preview* pane. If a change results in an error that prevents the parser from converting the syntax, an error message will be displayed in the **DITA** tab or in the **Results** view at the bottom of the editor.

The types of errors that will be reported include the following:

- Improper order of headers.
- The syntax in a documents begins with something other than a 1st level header.
- Tags are not closed properly if XHTML markup is used in the document.
- Invalid tag names if XHTML markup is used in the document.
- Markup is not well formed if XHTML markup is used in the document.

Related information

[Markdown Editor](#) on page 405

Working with Markdown Documents in DITA

Oxygen XML Developer plugin includes some unique features that allow you to easily integrate Markdown documents in a DITA project. This is especially helpful for teams that have contributors who are familiar with the Markdown syntax, but they want their output to be generated from DITA projects. The integration between the Markdown editor and DITA includes actions to export or convert Markdown documents to DITA topics and the **DITA** tab in the *Preview* pane provides a visualization of how the topic will look after conversion.

Export Markdown as a DITA Topic

The Markdown editor includes an option to quickly convert the current Markdown document into a DITA topic. The **Export as DITA Topic** action is available in the contextual menu of the left-side text editor and the right-side *Preview* pane when the **DITA** tab selected.

The conversion creates a new XML file that is defined as a DITA topic and opens it in the **Text** editing mode. You can then work with the document as you would with any other DITA topic, although you may need to manually correct some issues where the parser could not properly map Markdown syntax to DITA markup.



Tip: Oxygen XML Developer plugin comes with a sample ditamap project for converting Markdown to DITA. Go to the **Project** view, open the `sample.xpr` project, and navigate to the `dita/markdown-dita` folder.

Related information

[Markdown Editor](#) on page 405

[Actions Available in the Markdown Editor](#) on page 407

[Markdown Editor Syntax Rules and Specifications](#) on page 409

[Automatic Validation in Markdown Documents](#) on page 409

Markdown Editor Syntax Rules and Specifications

The Markdown editor in Oxygen XML Developer plugin uses rules that were integrated from the most common set of [default Markdown syntax rules](#) along with many of the [GitHub Flavored Markdown rules](#).

The Oxygen XML Developer plugin implementation of the most commonly used syntax rules are as follows:

Headers

The Markdown editor supports two styles of headers, *Setext* and *Atx*.

- **Setext Style**

Setext-style headers are underlined using equal signs (for first-level headers) and dashes (for second-level headers). Any number of equal signs or dashes will result in the same output.

Example: Setext Style Headers

```
First-Level Header (H1)
=====

Second-Level Header (H2)
-----
```

- **Atx Style**

Atx-style headers use 1-6 hash characters at the start of the line, corresponding to header levels 1-6. Optionally, you may close atx-style headers. This is purely cosmetic and the closing hashes do not need to match the number of hashes used to open the header. It is the number of opening hashes that determines the header level.

Example: Atx Style Headers

```
# H1 text #
## H2 text
### H3 text #####
#### H4 text
##### H5 text ###
##### H6 text
```

Horizontal Rules (for HMTL output only)

You can produce a horizontal rule tag (<hr />) by placing three or more hyphens, asterisks, or underscores on a line by themselves. Optionally, they can be separated by spaces.

- **Example: Horizontal Rules**

```
* * *
*****
-----
_ _ _ _
```

Paragraphs and Line Breaks

A paragraph is simply one or more consecutive lines of text, separated by one or more blank lines. The text at the beginning of a paragraph should not be indented with spaces or tabs. To create a new paragraph, simply insert a blank line in between them.

 **Important:** When converting to HTML, if you break a paragraph on multiple lines (without a blank line in between them), it will create a break tag (
. When converting to DITA, the text is kept in a single paragraph in this case and a blank line is required to break a paragraph. This behavior differs slightly from the default Markdown rules.

- **Example: Paragraphs**

```
This is a paragraph that contains
two lines of text. (In HTML, a break tag is created in between the two lines)

This is a new paragraph.
```

Styling Text

The Markdown editor supports some syntax rules for styling text (such as bold, italic, or strikethrough).

- **Italic (Emphasis)** - Text wrapped with one asterisk or underscore produces an italic (emphasis) tag.

```
*italic*  
_italic_
```

- **Bold (Strong)** - Text wrapped with two asterisks or underscores produces a bold (strong) tag.

```
**bold**  
__bold__
```

- **Strikethrough** - In HTML only, text wrapped with two tildes (~~) produces a strikethrough tag.

```
~~strikethrough~~
```



Tip: You can also combine these styling rules. For example, `**BoldText _ItalicText_ BoldText**` would produce italicized text within bold text. Also, if you surround an asterisk or underscore with spaces, it will be treated as a literal asterisk or underscore. To produce a literal asterisk or underscore at a position where it would otherwise be used as a styling delimiter, you can escape it with a backslash (for example, `\*literal asterisks\*`).

Links

The Markdown editor supports two types of links, *inline* and *reference*. In both cases, it begins with link text that is delimited by [square brackets].

- **Inline Links**

To create an inline link, use a set of regular parentheses immediately after the closing square bracket for the link text. Inside the parentheses, put the URL where you want the link to point, and optionally a title surrounded in quotes. Also, if you referencing a local resource on the same server, you can use relative paths.

Examples: Inline Link

With a title:

```
Text with [example link text](http://www.example.com/path "Title") an inline link  
with a title.
```

Without a title:

```
Text with [example link text](http://www.example.com/path) an inline link without  
a title.
```

Relative path:

```
Text with [example link text](/relative_path/) an inline link with relative path.
```

- **Reference Links**

Reference-type links use a second set of square brackets that include a label (link identifier) to identify the link (it may consist of letters, numbers, spaces, and punctuation and it is not case sensitive). You can optionally use a space to separate the sets of brackets. The labels (link identifiers) are only used for creating the links and do not appear in the output.

```
Text with [link text1][id 1] a reference-type link and [link text2][id_2] another  
one.
```

Then, somewhere in the document, you need to define your link label on a line by itself. The link identifier must be within square brackets followed by a colon, then after one or more spaces the URL for the link. Optionally this can be followed by a title enclosed in single quotes, double quotes, or parentheses. Also, the link may optionally be enclosed in angle brackets (< >).

```
[id 1]: http://example1.com/ "Optional Title"  
[id_2]: <http://example2.com/> "Optional Title2"
```

Other notes about Reference Links:

- You can put the title on a second line and use extra spaces or tabs for padding. This is useful for aesthetics when the URL is long.

```
[id]: http://example.com/long/path/to/resource/here
    "Optional Title Here"
```

- The label (link identifier) can be missing, in which case the link text (in square brackets) is used as the name.

```
[My Link][]
```

and then defined as:

```
[My Link]: http://example.com/
```

Automatic Links

The Markdown editor supports a shortcut style for creating automatic links for URLs and email addresses. You simply surround the URL or email address with angle brackets.



Note: These automatic links only work properly in HTML conversions. The *Preview* pane may display them properly in the DITA tab, but the DITA output will not properly recognize the format.

- **URLs**

By surrounding a URL with angle brackets, you can show the actual text of the URL while also making it clickable in the output.

```
<http://example.com/>
```

For example, in HTML it is converted to:

```
<a href="http://example.com/">http://example.com/</a>
```

- **Email Addresses**

Automatic links for email addresses work similarly, except that Markdown will also perform a bit of randomized decimal and hex entity-encoding to help obscure your address from address-harvesting *spambots*.

```
<address@example.com>
```

In HTML, it is converted to something like:

```
<a href="&#x6D;&#x61;i&#x6C;&#x74;&#x6F;:&#x61;&#x64;&#x64;&#x72;&#x65;
&#115;&#115;&#64;&#101;&#120;&#x61;&#109;&#x70;&#x6C;e&#x2E;&#99;&#111;
&#109;">&#x61;&#x64;&#x64;&#x72;&#x65;&#115;&#64;&#101;&#120;&#x61;
&#109;&#x70;&#x6C;e&#x2E;&#99;&#111;&#109;</a>
```

Images

The Markdown editor uses an image syntax that is intended to resemble the syntax for links, allowing for the same two types: *inline* and *reference*. In both cases, the syntax for images begin with an exclamation mark, followed by `alt` attribute text surrounded by square brackets., and then followed by a set of parentheses that contain the URL or path to the image.

- **Inline Images**

For inline images, use a set of regular parentheses immediately after the closing square bracket for the `alt` attribute text. Inside the parentheses, put the URL or path of the image, and optionally a title surrounded in quotes.

Examples: Inline Images

With a title:

Text with `![Alt text](/path/to/img.jpg "Optional title")` an inline image with a title.

Without a title:

Text with ![Alt text](/path/to/img.jpg) an inline link without a title.

- **Reference Images**

For reference-type images, use a second set of square brackets that include a label (image identifier) to identify the image (it may consist of letters, numbers, spaces, and punctuation and it is not case sensitive). You can optionally use a space to separate the sets of brackets. The labels (image identifiers) do not appear in the output.

Text with ![Alt text1][id] a reference-type image.

Then, somewhere in the document, you need to define your image label on a line by itself. The image identifier must be within square brackets followed by a colon, then after one or more spaces the URL or path of the image. Optionally this can be followed by a title enclosed in single quotes, double quotes, or parentheses.

[id]: url/to/image "Optional Title"

Blockquotes

The Markdown editor uses email-style greater than characters (>) for *blockquotes*. You only need to put the > before the first line of a hard-wrapped paragraph, but it looks better (and is more clear) if you put a > before every line.

- **Example: Blockquotes**

```
> This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet,  
> consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus.  
> Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.  
>  
> Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse  
> id sem consectetur libero luctus adipiscing.
```

- *Blockquotes* can be nested by adding additional levels of > characters.

Example: Nested Blockquotes

```
> This is the first level of quoting.  
>  
> > This is nested blockquote.  
>  
> Back to the first level.
```

- *Blockquotes* can also contain other Markdown elements (such as headers, lists, and code blocks).

Example: Blockquotes with Other Markdown Elements

```
> ## This is a header.  
>  
> 1. This is the first list item.  
> 2. This is the second list item.  
>  
> Here's some example code:  
>  
>     return shell_exec("echo $input | $markdown_script")
```

Quoting Code (Inline and Code Blocks)

The Markdown editor supports quoting code or commands inline within a sentence or in distinct blocks.

- **Inline**

You can quote or emphasize code within a sentence (inline) with single backticks (`). The text within the backticks will not be formatted.

Example: Inline Code Emphasis

This is a normal sentence with a `code` in the middle.

- **Code Blocks**

You can format code or text into its own distinct block by inserting a blank line before and after the content and using at least 4 spaces (or 1 tab), or by using opening and closing triple backticks (```) on separate lines.

Example: Code Block

This is a normal paragraph:

```
    This is a code block
```

This is a normal paragraph:

```
```\n
```

```
 This is a code block\n```\n
```

One level of indentation is removed from each line of a codeblock and it continues until it reaches a line that is not indented (or until the closing backticks).

**Example: Code Block with Indentation**

```
 tell application "something"
 beep
 end tell
```

For example, in HTML the result would look like this:

```
<pre><code>tell application "Foo"
 beep
end tell
</code></pre>
```

You can also add an optional language identifier to enable syntax highlighting in your code blocks. The Oxygen XML Developer plugin Markdown editor supports the following languages: *Java*, *JavaScript*, *CSS*, and *Python*.

**Example: Syntax Highlighting in Code Block**

```
```css
input[type="submit"] {
  color: white;
  font-weight: bold;
}
```

Inline XHTML (for HMTL output only)

The Markdown editor supports writing inline XHTML. Since Markdown is just a writing format, it requires a conversion for publishing purposes. If you are using the HTML conversion, for any markup that is not covered by Markdown syntax, you can simply use XHTML syntax.

- **Example: Inline XHTML**

This is a regular paragraph.

```
<table>
  <tr>
    <td>Col 1</td>
    <td>Col 2</td>
  </tr>
</table>
```

This is another regular paragraph.

Lists

The Markdown editor supports ordered and unordered lists. You can also insert *blockquotes* and *code blocks* inside list items. List markers typically start at the left margin, but may be indented by up to three spaces.

- **Unordered Lists**

For unordered lists, you can use asterisks (*), plus signs (+), and hyphens (-) interchangeably.

```
* List item 1
+ List item 2
- List item 3
```

- **Ordered Lists**

For ordered lists, use numbers followed by periods. The actual numbers you use have no effect on the output. It simply converts them to list items within an ordered list and the actual number of list items will determine the numbers in the output.

```
1. List item 1
8. List item 2
5. List item 3
```

- **Nested Lists**

You can create nested lists by indenting lines by two spaces.

```
1. Ordered list item 1
  1. Nested ordered list item 1
  2. Nested ordered list item 2
    * 2nd level nested unordered list item 1
    * 2nd level nested unordered list item 2
      * 3rd level nested unordered list item 1
2. Ordered list item 2
```

- **Paragraphs Inside Lists**

If list items are separated by blank lines, Markdown will wrap the items in a paragraph in the output.

```
* List item 1
* List item 2
```

For both DITA and HTML output, this would result in:

```
<ul>
<li><p>List item 1</p></li>
<li><p>List item 2</p></li>
</ul>
```

- **Multiple Paragraphs Inside Lists**

List items may consist of multiple paragraphs. Each subsequent paragraph in a list item must be indented by either 4 spaces or one tab. Optionally, you can also indent each line of a paragraph to make it look nicer.

```
1. This is a list item with two paragraphs. Lorem ipsum dolor
   sit amet, consectetur adipiscing elit. Aliquam hendrerit
   mi posuere lectus.

   Vestibulum enim wisi, viverra nec, fringilla in, laoreet
   vitae, risus. Donec sit amet nisl. Aliquam semper ipsum
   sit amet velit.

2. Suspendisse id sem consectetur libero luctus adipiscing.
```

- **Blockquotes Inside Lists**

To put a *blockquote* within a list item, the blockquote delimiters (>) need to be indented so that it is under the first letter of the text after the list item marker.

```
* A list item with a blockquote:  
  > This is a blockquote  
  > inside a list item.
```

- **Code Blocks Inside Lists**

To put a code block within a list item, insert an empty line in between the list item and the code block, and the code block needs to be indented twice (with 8 spaces or 2 tabs), or if you are using the triple backticks method, the opening triple backtick needs to be indented with 4 spaces or 1 tab.

```
* A list item with a code block:  
  
    This is a code block inside a list item  
  
    ```  
 This is a code block inside a list item using the backticks method
    ```
```

Task Lists

You can create task lists by prefacing list items with square brackets ([]). To mark a task as complete, use [x].

- **Example: Task Lists**

```
- [ ] Unfinished task 1  
- [x] Finished task 2
```

Definition Lists

You can create definition lists by using a colon plus a space for each list item.

- **Example: Definition Lists**

```
Term 1  
: Definition A  
: Definition B
```

Tables

You can create tables in the Markdown editor by using pipes (|) and hyphens (-).

- **Creating a Table**

Pipes are used to separate each column, while hyphens are used to create column headers. The pipes on either end of the table are optional. Cells can vary in width and do not need to be perfectly aligned within columns, but there must be at least three hyphens in each column of the header row.

```
| First Header | Second Header |  
| ----- | ----- |  
| Column 1 Row 1 Cell | Column 2 Row 1 Cell |  
| Column 1 Row 2 Cell | Column 2 Row 2 Cell |
```

- **Formatting Rules in Table Cells**

You can use formatting rules inside the cells of the table (such as links, inline code blocks, and text styling).

```
| First Header | Second Header |  
| --- | --- |  
| `inline code` | Content with bold text inside cell |
```


- **Aligning Text in Tables**

You can align text to the left, right, or center of a column by including colons (:) to the left, right, or on both sides of the hyphens within the header row.

Left-aligned	Center-aligned	Right-aligned	
:---	:---:	---:	
Content Cell	Content Cell	Content Cell	

Emoji

You can add *emoji* in the Markdown editor by surrounding the EMOJICODE with colons (:EMOJICODE:).

- **Example: Emoji**

```
:smile:
:laughing:
```

The resulting emoticons will appear in the output, but they are not displayed in the *Preview* pane.

For a full list of available emoji codes, see [Emoji Cheat Sheet](#).

Backslash Escapes

You can ignore Markdown formatting by using backslash escapes (\) to generate literal characters that would otherwise have special meaning in the Markdown syntax. For example, if you want to surround a word with literal asterisks (instead of an italic or emphasis tag), you can use backslashes to escape the asterisks.

```
\*literal asterisks\*
```

The Markdown editor provides backslash escapes for the following characters:

```
\  backslash
`  backtick
*  asterisk
_  underscore
{} curly braces
[] square brackets
() parentheses
#  hash mark
+  plus sign
-  minus sign (hyphen)
.  dot
!  exclamation mark
```

Automatic Escaping for Special Characters

The Markdown editor includes support for automatically escaping special characters such as angle brackets (< >) and ampersands (&). If you want to use them as literal characters, you must escape them as entities, as in the table below. The exception to this is in HTML output, if the special characters for a valid tag (for example,), they are treated as literal characters and escaping is not necessary.

Literal Character	Escaping Code
<	<
>	>
&	&

Footnotes

The Markdown editor in Oxygen XML Developer plugin supports normal and inline footnotes. The following examples show the required syntax.

- **Example: Normal Footnote**

```
Here is a footnote reference, [^1]

[^1]: Here is the footnote.
```

- **Example: Normal Footnote with Multiple Blocks**

```
Here is a footnote reference, [^longnote]

[^longnote]: Here is the footnote with multiple blocks.

    Subsequent paragraphs are indented with 4 spaces or 1 tab to show that they
    belong to the previous footnote.
```

- **Example: Inline Footnote**

```
Here is an inline note.^[Inlines notes are easier to write, since
you don't have to pick an identifier and move down to type the
note.]
```

Related information

[Default Markdown Syntax](#)

[GitHub Flavored Markdown Rules](#)

[Markdown Editor](#) on page 405

[Actions Available in the Markdown Editor](#) on page 407

Spell Checking

Oxygen XML Developer plugin includes an [automatic \(as-you-type\) spell checking feature](#), as well as a manual spell checking action to open a **Spelling** dialog box that offers a variety of options.

To manually check spelling in the current document, use the  **Check Spelling** action on the toolbar.

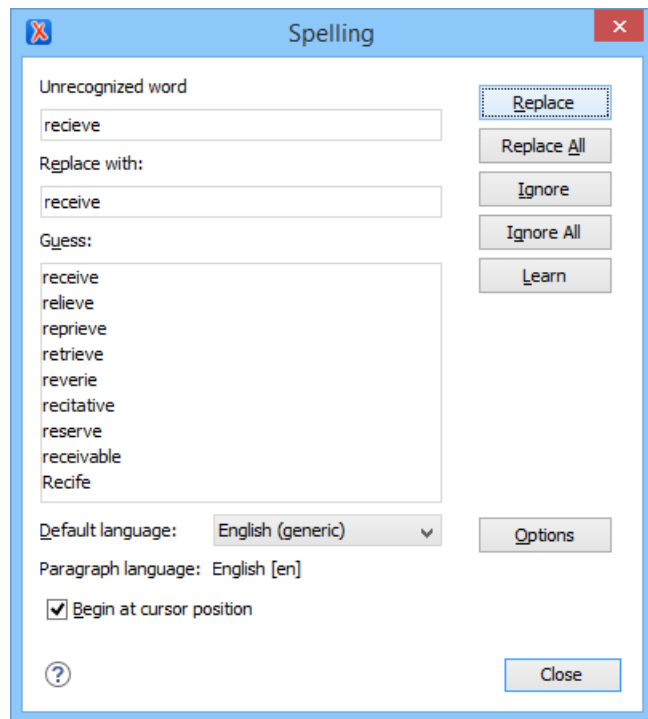


Figure 270: Check Spelling Dialog Box

The **Spelling** dialog box contains the following:

Unrecognized word

Displays the word that cannot be found in the selected dictionary. The word is also highlighted in the XML document.

Replace with

The character string that will replace the misspelled 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 language dictionary used by the spelling engine.

Paragraph language

In an XML document, you can mix content written in multiple languages. You can set the language code in the `lang` or `xml:lang` attribute for any particular section and Oxygen XML Developer plugin will automatically instruct the spell checker engine to apply the appropriate language dictionary for that section.

Begin at cursor position

Instructs the spell checker to begin checking the document starting from the current cursor position.

Action Buttons

Replace

Use this button to replace the unrecognized word with the selected word from the **Replace with** field.

Replace All

Use this button to replace all occurrences of the unrecognized word with the selected word from the **Replace with** field.

Ignore

Ignores the first occurrence of the unrecognized word and allows you to continue checking the document. Oxygen XML Developer plugin skips the content of the XML elements *marked to be ignored*.

Ignore All

Ignores all instances of the unrecognized word in the current document.

Learn

Adds the unrecognized word to the list of valid words.

Options

Opens the [Spell Check preferences page](#) where you can configure various options in regards to the feature.

Spell Check Dictionaries and Term Lists

Oxygen XML Developer plugin uses the *Hunspell* engine for the spell checking feature. The Hunspell spell checking engine is open source and has an LGPL license. It is designed for languages with rich morphology and complex compounding or character encoding. Each language-country variant combination have their own specific dictionaries. Oxygen XML Developer plugin includes the following built-in dictionaries for the spell checker:

- English (US) [en_US]
- English (UK) [en_GB]
- French [fr]
- German [de_DE]
- Spanish [es_ES]

Other Hunspell Dictionaries

You can also download Hunspell dictionaries for other languages and add them to the Oxygen XML Developer plugin spell checker. An example of a website that includes numerous dictionary files is: <http://extensions.services.openoffice.org/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 full Hunspell dictionary, follow [these instructions](#) and then add the dictionary to the Oxygen XML Developer plugin spell checker by following [this procedure](#).

Personalized Term Lists

Authoring in certain areas of expertise (for example, the pharmaceutical or automobile industries) might require the use of specific terms that are not part of the standard spell checker dictionary. To avoid marking these terms as errors, Oxygen XML Developer plugin provides a way of [adding personalized term lists](#) to the spell check engine. This involves creating a term list file that the spell checker will recognize and it is similar to the file Oxygen XML Developer plugin uses for storing [learned words](#).

The term list files are specific for each language and can be specific to each domain or area of expertise (for example, *legal*, *medical*, *automotive*). They can also be used to control forbidden words.

Related information

[Adding Spell Check Dictionaries](#) on page 420

[Adding Spell Check Term Lists](#) on page 422

[Building and Testing Hunspell Dictionaries](#)

Adding Custom Dictionaries and Term Lists

The Oxygen XML Developer plugin spell checker allows you to add customized Hunspell dictionaries and personalized term lists. The Hunspell dictionary mechanism requires a dictionary file (with a `.dic` file extension) and an affix file (with an `.aff` file extension). The personalized term lists are custom files (with a `.tdi` file extension) that you can create to include specialized terms or specify forbidden words in the Oxygen XML Developer plugin spell checker.

You can [add dictionaries](#) and [personalized term lists](#) to the default folder where they are stored or specify your own custom locations. You can view the default storage location in the [Spell Check Dictionaries preferences page](#) and the [Include dictionaries and term list from option](#) allows you to choose a custom storage location. All the dictionaries and term lists for a particular language that are found in either location are merged and used by the spell checker in Oxygen XML Developer plugin.

Related information

[Replacing a Spell Check Dictionary](#) on page 422

Adding Spell Check Dictionaries

There are three possible scenarios for adding Hunspell dictionaries to the Oxygen XML Developer plugin spell checker:

- You can download a pre-built Hunspell dictionary and add it to the spell checking mechanism.
- You can create a custom Hunspell dictionary file that defines your own list of words and add it to the spell checking mechanism.
- You can build your own full Hunspell dictionary and add it to the spell checking mechanism.

Download and Add a Pre-Built Hunspell Dictionary

To add a downloaded pre-built dictionary, follow these steps:

1. Download the files needed for your dictionary. You will need a *dictionary* file (with a `.dic` file extension) and an *affix* file (with an `.aff` file extension). If the dictionary does not include an affix file (`.aff`), you can create one and leave it empty, but it is needed for the mechanism to work properly. An example of a website that includes numerous dictionary files is: <http://extensions.services.openoffice.org/dictionary>.



Important: The name of the files should begin with a two letter prefix that indicates the language it should be attached to, followed by an underscore or hyphen, and then a descriptive name (for example, `en_US_medical.dic` for a medical dictionary in the US version of the English language or `en_medical.dic` for a less specific English medical dictionary). For a list of language codes, see https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.

2. *Open the Preferences dialog box* and go to **Editor > Spell Check > Dictionaries**.
3. Choose one of the following two options for adding the downloaded files.
 - a. Copy both files (`.dic` and `.aff`) to the default directory displayed in the *Dictionaries and term lists default folder option*.
 - b. Copy both files (`.dic` and `.aff`) to any other directory, enable the *Include dictionaries and term list from option*, and select that directory. If you choose this option, make sure you read *this important note*.
4. Restart the application for the spell checker to start using the new dictionary.

Create a Custom Hunspell Dictionary that Defines a List of Words

To create a custom Hunspell dictionary that defines your own list of words, follow these steps:

1. Create a *dictionary* file (with a `.dic` file extension) and an *affix* file (with an `.aff` file extension). The affix file (`.aff`) can be left empty, but it is needed for the mechanism to work properly.



Important: The name of the files should begin with a two letter prefix that indicates the language it should be attached to, followed by an underscore or hyphen, and then a descriptive name (for example, `en_US_medical.dic` for a medical dictionary in the US version of the English language or `en_medical.dic` for a less specific English medical dictionary). For a list of language codes, see https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.

2. In the dictionary file (`.dic` extension), add the words you want to be included in your custom dictionary. Add one word per row and the first line needs to contain the number of words, as in the following example:

```
2
parabola
asimptotic
```

3. *Open the Preferences dialog box* and go to **Editor > Spell Check > Dictionaries**.
4. Choose one of the following two options for saving the files.
 - a. Save both files (`.dic` and `.aff`) to the default directory displayed in the *Dictionaries and term lists default folder option*.
 - b. Save both files (`.dic` and `.aff`) to any other directory, enable the *Include dictionaries and term list from option*, and select that directory. If you choose this option, make sure you read *this important note*.
5. Restart the application for the spell checker to start using the new dictionary.

Build and Add a Full Hunspell Dictionary

To build and add a full Hunspell dictionary, follow these steps:

1. Follow these instructions: *Building and Testing Hunspell Dictionaries*.

Result: You should end up with a *dictionary* file (with a `.dic` file extension) and an *affix* file (with an `.aff` file extension). The affix file (`.aff`) can be empty, but it is needed for the mechanism to work properly.



Important: The name of the files should begin with a two letter prefix that indicates the language it should be attached to, followed by an underscore or hyphen, and then a descriptive name (for example, `en_US_medical.dic` for a medical dictionary in the US version of the English language or `en_medical.dic` for a less specific English medical dictionary). For a list of language codes, see https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.

2. Open the **Preferences dialog box** and go to **Editor > Spell Check > Dictionaries**.
3. Choose one of the following two options for saving the files.
 - a. Save both files (`.dic` and `.aff`) to the default directory displayed in the **Dictionaries and term lists default folder option**.
 - b. Save both files (`.dic` and `.aff`) to any other directory, enable the **Include dictionaries and term list from option**, and select that directory. If you choose this option, make sure you read [this important note](#).
4. Restart the application for the spell checker to start using the new dictionary.

Related information

[Adding Spell Check Term Lists](#) on page 422

Adding Spell Check Term Lists

You can create personalized term lists that are used to store specialized terms or control forbidden words. They can then be added to one of the directories that store the spell check dictionaries and the spell checker will merge them with all the dictionaries and other term lists for a particular language.

Create and Add Personalized Term Lists

To create and add a personalized term list, follow these steps:

1. Create a *term list* file (with a `.tdi` file extension). The name of the file must begin with a two letter prefix that indicates the language it should be attached to, followed by an underscore or hyphen, and then a descriptive name (for example, `en_US_myterms.tdi` for term list in the US version of the English language or `en_myterms.tdi` for a less specific English term list). For a list of language codes, see https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.
2. In the term list file (`.tdi` extension), add the terms you want to be included in your custom dictionary. If you need to specify forbidden terms, those words simply need to be preceded by an asterisk. Add one word per row, as in the following example:

```
parabola
asimptotic
*hyperbola
```

3. Open the **Preferences dialog box** and go to **Editor > Spell Check > Dictionaries**.
4. Choose one of the following two options for saving the file.
 - a. Save the file (`.tdi`) to the default directory displayed in the **Dictionaries and term lists default folder option**.
 - b. Save the file (`.tdi`) to any other directory, enable the **Include dictionaries and term list from option**, and select that directory. If you choose this option, make sure you read [this important note](#).
5. Restart the application for the spell checker to start using the new term list.

Related information

[Adding Spell Check Dictionaries](#) on page 420

Replacing a Spell Check Dictionary

There are several possible scenarios for replacing an existing Hunspell dictionary for the Oxygen XML Developer plugin spell checker:

- You can download a pre-built Hunspell dictionary and replace an existing dictionary with it.
- You can build your own full Hunspell dictionary and replace an existing dictionary with it.

Download a Pre-Built Hunspell Dictionary and Replace an Existing One

To replace an existing dictionary with a downloaded pre-built dictionary, follow these steps:

1. Download the files needed for your dictionary. You will need a *dictionary* file (with a `.dic` file extension) and an *affix* file (with an `.aff` file extension). If the dictionary does not include an affix file (`.aff`), you can create one and leave it empty, but it is needed for the mechanism to work properly. An example of a website that includes numerous dictionary files is: <http://extensions.services.openoffice.org/dictionary>.
 2. *Open the Preferences dialog box* and go to **Editor > Spell Check > Dictionaries**.
 3. Choose one of the following two options to replace existing files.
 - a. Replace the existing files (`.dic` and `.aff`) for the particular language in the default directory displayed in the **Dictionaries and term lists default folder option**. Leave the **Include dictionaries and term list from** option disabled.
 - b. Replace existing files (`.dic` and `.aff`) for the particular language in a directory specified in the **Include dictionaries and term list from option**. If you choose this option, make sure you read [this important note](#).
-  **Important:** Do not alter the naming convention. The name of the files must begin with a two letter prefix that indicates the language it should be attached to (for example, `en_US.dic` for a US English dictionary or `en.dic` for a less specific English dictionary). For a list of language codes, see https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.
4. Restart the application for the spell checker to start using the new dictionary.

Build a Full Hunspell Dictionary and Replace an Existing One

To replace an existing dictionary with a full Hunspell dictionary that you build, follow these steps:

1. Follow these instructions: [Building and Testing Hunspell Dictionaries](#).

Result: You should end up with a *dictionary* file (with a `.dic` file extension) and an *affix* file (with an `.aff` file extension). The affix file (`.aff`) can be empty, but it is needed for the mechanism to work properly.
2. *Open the Preferences dialog box* and go to **Editor > Spell Check > Dictionaries**.
3. Choose one of the following two options to replace existing files.
 - a. Replace the existing files (`.dic` and `.aff`) for the particular language in the default directory displayed in the **Dictionaries and term lists default folder option**. Leave the **Include dictionaries and term list from** option disabled.
 - b. Replace existing files (`.dic` and `.aff`) for the particular language in a directory specified in the **Include dictionaries and term list from option**. If you choose this option, make sure you read [this important note](#).
4. Restart the application for the spell checker to start using the new dictionary.

Related information

[Adding Custom Dictionaries and Term Lists](#) on page 420

Learned Words

Spell checker engines rely on dictionaries to decide if a word is spelled correctly. To instruct the spell checker engine that an unknown word is actually correctly spelled, you need to add that word to a list of learned words. There are two ways to do this:

- Invoke the contextual menu on an unknown word, then select **Learn word**.
- Press the **Learn** button from the *Spelling dialog box* that is invoked by using the  **Check Spelling** action on the toolbar.



Note: To delete items from the list of learned words, use the **Delete learned words** option in the **Editor > Spell Check > Dictionaries preferences page**.

Ignored Words (Elements)

There are certain XML elements (such as `programlisting`, `codeblock`, or `screen`) in which you may want the content to always be skipped during the spell check process. This can be done in one of several ways:

- You can skip through them manually, word by word, using the **Ignore** button in the *Spelling dialog box* that is invoked by using the  **Check Spelling** action on the toolbar.

- You can automatically skip the content of certain elements by maintaining a set of known element names that should never be checked. You can manage this set of element names by using the [Ignore elements section](#) in the **Spell Check** preferences page.

Automatic Spell Check

Oxygen XML Developer plugin includes an option to automatically check the spelling as you type. This feature is disabled by default, but it can be enabled and configured in the [Spell Check preferences page](#). When the **Automatic Spell Check option** is enabled, unknown words are underlined and some actions are available in the contextual menu to help you correct the word or prevent the word from being reported in the future.

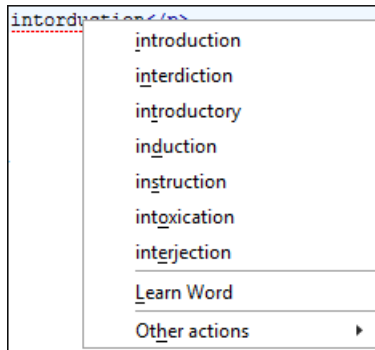


Figure 271: Automatic Spell Checking in Text Mode

The contextual menu includes the following actions:

Delete Repeated Word

Allows you to delete words that were repeated in consecutive order.

List of Suggestions

A list of words suggested by the spell checking engine as possible replacements for the unknown word.

Learn Word

Allows you to add the current unknown word to the persistent dictionary of [learned words](#).

Other actions

This submenu give you access to all the usual contextual menu actions.

Related information

[Learned Words](#) on page 423

Spell Check 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.

This action opens the **Check Spelling in Files** dialog box that allows you to define the scope and several other options. After you configure the settings for the operation, click the **Check All** button to check the spelling in all specified files. The spelling corrections are displayed in [the Results view](#) at the bottom of the editor and you can group the reported errors as a tree with two levels.

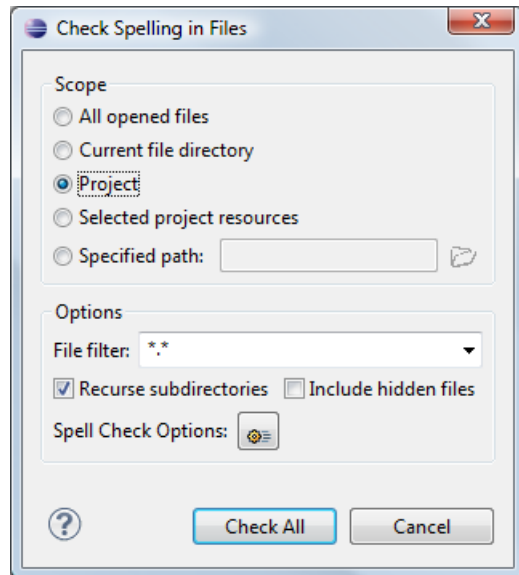


Figure 272: Check Spelling in Files Dialog Box

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](#).

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 can modify it on the spot without having to reopen it.

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 their own private key. Only the originator has that private key and that person is the only one who can encrypt the hash so that it can be unencrypted using their 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 for applications that require the ability to test whether or not the information content of a document or document subset has been changed. This is done by comparing the canonical form of the original document before application processing with the canonical form of the document result of the application processing.

A digital signature over the canonical form of an XML document or document subset would allow the signature digest calculations to be oblivious to changes in the original document's physical representation. During signature generation, the digest is computed over the canonical form of the document. The document is then transferred to the relying party, which validates the signature by reading the document and computing a digest of the canonical form of the received document. The equivalence of the digests computed by the signing and relying parties (hence, the equivalence of the canonical forms for which they were computed) ensures that the information content of the document has not been altered since it was signed.

The following canonicalization algorithms are used in Oxygen XML Developer plugin: Canonical XML (or Inclusive XML Canonicalization)([XMLC14N](#)) and Exclusive XML Canonicalization([EXCC14N](#)). The first is used for XML where the context doesn't change while the second was designed for canonicalization where the context might change.

Inclusive Canonicalization copies all the declarations, even if they are defined outside of the scope of the signature, 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 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 various 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.

Related tasks

[Example of How to Digitally Sign XML Files or Content](#) on page 430

Related information

[Certificates](#) on page 427

[Canonicalizing Files](#) on page 427

[Signing Files](#) on page 428

[Verifying Signature](#) on page 430

Certificates

A certificate is a digitally signed statement from the issuer (an individual, an organization, a website or a firm), saying that the public key (and some other information) of some other entity has a particular value. When data is digitally signed, the signature can be verified to check the data integrity and authenticity. Integrity means that the data has not been modified. Authenticity means the data comes indeed from the entity that claims to have created and signed it. Certificates are kept in special repositories called *keystores*.

A *keystore* is an encrypted file that contains private keys and certificates. All *keystore* entries (key and trusted certificate entries) are accessed via unique aliases. An alias must be assigned for every new entry of either a key or certificate as a reference for that entity. No *keystore* can store an entity if its alias already exists in that *keystore* and cannot store trusted certificates generated with keys in its *keystore*.

In Oxygen XML Developer plugin there are provided two types of *keystores*: Java Key Store (JKS) and Public-Key Cryptography Standards version 12 (PKCS-12). A *keystore* file is protected by a password. In a PKCS 12 *keystore* you should not store a certificate without alias together with other certificates, with or without alias, as in such a case the certificate without alias cannot be extracted from the *keystore*.

To configure the options for a certificate or to validate it, [open the Preferences dialog box](#) and go to **XML > XML Signing Certificates**. This opens [the certificates preferences page](#).

Related information

[Digital Signatures Overview](#) on page 425

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.

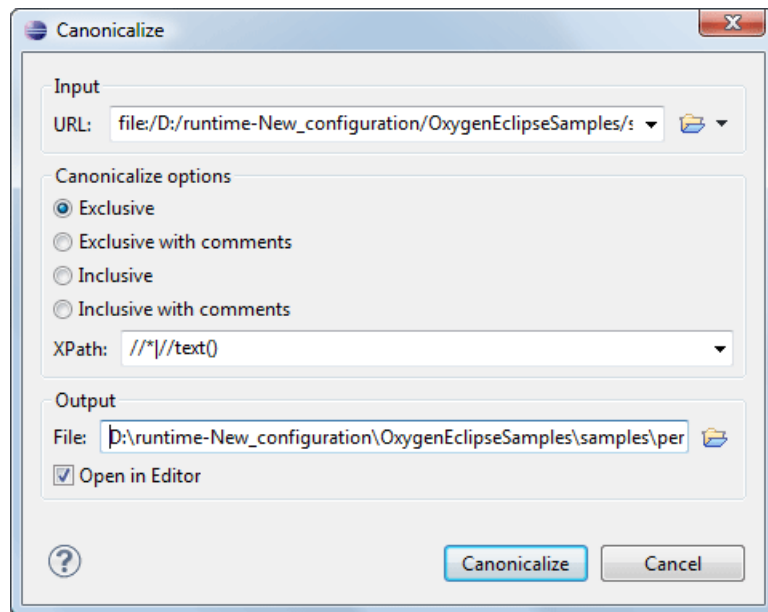


Figure 273: Canonicalization Settings Dialog Box

The **Canonicalize** dialog box allows you to set the following options:

- **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 various 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 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.

Related information

[Digital Signatures Overview](#) on page 425

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.

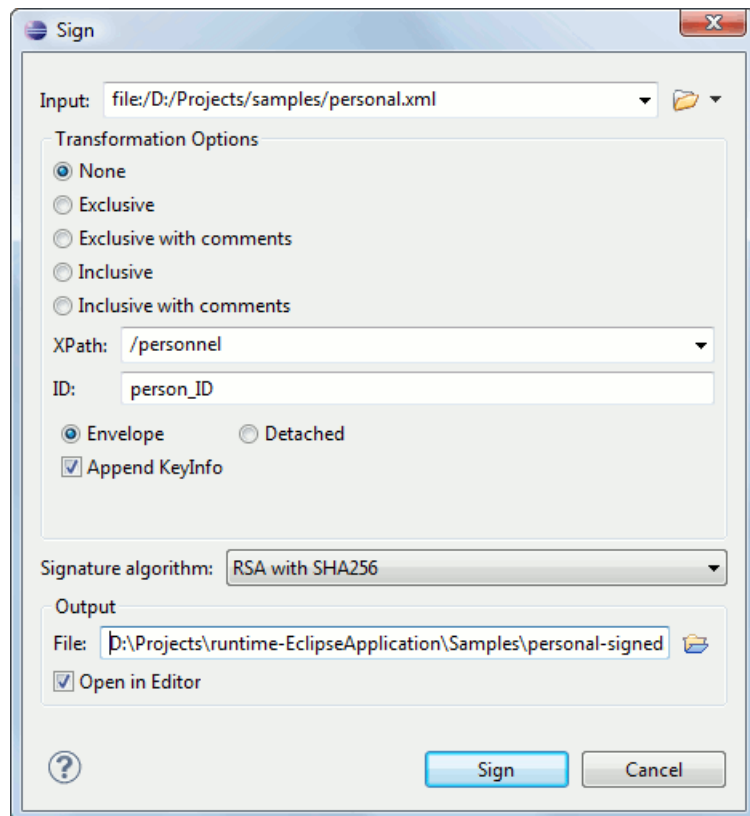


Figure 274: Signature Settings Dialog Box

The following options are available:



Note: If Oxygen XML Developer 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.

❗ Could not obtain a valid certificate. [You must 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 various 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 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 Developer plugin.

Related tasks

[Example of How to Digitally Sign XML Files or Content](#) on page 430

Related information

[Digital Signatures Overview](#) on page 425

[Verifying Signature](#) on page 430

Verifying 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.

Related tasks

[Example of How to Digitally Sign XML Files or Content](#) on page 430

Related information

[Digital Signatures Overview](#) on page 425

[Signing Files](#) on page 428

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 Developer plugin includes a signature tool that allows you to digitally sign XML documents or specific content.

The Oxygen XML Developer 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 Developer 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 Developer 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.

Related information

Digital Signatures Overview on page 425

Signing Files on page 428

Verifying Signature on page 430

Chapter

8

Document Types (Frameworks)

Topics:

- [Predefined Document Types \(Frameworks\)](#)

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 **Text** mode.
- Predefined scenarios used for transformations for the class of XML documents defined by the document type.
- XML catalogs.
- Directories with file templates.

Oxygen XML Developer plugin includes built-in support for many common document types. Each document type is defined in a framework.

To see a video on configuring a framework in Oxygen XML Developer plugin, go to <https://www.oxygenxml.com/demo/FrameworkConfiguration.html>.

Predefined Document Types (Frameworks)

Oxygen XML Developer plugin includes a variety of specialized editors, views, and features that are dynamic according to the type of document that you open or create. The type of documents that are supported include the most popular predefined XML frameworks that include a full set of features as well as other document types that include more generic or common features.

Predefined Document Types

The following predefined document types (frameworks) are fully supported in Oxygen XML Developer plugin and each of these document types include built-in transformation scenarios, validation, content completion and file templates:

- [DocBook 4](#) - A document type standard for books, articles, and other prose documents (particularly technical documentation).
- [DocBook 5](#) - An enhanced (version 5) document type standard designed for a variety of documents (particularly technical documentation).
- [DITA](#) - An XML-based architecture designed for authoring, producing, and delivering technical information.
- [DITA Map](#) - A document type that collects and organizes references to DITA topics or other maps.
- [XHTML](#) - Extensible HyperText Markup Language includes the same depth of expression as HTML, but also conforms to XML syntax.
- [TEI ODD](#) - Text Encoding Initiative One Document Does it all is an XML-conformant specification that allows you to create TEI P5 schema in a literate programming style.
- [TEI P4](#) - The Text Encoding Initiative guidelines is a standard for the academic community that collectively define an XML format for text that is primarily semantic rather than presentational.
- [TEI P5](#) - The Text Encoding Initiative guidelines is a standard for the academic community that collectively define an XML format for text that is primarily semantic rather than presentational.
- [JATS](#) - The NISO Journal Article Tag Suite is a technical standard that defines an XML format for scientific literature.

Other Supported Document Types

Oxygen XML Developer plugin also provides limited support (including file templates) for editing a variety of other document types. All the specialized views, editors, actions, and options are dynamic according to the type of file that is opened or created. Other document types that are supported in Oxygen XML Developer plugin include:

- [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.
- [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.
- [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.

DocBook 4 Document Type

DocBook is a very popular set of tags for describing books, articles, and other prose documents, particularly technical documentation. DocBook provides a vast number of semantic element tags, divided into three broad categories: structural, block-level, and inline. DocBook content can then be published in a variety of formats, including HTML, PDF, WebHelp, and EPUB.

File Definition

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`.

Default Document Templates

There are a variety of default *DocBook 4* templates available when creating [new documents from templates](#), including **Article**, **Book**, **Chapter**, **Glossary**, **Section**, and more.

The default templates for DocBook 4 documents are located in the `[OXYGEN_INSTALL_DIR]/frameworks/docbook/templates/Docbook 4` folder.

Default Schema

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_INSTALL_DIR]/frameworks/docbook/4.5/dtd/`.

Default XML Catalog

The default XML catalog, `catalog.xml`, is stored in `[OXYGEN_INSTALL_DIR]/frameworks/docbook/`.

Resources

- [DocBook Specifications](#)

Related information

[Editing XML Documents in Text Mode](#) on page 161

DocBook 4 Transformation Scenarios

Default transformation scenarios allow you to convert DocBook 4 to DocBook 5 documents and transform DocBook documents to a variety of outputs, such as WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, and EPUB.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

DocBook 4 to WebHelp Output

DocBook 4 documents can be transformed into several types of WebHelp systems.

WebHelp Classic Output

To publish a DocBook 4 document as a *WebHelp Classic* system, follow these steps:

1. Click the  **Configure Transformation Scenario(s)** action from the toolbar.
2. Select the **DocBook WebHelp Classic** scenario from the **DocBook 4** section.
3. Click **Apply associated**.

When the **DocBook WebHelp Classic** transformation is complete, the output is automatically opened in your default browser.

WebHelp Classic with Feedback Output

To publish a DocBook 4 document as a *WebHelp Classic with Feedback* system, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp Classic 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 Classic 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. Follow the instructions to complete the system deployment. For more information, see [Deploying a Feedback-Enabled WebHelp System](#) on page 563.

To watch our video demonstration about the feedback-enabled WebHelp system, go to https://www.oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

WebHelp Classic Mobile Output

To publish a DocBook 4 document as a *WebHelp Classic Mobile* system, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp Classic Mobile** scenario from the **DocBook 4** section.
3. Click **Apply associated**.

When the **DocBook WebHelp Classic Mobile** transformation is complete, the output is automatically opened in your default browser.

Customizing WebHelp Transformation Scenarios

To customize a DocBook WebHelp transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

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.

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.logo.image.target.url

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

webhelp.logo.image

Specifies a path to an image displayed as a logo in the left side of the output header.

webhelp.product.id (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

webhelp.product.version (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.skin.css

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.

For more information about all the DocBook transformation parameters, go to <http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html>.

Related information

[WebHelp Classic Output](#) on page 456

[WebHelp Classic With Feedback Output](#) on page 458

[WebHelp Classic Mobile System](#) on page 634

[Customizing WebHelp Classic Systems](#) on page 613

[Customizing WebHelp Classic Mobile Systems](#) on page 634

DocBook to PDF Output Customization

When the default layout and output of the DocBook to PDF transformation needs to be customized, follow these steps:

1. Create a custom version of the DocBook title spec file.

You should start from a copy of the file

`[OXYGEN_INSTALL_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_INSTALL_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 this:

```
<xsl:import href="dir-name/mytitlepages.xsl"/>
```

4. Insert a 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 to duplicate the transformation scenario **DocBook PDF** that is included with Oxygen XML Developer plugin and set the customization layer in [the XSL URL property of the scenario](#).

Related information

[The book *DocBook XSL: The Complete Guide* by Bob Stayton contains more details about customizing the PDF output.](#)

[Video demonstration for creating a DocBook customization layer in Oxygen XML Developer plugin.](#)

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. Open the **Configure Transformation Scenario(s)** dialog box, select the **DocBook EPUB** transformation scenario, and click **Edit**.
4. In the **Parameters** tab, 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.

5. Run the transformation scenario.

DocBook to DITA Transformation

Oxygen XML Developer plugin includes a built-in transformation scenario that is designed to convert DocBook content to DITA. This transformation scenario is based upon a DITA Open Toolkit plugin that is available at sourceforge.net.

To convert a DocBook document to DITA, follow these steps:

1. Use one of the following two methods to begin the transformation process:
 - To apply the transformation scenario to a newly opened file, use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the toolbar or the **XML** menu.
 - To customize the transformation or change the scenario that is associated with the document, use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu.
2. Select the **DocBook to DITA** transformation scenario in the **DocBook 4** or **DocBook 5** section.
3. Click the **Apply associated** button to run the transformation.

Step Result: The transformation will convert as many of the DocBook elements into equivalent DITA elements as it can recognize in its mapping process. For elements that cannot be mapped, the transformation will insert XML comments so that you can see which elements could not be converted.

4. Adjust the resulting DITA composite to suit your needs. You may have to remove comments, fix validation errors, adjust certain attributes, or split the content into individual topics.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

Inserting an olink in DocBook Documents

The `olink` element is used for linking to resources outside the current DocBook document. The `targetdoc` attribute is used for the document ID that contains the target element and the `targetptr` attribute for the ID of the target element (the value of an `id` or `xml:id` attribute). 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 which 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 `o link`. 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 `o links` 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 of a target database document. It structures a collection of documents in 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 agttargets 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">
            &agttargets;
          </document>
        </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 the **Parameters** tab, 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>
  <xrefext>How to administer user accounts</xrefext>
  <div element="part" href="#d5e4" number="I">
    <ttl>First Part</ttl>
    <xrefext>Part I, "First Part"</xrefext>
    <div element="chapter" href="#d5e6" number="1">
      <ttl>Chapter Title</ttl>
      <xrefext>Chapter 1, Chapter Title</xrefext>
      <div element="sect1" href="#src_chapter" number="1" targetptr="src_chapter">
        <ttl>Section1 Title</ttl>
        <xrefext>xreflabel_here</xrefext>
      </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** attributes, 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.

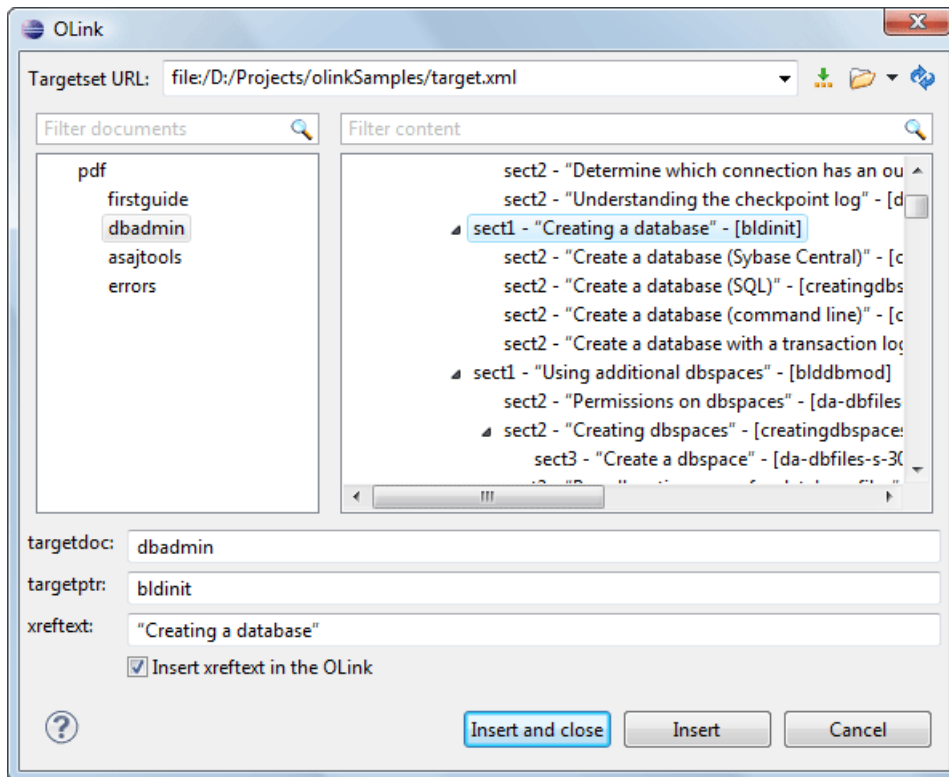


Figure 275: Insert OLink Dialog Box

6. Process a DocBook transformation for each document to generate the 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.

DocBook 5 Document Type

DocBook is a very popular set of tags for describing books, articles, and other prose documents, particularly technical documentation. DocBook provides a vast number of semantic element tags, divided into three broad categories: structural, block-level, and inline. DocBook content can then be published in a variety of formats, including HTML, PDF, WebHelp, and EPUB.

File Definition

A file is considered to be a DocBook 5 document when the namespace is *http://docbook.org/ns/docbook*.

Default Document Templates

There are a variety of default *DocBook 5* templates available when creating *new documents from templates*, including **Article**, **Book**, **Chapter**, **Glossary**, **Section**, and more.

The default templates for DocBook 5 documents are located in the `[OXYGEN_INSTALL_DIR]/frameworks/docbook/templates/Docbook 5` folder.



Note: Default templates for experimental DocBook 5.1 documents are also available in the `[OXYGEN_INSTALL_DIR]/frameworks/docbook/templates/Docbook 5.1 (Experimental)/` directory.

Default Schema

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_INSTALL_DIR]/frameworks/docbook/5.0/rng/`. Other types of schemas are also located in various folders inside the `[OXYGEN_INSTALL_DIR]/frameworks/docbook/5.0/` directory (for example, XML Schema files are located in the `xsd` folder).

Default XML Catalog

The default XML catalog, `catalog.xml`, is stored in `[OXYGEN_INSTALL_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_INSTALL_DIR]/frameworks/docbook/5.1/` directory.

Resources

- [DocBook Specifications](#)

Related information

[Editing XML Documents in Text Mode](#) on page 161

DocBook 5 Transformation Scenarios

Default transformation scenarios allow you to transform DocBook 5 documents to a variety of outputs, such as WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, and EPUB.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

DocBook 5 to WebHelp Output

DocBook 5 documents can be transformed into several types of WebHelp systems.

WebHelp Classic Output

To publish a DocBook 5 document as a *WebHelp Classic* system, follow these steps:

1. Click the  **Configure Transformation Scenario(s)** action from the toolbar.
2. Select the **DocBook WebHelp Classic** scenario from the **DocBook 5** section.
3. Click **Apply associated**.

When the **DocBook WebHelp Classic** transformation is complete, the output is automatically opened in your default browser.

WebHelp Classic with Feedback Output

To publish a DocBook 5 document as a *WebHelp Classic with Feedback* system, follow these steps:

1. Click  **Configure Transformation Scenarios**.

2. Select the **DocBook WebHelp Classic 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 Classic 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. Follow the instructions to complete the system deployment. For more information, see [Deploying a Feedback-Enabled WebHelp System](#) on page 563.

To watch our video demonstration about the feedback-enabled WebHelp system, go to https://www.oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

WebHelp Classic Mobile Output

To publish a DocBook 5 document as a *WebHelp Classic Mobile* system, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp Classic Mobile** scenario from the **DocBook 5** section.
3. Click **Apply associated**.

When the **DocBook WebHelp Classic Mobile** transformation is complete, the output is automatically opened in your default browser.

Customizing WebHelp Transformation Scenarios

To customize a DocBook WebHelp transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

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.

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.logo.image.target.url

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

webhelp.logo.image

Specifies a path to an image displayed as a logo in the left side of the output header.

webhelp.product.id (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

webhelp.product.version (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.skin.css

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.

For more information about all the DocBook transformation parameters, go to <http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html>.

Related information

[WebHelp Classic Output](#) on page 456

[WebHelp Classic With Feedback Output](#) on page 458

[WebHelp Classic Mobile System](#) on page 634

[Customizing WebHelp Classic Systems](#) on page 613

[Customizing WebHelp Classic Mobile Systems](#) on page 634

DocBook to PDF Output Customization

When the default layout and output of the DocBook to PDF transformation needs to be customized, follow these steps:

1. Create a custom version of the DocBook title spec file.

You should start from a copy of the file

[*OXYGEN_INSTALL_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_INSTALL_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 this:

```
<xsl:import href="dir-name/mytitlepages.xsl"/>
```

4. Insert a 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 to duplicate the transformation scenario **DocBook PDF** that is included with Oxygen XML Developer plugin and set the customization layer in [the XSL URL property of the scenario](#).

Related information

[The book *DocBook XSL: The Complete Guide* by Bob Stayton contains more details about customizing the PDF output.](#)
[Video demonstration for creating a DocBook customization layer in Oxygen XML Developer plugin.](#)

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. Open the **Configure Transformation Scenario(s)** dialog box, select the **DocBook EPUB** transformation scenario, and click **Edit**.
4. In the **Parameters** tab, 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.

5. Run the transformation scenario.

DocBook to DITA Transformation

Oxygen XML Developer plugin includes a built-in transformation scenario that is designed to convert DocBook content to DITA. This transformation scenario is based upon a DITA Open Toolkit plugin that is available at sourceforge.net.

To convert a DocBook document to DITA, follow these steps:

1. Use one of the following two methods to begin the transformation process:
 - To apply the transformation scenario to a newly opened file, use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the toolbar or the **XML** menu.
 - To customize the transformation or change the scenario that is associated with the document, use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu.
2. Select the **DocBook to DITA** transformation scenario in the **DocBook 4** or **DocBook 5** section.
3. Click the **Apply associated** button to run the transformation.

Step Result: The transformation will convert as many of the DocBook elements into equivalent DITA elements as it can recognize in its mapping process. For elements that cannot be mapped, the transformation will insert XML comments so that you can see which elements could not be converted.

4. Adjust the resulting DITA composite to suit your needs. You may have to remove comments, fix validation errors, adjust certain attributes, or split the content into individual topics.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

Inserting an olink in DocBook Documents

The `olink` element is used for linking to resources outside the current DocBook document. The `targetdoc` attribute is used for the document ID that contains the target element and the `targetptr` attribute for the ID of the target element (the value of an `id` or `xml:id` attribute). 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 which 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 of a target database document. It structures a collection of documents in 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 agttargets 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">
            &agttargets;
          </document>
        </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 the **Parameters** tab, 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>
  <xrefext>How to administer user accounts</xrefext>
  <div element="part" href="#d5e4" number="I">
    <ttl>First Part</ttl>
    <xrefext>Part I, "First Part"</xrefext>
    <div element="chapter" href="#d5e6" number="1">
      <ttl>Chapter Title</ttl>
      <xrefext>Chapter 1, Chapter Title</xrefext>
      <div element="sect1" href="#src_chapter" number="1" targetptr="src_chapter">
        <ttl>Section1 Title</ttl>
        <xrefext>xreflabel_here</xrefext>
      </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` attributes, 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 `xrefext` field.

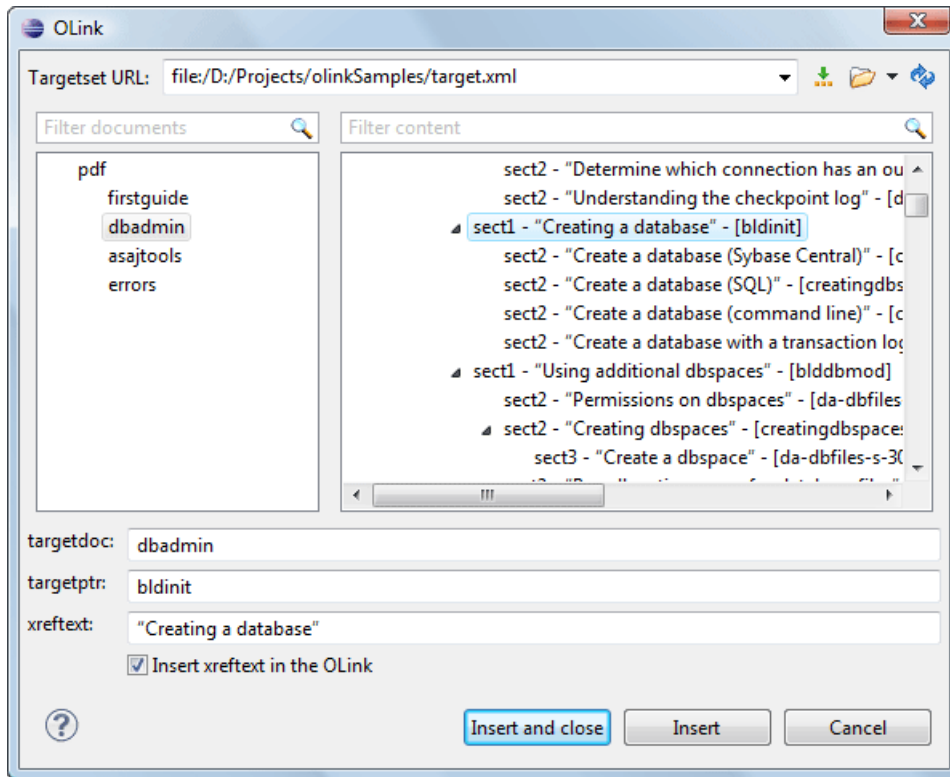


Figure 276: Insert OLink Dialog Box

6. Process a DocBook transformation for each document to generate the 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.

DocBook Targetset Document Type

DocBook *Targetset* documents are used to resolve cross references with the DocBook o link.

File Definition

A file is considered to be a *Targetset* when the root name is `targetset`.

Default Document Templates

A default **DocBook Targetset - Map** document template is available when creating *new documents from templates*.

The default template for DocBook Targetset documents is located in the `[OXYGEN_INSTALL_DIR]/frameworks/docbook/templates/Targetset` folder.

Default Schema

The default schema, `targetdatabase.dtd`, for this type of document is stored in `[OXYGEN_INSTALL_DIR]/frameworks/docbook/xsl/common/`.

Resources

- [DocBook Specifications](#)

DITA Topics Document Type

The Darwin Information Typing Architecture (DITA) is an XML-based architecture for authoring, producing, and delivering technical information. It divides content into small, self-contained topics that 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. DITA content is created as topics, each an individual XML file. Typically, each topic has a defined primary objective and structure, and DITA also includes several specialized topic types (*task*, *concept*, *reference*, *glossary entry*).

File Definition

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 Document Templates

There are a variety of default *DITA topic* templates available when creating [new documents from templates](#), including **Concept**, **Gossentry**, **Reference**, **Task**, **Topic**, and many more.

The default templates for DITA topic documents are located in the `[OXYGEN_INSTALL_DIR]/frameworks/dita/templates/topic` folder.

Default Schema

Default schemas that are used if one is not detected in the DITA documents are stored in the various folders inside `DITA_OT_DIR/dtd/` or `DITA_OT_DIR/schema/`.

Default XML Catalogs

The default catalogs for the DITA topic document type are as follows:

- `DITA_OT_DIR/catalog-dita.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/dita/catalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/dita/plugin/catalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/dita/styleguide/catalog.xml`

Resources

- [DITA Specifications](#)
- [DITA Style Guide Best Practices for Authors](#)
- [Oxygen Video Tutorial: DITA Editing](#)

Related information

[Editing XML Documents in Text Mode](#) on page 161

DITA Topic Transformation Scenarios

The following default transformation scenarios are available for individual 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.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

DITA Map Document Type

DITA maps are documents that collect and organize references to DITA topics to indicate the relationships between the topics. They can be used as a container for topics used to transform a collection of content into a publication and they offer a sequence and structure to the topics. They can also serve as outlines or tables of contents for DITA deliverables and as build manifests

for DITA projects. DITA maps allow scalable reuse of content across multiple contexts. Maps can reference topics or other maps, and can contain a variety of content types and metadata.

File Definition

A file is considered to be a DITA map document when one of the following conditions are 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` that 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/XML Schema processing in document type detection option](#) from the [Document Type Association preferences page](#) is enabled.

Default Document Templates

There are a variety of default *DITA map* templates available when creating [new documents from templates](#), including **Bookmap**, **Classification Map**, **Map**, **Subject Scheme**, and more.

The default templates for DITA map documents are located in the `[OXYGEN_INSTALL_DIR]/frameworks/sita/templates/map` folder.

Default Schema

Default schemas that are used if one is not detected in the DITA map document are stored in the various folders inside `DITA_OT_DIR/dtd/` or `DITA_OT_DIR/schema/`.

Default XML Catalogs

The default catalogs for the *DITA map* document type are as follows:

- `[OXYGEN_INSTALL_DIR]/frameworks/dita/catalog.xml`
- `DITA_OT_DIR/catalog-dita.xml`

Resources

- [DITA Specifications](#)
- [DITA Style Guide Best Practices for Authors](#)
- [Oxygen Video Tutorial: DITA Maps Manager](#)

Related information

[Editing XML Documents in Text Mode](#) on page 161

DITA Map Transformation Scenarios

The following types of transformations scenarios are available for DITA maps:

- Predefined transformation scenarios allow you to transform a DITA map to a variety of outputs, such as 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 plugins from the `DITA-OT/plugins` directory.
- **DITA Map Metrics Report** - Use this type of transformation scenario if you want to generate a DITA map statistics report. They contain 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.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

DITA Map to WebHelp Output

DITA maps can be transformed into a variety of WebHelp systems designed to suit your specific needs. This section contains the procedures for obtaining the output for the variants of the WebHelp system.

Related information

[WebHelp System Output](#) on page 557

WebHelp Responsive Output

To publish a DITA map as a *WebHelp Responsive* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Responsive** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- [Templates Tab](#) - This tab contains a set of predefined skins that you can use for the layout of your WebHelp system output.
- [Parameters Tab](#) - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
- [Filters Tab](#) - This tab allows you to filter certain content elements from the generated output.
- [Advanced Tab](#) - This tab allows you to specify some advanced options for the transformation scenario.
- [Output Tab](#) - This tab allows you to configure options that are related to the location where the output is generated.

4. Click **Apply associated** to process the transformation.

When the **DITA Map WebHelp Responsive** transformation is complete, the output is automatically opened in your default browser.

Parameters for Customizing WebHelp Responsive Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

webhelp.reload.stylesheet

Set this parameter to `true` if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is `false`.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as `'*'` (matches zero or more characters) or `'?'` (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.show.changes.and.comments

When set to `yes`, user comments, replies to comments, and tracked changes are published in the WebHelp output. The default value is `no`.

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). For example, 0.3, 0.5, or 0.8. 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`.

Parameters Specific to WebHelp Responsive Output**webhelp.fragment.after.body**

In the generated output it displays a given XHTML fragment after the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.logo_and_title

In the generated output it displays a given XHTML fragment after the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.main.page.search

In the generated output it displays a given XHTML fragment after the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.toc_or_tiles

In the generated output it displays a given XHTML fragment after the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.top_menu

In the generated output it displays a given XHTML fragment after the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.body

In the generated output it displays a given XHTML fragment before the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.logo_and_title

In the generated output it displays a given XHTML fragment before the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.main.page.search

In the generated output it displays a given XHTML fragment before the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.toc_or_tiles

In the generated output it displays a given XHTML fragment before the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.top_menu

In the generated output it displays a given XHTML fragment before the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.footer

In the generated output it displays a given XHTML fragment as the page footer. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.head

In the generated output it displays a given XHTML fragment as a page header. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.welcome

In the generated output it displays a given XHTML fragment as a welcome message (or title). The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.merge.nested.topics.related.links

Specifies if the related links from nested topics will be merged with the links in the parent topic. Thus the links will be moved from the topic content to the related links component and all of the links from the same group (for example, *Related Tasks*, *Related References*, *Related Information*) are merged into a single group. The default value is **yes**.

webhelp.show.breadcrumb

Specifies if the breadcrumb component will be presented in the output. The default value is **yes**.

webhelp.show.child.links

Specifies if child links will be generated in the output for all topics that have subtopics. The default value is **no**.

webhelp.show.indexterms.link

Specifies if an icon that links to the index will be presented in the output. The default value is **yes**.

webhelp.show.main.page.tiles

Specifies if the tiles component will be presented in the main page of the output. For a *tree* style layout, this parameter should be set to **no**.

webhelp.show.main.page.toc

Specifies if the table of contents will be presented in the main page of the output. The default value is **yes**.

webhelp.show.navigation.links

Specifies if navigation links will be presented in the output. The default value is **yes**.

webhelp.show.print.link

Specifies if a print link or icon will be presented within each topic in the output. The default value is **yes**.

webhelp.show.related.links

Specifies if the related links component will be presented in the WebHelp Responsive output. The default value is **yes**. The `webhelp.merge.nested.topics.related.links` parameter can be used in conjunction with this one to merge the related links from nested topics into the links in the parent topic.

webhelp.show.side.toc

Specifies if a side table of contents will be presented on the right side of each topic in the output. The default value is **yes**.

webhelp.show.top.menu

Specifies if a menu will be presented at the topic of the main page in the output. The default value is **yes**.

webhelp.top.menu.depth

Specifies the maximum depth level of the topics that will be included in the top menu. The default value is 2.

Related information

[Customizing the WebHelp Responsive Output](#) on page 578

[WebHelp Responsive System](#) on page 558

WebHelp Responsive with Feedback Output

To publish a DITA map as a *WebHelp Responsive with Feedback* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Responsive with Feedback** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- [Templates Tab](#) - This tab contains a set of predefined skins that you can use for the layout of your WebHelp system output.
 - [Parameters Tab](#) - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
 - [Filters Tab](#) - This tab allows you to filter certain content elements from the generated output.
 - [Advanced Tab](#) - This tab allows you to specify some advanced options for the transformation scenario.
 - [Output Tab](#) - This tab allows you to configure options that are related to the location where the output is generated.
4. Click **Apply associated** to begin the transformation.
 5. Enter the documentation product ID (value for the `webhelp.product.id` parameter) and the documentation version (value for the `webhelp.product.version` parameter).

When the **DITA Map WebHelp Responsive 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. Follow the [instructions to complete the system deployment](#).

Parameters for Customizing WebHelp Responsive with Feedback Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

webhelp.reload.stylesheet

Set this parameter to `true` if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is `false`.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as `'*'` (matches zero or more characters) or `'?'` (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.show.changes.and.comments

When set to `yes`, user comments, replies to comments, and tracked changes are published in the WebHelp output. The default value is `no`.

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). For example, 0.3, 0.5, or 0.8. 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`.

Parameters Specific to WebHelp Responsive with Feedback Output**webhelp.fragment.after.body**

In the generated output it displays a given XHTML fragment after the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.logo_and_title

In the generated output it displays a given XHTML fragment after the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.main.page.search

In the generated output it displays a given XHTML fragment after the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.toc_or_tiles

In the generated output it displays a given XHTML fragment after the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.top_menu

In the generated output it displays a given XHTML fragment after the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.body

In the generated output it displays a given XHTML fragment before the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.logo_and_title

In the generated output it displays a given XHTML fragment before the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.main.page.search

In the generated output it displays a given XHTML fragment before the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.toc_or_tiles

In the generated output it displays a given XHTML fragment before the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.top_menu

In the generated output it displays a given XHTML fragment before the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.footer

In the generated output it displays a given XHTML fragment as the page footer. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.head

In the generated output it displays a given XHTML fragment as a page header. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.welcome

In the generated output it displays a given XHTML fragment as a welcome message (or title). The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.merge.nested.topics.related.links

Specifies if the related links from nested topics will be merged with the links in the parent topic. Thus the links will be moved from the topic content to the related links component and all of the links from the same group (for example, *Related Tasks*, *Related References*, *Related Information*) are merged into a single group. The default value is **yes**.

webhelp.show.breadcrumb

Specifies if the breadcrumb component will be presented in the output. The default value is **yes**.

webhelp.show.child.links

Specifies if child links will be generated in the output for all topics that have subtopics. The default value is **no**.

webhelp.show.indexterms.link

Specifies if an icon that links to the index will be presented in the output. The default value is **yes**.

webhelp.show.main.page.tiles

Specifies if the tiles component will be presented in the main page of the output. For a *tree* style layout, this parameter should be set to **no**.

webhelp.show.main.page.toc

Specifies if the table of contents will be presented in the main page of the output. The default value is **yes**.

webhelp.show.navigation.links

Specifies if navigation links will be presented in the output. The default value is **yes**.

webhelp.show.print.link

Specifies if a print link or icon will be presented within each topic in the output. The default value is **yes**.

webhelp.show.related.links

Specifies if the related links component will be presented in the WebHelp Responsive output. The default value is **yes**. The `webhelp.merge.nested.topics.related.links` parameter can be used in conjunction with this one to merge the related links from nested topics into the links in the parent topic.

webhelp.show.side.toc

Specifies if a side table of contents will be presented on the right side of each topic in the output. The default value is **yes**.

webhelp.show.top.menu

Specifies if a menu will be presented at the topic of the main page in the output. The default value is **yes**.

webhelp.top.menu.depth

Specifies the maximum depth level of the topics that will be included in the top menu. The default value is 2.

Related information

[Customizing the WebHelp Responsive Output](#) on page 578

[WebHelp Responsive with Feedback System](#) on page 561

WebHelp Classic Output

To publish a DITA map as a *WebHelp Classic* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Classic** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- [Skins Tab](#) - This tab contains a set of predefined skins that you can use for the layout of your WebHelp system output.
- [Parameters Tab](#) - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
- [Filters Tab](#) - This tab allows you to filter certain content elements from the generated output.
- [Advanced Tab](#) - This tab allows you to specify some advanced options for the transformation scenario.
- [Output Tab](#) - This tab allows you to configure options that are related to the location where the output is generated.

4. Click **Apply associated** to process the transformation.

When the **DITA Map WebHelp Classic** transformation is complete, the output is automatically opened in your default browser.

To further customize this transformation, you can edit various parameters, including the following most commonly used ones:

Parameters for Customizing WebHelp Classic Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is **en-us**.

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**).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is **false**).

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.

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.google.search.results

A file path 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 recommended 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>`

webhelp.google.search.script

A file path that specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google.

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.logo.image.target.url

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

webhelp.logo.image

Specifies a path to an image displayed as a logo in the left side of the output header.

webhelp.reload.stylesheet

Set this parameter to `true` if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is `false`.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as `'*'` (matches zero or more characters) or `'?'` (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.show.changes.and.comments

When set to `yes`, user comments, replies to comments, and tracked changes are published in the WebHelp output. The default value is `no`.

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). For example, 0.3, 0.5, or 0.8. 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`.

Related information

[Customizing WebHelp Classic Systems](#) on page 613

[WebHelp Classic System](#) on page 602

WebHelp Classic With Feedback Output

To publish a DITA map as a *WebHelp Classic with Feedback* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Classic with Feedback** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- [Skins Tab](#) - This tab contains a set of predefined skins that you can use for the layout of your WebHelp system output.
 - [Parameters Tab](#) - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
 - [Filters Tab](#) - This tab allows you to filter certain content elements from the generated output.
 - [Advanced Tab](#) - This tab allows you to specify some advanced options for the transformation scenario.
 - [Output Tab](#) - This tab allows you to configure options that are related to the location where the output is generated.
4. Click **Apply associated** to begin the transformation.
 5. Enter the documentation product ID (value for the `webhelp.product.id` parameter) and the documentation version (value for the `webhelp.product.version` parameter).

When the **DITA Map WebHelp Classic 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. Follow the [instructions to complete the system deployment](#).

To watch our video demonstration about the feedback-enabled WebHelp system, go to https://www.oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

Parameters for Customizing WebHelp Classic with Feedback Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

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.

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.google.search.results

A file path 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>`

webhelp.google.search.script

A file path that specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google.

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.logo.image.target.url

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

webhelp.logo.image

Specifies a path to an image displayed as a logo in the left side of the output header.

webhelp.reload.stylesheet

Set this parameter to `true` if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is `false`.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as `'*'` (matches zero or more characters) or `'?'` (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.show.changes.and.comments

When set to `yes`, user comments, replies to comments, and tracked changes are published in the WebHelp output. The default value is `no`.

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). For example, 0.3, 0.5, or 0.8. 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`.

Related information

[Customizing WebHelp Classic Systems](#) on page 613

WebHelp Classic Mobile Output

To publish a DITA map as a *WebHelp Classic Mobile* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Classic Mobile** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- [Parameters Tab](#) - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
 - [Filters Tab](#) - This tab allows you to filter certain content elements from the generated output.
 - [Advanced Tab](#) - This tab allows you to specify some advanced options for the transformation scenario.
 - [Output Tab](#) - This tab allows you to configure options that are related to the location where the output is generated.
4. Click **Apply associated** to process the transformation.

When the **DITA Map WebHelp Classic Mobile** transformation is complete, the output is automatically opened in your default browser.

Parameters for Customizing WebHelp Classic Mobile Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.google.search.results

A file path 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 recommended 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>`

webhelp.google.search.script

A file path that specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google.

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.reload.stylesheet

Set this parameter to **true** if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is **false**.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as `'*'` (matches zero or more characters) or `'?'` (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

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). For example, 0.3, 0.5, or 0.8. 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`.

Related information

[Customizing WebHelp Classic Mobile Systems](#) on page 634

[WebHelp Classic Mobile System](#) on page 634

DITA Map to PDF Output Customization

Oxygen XML Developer plugin comes bundled with the DITA Open Toolkit that provides a mechanism for converting DITA maps to PDF output. There are numerous ways that you can customize the PDF output and the topics in this section discuss some of those possibilities.

Creating a DITA Map to PDF Transformation Scenario

To create a *DITA Map to PDF* transformation scenario, follow these steps:

1. Click the  **Configure Transformation Scenario(s)** button.
2. Select **DITA Map PDF** and click the **Edit** button (or use the **Duplicate** button if your framework is read-only).
3. Use the various tabs to configure the transformation scenario. In the **Parameters** tab, you can use a variety of parameters to customize the output. For example, the following parameters are just a few of the most commonly used ones:
 - `show.changes.and.comments` - If set to **yes**, user comments, replies to comments, and tracked changes are published in the PDF output.
 - `customization.dir` - Specifies the path to a customization directory.
4. Click **OK** and then the **Apply Associated** button to run the transformation scenario.

Creating a Customization Directory for PDF Output

DITA Open Toolkit PDF output can be customized in several ways:

- Creating a DITA Open Toolkit plugin that adds extensions to the PDF plugin. More details can be found in the [DITA Open Toolkit Documentation](#).
- Creating a customization directory and using it from the PDF transformation scenario. A small example of this procedure can be found below.

How to Create a Customization Directory for PDF Output

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 Developer plugin and *uncomment* this line:

```
<!--uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"/-->
```

It now 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 Developer plugin and create the template called `createFrontCoverContents` for DITA-OT 2.3.3 (or `createFrontMatter_1.0` for DITA-OT 1.8.5).



Tip: You can copy the same template from `DITA_OT_DIR\plugins\org.dita.pdf2\xsl\fo\front-matter.xsl` and modify it in whatever way necessary to achieve your specific goal. This new template in the `custom.xsl` file will override the same template from `DITA_OT_DIR\plugins\org.dita.pdf2\xsl\fo\front-matter.xsl`.

DITA OT 2.3.3 Example:

For example, if you are using DITA OT 2.3.3, the `custom.xsl` could look like this:

```
<?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="2.0">

<xsl:template name="createFrontCoverContents">
  <!-- 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>

    <!-- Load the image logo -->
    <fo:block text-align="center" width="100%">
      <fo:external-graphic
        src="url({concat($artworkPrefix,
          '/Customization/OpenTopic/common/artwork/logo.png')})"
      />
    </fo:block>
  </xsl:template>
</xsl:stylesheet>
```

DITA OT 1.8.5 Example:

For example, if you are using DITA OT 1.8.5, the `custom.xsl` could look like this:

```
<?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>
      </fo:flow>
    </fo:page-sequence>
  </xsl:template>
```



```

</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 PDF** transformation scenario and in the **Parameters** tab, set the customization.dir parameter to C:\Customization.

Related information

[Automatic PDF plugin customization generator by Jarno Elovirta.](#)

[DITA OT Documentation - PDF Customization Plugin](#)

Customizing the Header and Footer in PDF Output

The XSLT stylesheet *DITA_OT_DIR/plugins/org.dita.pdf2/xsl/fo/static-content.xsl* contains templates that output the static header and footers for various parts of the PDF such as 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 that 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 (such as *Body odd footer*) that 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](#).

Adding a Watermark to PDF Output

To add a watermark to the PDF output of a DITA map transformation, create a DITA-OT customization directory and use it from a DITA Map to PDF transformation scenario, as in the following procedure:

1. Copy the entire directory: *DITA_OT_DIR\plugins\org.dita.pdf2\Customization* to another location (for instance, C:\Customization).
2. Copy your watermark image (for example, *watermark.png*) to:
C:\Customization\common\artwork\watermark.png.
3. Rename the C:\Customization\catalog.xml.orig file to: C:\Customization\catalog.xml.
4. Open the *catalog.xml* in Oxygen XML Developer plugin and *uncomment* this line:

```
<!--uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"/-->
```

The uncommented line should look 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 C:\Customization\fo\xsl\custom.xsl file in Oxygen XML Developer plugin to overwrite two XSLT templates:
 - The first template is located in the XSLT stylesheet *DITA_OT_DIR\plugins\org.dita.pdf2\xsl\fo\static-content.xsl* and we override it specifying

a watermark image for every page in the PDF content, using a *block-container* element that references the watermark image file:

```
<fo:static-content flow-name="odd-body-header">
  <fo:block-container absolute-position="absolute"
    top="-2cm" left="-3cm" width="21cm" height="29.7cm"
    background-image="{concat($artworkPrefix,
'/Customization/OpenTopic/common/artwork/watermark.png')}">
    <fo:block/>
  </fo:block-container>
  <fo:block xsl:use-attribute-sets="__body_odd_header">
    <xsl:call-template name="insertVariable">
      <xsl:with-param name="theVariableID" select="'Body odd header'"/>
      <xsl:with-param name="theParameters">
        <prodname>
          <xsl:value-of select="$productName"/>
        </prodname>
        <heading>
          <fo:inline xsl:use-attribute-sets="__body_odd_header_heading">
            <fo:retrieve-marker retrieve-class-name="current-header"/>
          </fo:inline>
        </heading>
        <pagenum>
          <fo:inline xsl:use-attribute-sets="__body_odd_header_pagenum">
            <fo:page-number/>
          </fo:inline>
        </pagenum>
      </xsl:with-param>
    </xsl:call-template>
  </fo:block>
</fo:static-content>
</xsl:template>
```

- The second template that we override is located in the XSLT stylesheet *DITA_OT_DIR\plugins\org.dita.pdf2\xsl\fo\commons.xsl* and is used for styling the first page of the output. We also override it by copying the original template content in our *custom.xsl* and adding the *block-container* element that references the watermark image file:

```
<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-container absolute-position="absolute"
        top="-2cm" left="-3cm" width="21cm" height="29.7cm"
        background-image="{concat($artworkPrefix,
'/Customization/OpenTopic/common/artwork/watermark.png')}">
        <fo:block/>
      </fo:block-container>
      <fo:block xsl:use-attribute-sets="__frontmatter">
        <!-- set the title -->
        <fo:block xsl:use-attribute-sets="__frontmatter_title">
          <xsl:choose>
            <xsl:when test="$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="$map/*[contains
(@class,' bookmap/bookmeta ')]" />
        </fo:block>
      </fo:block>
      <!--<xsl:call-template name="createPreface"/>-->
    </fo:flow>
  </fo:page-sequence>
  <xsl:if test="not($retain-bookmap-order)">
    <xsl:call-template name="createNotices"/>
  </xsl:if>
</xsl:template>
```

```
</xsl:if>
</xsl:template>
```

7. Edit your **DITA Map PDF** transformation scenario. In the **Parameters** tab, set the customization.dir parameter to C:\Customization.

Force Page Breaks Between Two Block Elements in PDF Output

Suppose that 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 (for example, `DITA_OT_DIR/plugins/pdf-page-break`).
3. In this new folder, create a new `plugin.xml` file with the following 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>
```

Customizing note Images in PDF

To customize the images that appear next to each type of note in the PDF output, use a [PDF customization folder](#) with the following procedure:

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 Map PDF** transformation scenario and in the **Parameters** tab set the path for the `customization.dir` property to point to the customization folder.

Show Comments and Tracked Changes in PDF Output

To make comments and tracked changes (stored within your DITA topics) appear in the PDF output, follow these steps:

1. Click the  **Configure Transformation Scenario(s)** button.
2. Select **DITA Map PDF** and click the **Edit** button (or use the **Duplicate** button if your framework is read-only).
3. In the **Parameters** tab, set the value of the `show.changes.and.comments` parameter to **yes**.
4. Click **OK** and then the **Apply Associated** button to run the transformation scenario.

Result: Comment threads and tracked changes will now appear in the PDF output. Details about each comment or change will be available in the footer section for each page.

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 [Add a Font to the Built-in FO Processor](#) on page 554.

DITA OT PDF Font Mapping

The DITA OT contains a file `DITA_OT_DIR/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml` that maps logical fonts used in the XSLT stylesheets to physical fonts that 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 this:

```
<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. Therefore, 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**.

Related information

<http://www.elovirta.com/2016/02/18/font-configuration-in-pdf2.html>

DITA Map to PDF WYSIWYG Transformation

Oxygen XML Developer plugin comes bundled with a DITA OT plugin that converts DITA maps to PDF using a CSS layout processor. Oxygen XML Developer plugin supports the following processors (not included in the Oxygen XML Developer 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 Developer 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.
2. Select **DITA Map PDF - WYSIWYG - Experimental**.
3. 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**, user comments, replies to comments, and tracked changes are published in the PDF output. The default value is **no**.
 - `dita.css.list` - Allows you to specify a list of CSS URLs to be used by the PDF processor. The files must have URL syntax and be separated using semicolons.
 - `args.css` - Allows you to specify a list of CSS URLs to be used in addition to those specified in the `dita.css.list` parameter.
4. Click **OK** and run the transformation scenario.

Customizing the Styles (for Output and Editing)

If you need to change the styles in the associated CSS, make sure you install Oxygen XML Developer plugin in a folder where you have full read and write privileges (for instance, your user home directory). This is due to the fact that the installed files are usually placed in a read-only folder (for instance, in Windows, Oxygen XML Developer plugin is installed in the **Program Files** folder where the users do not have change rights).

To change the styles of an element you need to create an additional CSS file that will store the customization rules. Once you have created this file, you need to instruct the editor how to use this additional CSS.

- Use the `args.css` parameter that allows you to specify a list of CSS URLs to be used in addition to the `dita.css.list` parameter:
 1. Configure a *DITA map to PDF WYSIWYG* transformation scenario, as described in the procedure [above](#).
 2. In the **Parameters** tab, specify the path to your custom CSS files in the `args.css` parameter.
 3. Click **OK** and run the transformation scenario.

This method is appropriate if you just want to apply the styling customization to the output.

How to Make Remote Resources Appear in the Output When Using the Prince Processor

If your documentation references external resources (such as images that are stored on a Web-based repository) and your system is behind an HTTP(S) proxy, you may find that they do not appear in the output when using the **Prince Print with CSS** processor. To solve this, follow this procedure:

1. Edit the **build.xml** file that is located in `DITA_OT_DIR/plugins/com.oxygenxml.pdf.css`.
2. There are two instances in the file where a pair of arguments are commented out:

```
<!-- Please remove the comment wrapping the following two arguments
if you are behind a proxy and your documentation refers remote resources,
for example images.

Note: You also need to remove the backslash '\' that appears before the
property name: "http-proxy". That backslash is there because a sequence of
two dashes ('-') is not permitted inside a comment.
-->
<!--
  <arg value="-\-http-proxy=${http.proxyHost}:${http.proxyPort}"/>
```

```
<arg value="-\-http-proxy=${https.proxyHost}:${https.proxyPort}"/>
-->
```

3. Remove the comment in both instances.
4. Make sure you also remove the slash that appears before the property name `http-proxy` in each instance.

Step Result: The arguments should now look like this:

```
<arg value="-http-proxy=${http.proxyHost}:${http.proxyPort}"/>
<arg value="-http-proxy=${https.proxyHost}:${https.proxyPort}"/>
```

5. Save the `build.xml` file.
6. Run the *DITA map to PDF WYSIWYG transformation scenario*.

Result: Your external resources should now appear in your output.

Related information

Editing a Transformation Scenario on page 543

Configure Transformation Scenario(s) Dialog Box on page 545

Compiled HTML Help (CHM) Output Format

To perform a **Compiled HTML Help (CHM)** transformation Oxygen XML Developer plugin needs Microsoft HTML Help Workshop to be installed on your computer. Oxygen XML Developer plugin automatically detects HTML Help Workshop and uses it.



Note: HTML Help Workshop might fail if the files used for transformation contain accents in their names, due to different encodings used when writing the `.hhp` and `.hhc` files. If the transformation fails to produce the CHM output but the `.hhp` (HTML Help Project) file is already generated, you can manually try to build the CHM output using HTML Help Workshop.

Changing the Output Encoding

Oxygen XML Developer plugin uses the `htmlhelp.locale` parameter to correctly display specific characters of different languages in the output of the **Compiled HTML Help (CHM)** transformation. The **Compiled HTML Help (CHM)** default scenario that comes bundled with Oxygen XML Developer plugin has the `htmlhelp.locale` parameter set to `en-US`.

The default value of the `htmlhelp.locale` is `en-US`. To customize this parameter, go to **Configure Transformation Scenarios** and click the **Edit** button. In the parameter tab search for the `htmlhelp.locale` parameter and change its value to the desired language tag.

The format of the `htmlhelp.locale` parameter is `LL-CC`, where `LL` represents the language code (`en`, for example) and `CC` represents the country code (`US`, for example). The language codes are contained in the ISO 639-1 standard and the country codes are contained in the ISO 3166-1 standard. For further details about language tags, go to <http://www.rfc-editor.org/rfc/rfc5646.txt>.

Kindle Output Format

Oxygen XML Developer plugin requires KindleGen to generate Kindle output from DITA maps. To install KindleGen for use by Oxygen XML Developer plugin, follow these steps:

1. Go to www.amazon.com/kindleformat/kindlegen and download the zip file that matches your operating system.
2. Unzip the file on your local disk.
3. Start Oxygen XML Developer plugin and open a DITA map in the **DITA Maps Manager** view.
4. In the **DITA Maps Manager** view, open the **Configure Transformation Scenario(s)** dialog box.
5. Select the **DITA Map Kindle** transformation and press the **Edit** button to edit it.
6. Go to **Parameters** tab and set the `kindlegen.executable` parameter as the path to the *KindleGen* directory.
7. Accept the changes.

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.

File Definition

A file is considered to be an XHTML document when the root element name is `html`.

Default Document Templates

There are a variety of default *XHTML* templates available when creating *new documents from templates*, including **1.0 Strict**, **1.0 Transitional**, **1.1 DTD Based**, **1.1 Schema Based**, and more.

The default templates for XHTML documents are located in the `[OXYGEN_INSTALL_DIR]/frameworks/xhtml/templates/` folder.

Default Schema

Default schemas that are used if one is not detected in the XHTML file are stored in the following locations:

- XHTML 1.0 - `[OXYGEN_INSTALL_DIR]/frameworks/xhtml/dtd/` or `[OXYGEN_INSTALL_DIR]/frameworks/xhtml/nvdl/`.
- XHTML 1.1 - `[OXYGEN_INSTALL_DIR]/frameworks/xhtml11/dtd/` or `[OXYGEN_INSTALL_DIR]/frameworks/xhtml11/schema/`.
- XHTML 5 - `[OXYGEN_INSTALL_DIR]/frameworks/xhtml/xhtml5 (epub3)/`.

Default XML Catalogs

The default catalogs for the XHTML document type are as follows:

- `[OXYGEN_INSTALL_DIR]/frameworks/xhtml/dtd/xhtmlcatalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/relaxng/catalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/nvdl/catalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/xhtml11/dtd/xhtmlcatalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/xhtml11/schema/xhtmlcatalog.xml`
- `[OXYGEN_INSTALL_DIR]/xhtml5 (epub3)/catalog-compatible.xml`

Resources

- [XHTML Specifications](#)

Related information

[Editing XHTML Documents](#) on page 404

[Editing XML Documents in Text Mode](#) on page 161

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.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

TEI ODD Document Type

The *TEI ODD (Text Encoding Initiative - One Document Does it all)* document type 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.

File Definition

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`.

Default Document Templates

There is a default *TEI ODD* document template available when creating [new documents from templates](#).

The default template is located in the `[OXYGEN_INSTALL_DIR]/frameworks/tei/templates/TEI ODD` folder.

Default Schema

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_INSTALL_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/`.

Default XML Catalogs

The default catalogs for the TEI ODD document type are as follows:

- `[OXYGEN_INSTALL_DIR]/frameworks/tei/xml/tei/custom/schema/catalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/tei/xml/tei/schema/catalog.xml`

Resources

-
- [TEI: Getting Started with ODD](#)

Related information

[Editing XML Documents in Text Mode](#) on page 161

TEI ODD Transformation Scenarios

The following default transformations are available for TEI ODD document types:

- **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

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

TEI P4 Document Type

The *TEI (Text Encoding Initiative)* document type 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.

File Definition

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`.

Default Document Templates

Some default *TEI P4* templates are available when creating [new documents from templates](#), including **Lite** and **P4 Document**.

The default templates for TEI P4 documents are located in the `[OXYGEN_INSTALL_DIR]/frameworks/tei/templates/TEI P4` folder.

Default Schema

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_INSTALL_DIR]/frameworks/tei/xml/teip4/schema/dtd/`.

Default XML Catalogs

The default catalogs for the TEI P4 document type are as follows:

- `[OXYGEN_INSTALL_DIR]/frameworks/tei/xml/teip4/schema/dtd/catalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/tei/xml/teip4/custom/schema/dtd/catalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/tei/xml/teip4/stylesheet/catalog.xml`

Resources

-

Related information

[Editing XML Documents in Text Mode](#) on page 161

TEI P4 Transformation Scenarios

The following default transformations are available for TEI P4 documents:

- **TEI HTML** - Transforms a TEI document into an HTML document.
- **TEI P4 - TEI P5 Conversion** - Convert a TEI P4 document into a TEI P5 document.
- **TEI PDF** - Transforms a TEI document into a PDF document using the Apache FOP engine.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

TEI P5 Document Type

The *TEI (Text Encoding Initiative)* document type 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.

File Definition

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`.

Default Document Templates

There are a variety of default *TEI P5* templates available when creating [new documents from templates](#), including **Bare**, **Lite**, **Math**, **Speech**, and more.

The default templates for TEI P5 documents are located in the `[OXYGEN_INSTALL_DIR]/frameworks/tei/templates/TEI P5` folder.

Default Schema

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_INSTALL_DIR]/frameworks/tei/xml/tei/custom/schema/relaxng/`.

Default XML Catalogs

The default catalogs for the TEI P5 document type are as follows:

- `[OXYGEN_INSTALL_DIR]/frameworks/tei/xml/tei/schema/dtd/catalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/tei/xml/tei/custom/schema/dtd/catalog.xml`
- `[OXYGEN_INSTALL_DIR]/frameworks/tei/xml/tei/stylesheet/catalog.xml`

Resources

-
- [TEI: P5 Guidelines](#)

Related information

[Editing XML Documents in Text Mode](#) on page 161

TEI P5 Transformation Scenarios

The following default transformations are available for TEI P5 documents:

- **TEI P5 XHTML** - Transforms a TEI P5 document into an XHTML document.
- **TEI P5 PDF** - Transforms a TEI P5 document into a PDF document using the Apache FOP engine.
- **TEI EPUB** - Transforms a TEI P5 document into an EPUB output. The EPUB output will contain any images referenced in the TEI XML document.
- **TEI DOCX** - Transforms a TEI P5 document into a DOCX (OOXML) document. The DOCX document will contain any images referenced in the TEI XML document.
- **TEI ODT** - Transforms a TEI P5 document into an ODT (ODF) document. The ODT document will contain any images referenced in the TEI XML document.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

Customization of TEI Frameworks Using the Compiled Sources

The following procedure describes how to update to the latest stable version of TEI Schema and TEI XSL, already integrated in the TEI framework for Oxygen XML Developer plugin.

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 Developer plugin `frameworks` folder.

JATS Document Type

The *JATS (NISO Journal Article Tag Suite)* document type is a technical standard that defines an XML format for scientific literature.

File Definition

A file is considered to be a JATS document when the PUBLIC ID of the document contains the string `-//NLM//DTD`.

Default Document Templates

There are some default *JATS* templates available when creating [new documents from templates](#), including **Archiving**, **Authoring**, **Book**, and **Publishing**.

The default templates for JATS documents are located in the `[OXYGEN_INSTALL_DIR]/frameworks/jats/templates/` folder.

Default Schema

Default schemas that are used if one is not detected in the JATS document are stored in `[OXYGEN_INSTALL_DIR]/frameworks/jats/lib/schemas/`.

Default XML Catalog

The default XML catalog, `jatskit-catalog.xml`, is stored in `[OXYGEN_INSTALL_DIR]/frameworks/jats/lib/schemas/`.

Resources

-
- [NLM Journal Archiving and Interchange Tag Suite](#)

Related information

[Editing XML Documents in Text Mode](#) on page 161

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.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

EPUB Document Type

EPUB is an e-book file format that is a ZIP archive and can be downloaded and read on devices such as phones, tablets, computers, or e-readers. You can [view the contents and structure of this type of file](#) in the **Navigator** view or main editing pane.

Three distinct frameworks are supported for the EPUB document type:

- **NCX** - A declarative global navigation definition.
- **OCF** - The Open Container Format (OCF) defines a mechanism by which all components of an Open Publication Structure (OPS) can be combined into a single file system entity.
- **OPF** - The Open Packaging Format (OPF) defines the mechanism by which all components of a published work that conforms to the Open Publication Structure (OPS) standard (including metadata, reading order, and navigational information) are packaged in an OPS Publication.



Note: Oxygen XML Developer plugin supports both OPF 2.0 and OPF 3.0.

File Definition

A file is considered to be an *EPUB* document if it has a file extension of `.epub`.

Default Document Templates

There are a variety of default *EPUB* templates available when creating [new documents from templates](#), including **NCX - Toc**, **OCF - Container**, **OCF - Encryption**, **OCF - Signatures**, **OPF 2.0 - Content**, and **OPF 3.0 - Content**.

The default templates for the *NCX* document types are located in the `[OXYGEN_INSTALL_DIR]/frameworks/ncx/templates` folder.

The default templates for the *OCF* document types are located in the `[OXYGEN_INSTALL_DIR]/frameworks/ocf/templates` folder.

The default template for the *OPF 2.0* document type is located in the `[OXYGEN_INSTALL_DIR]/frameworks/opf/templates/2.0` folder.

The default template for the *OPF 3.0* document type is located in the `[OXYGEN_INSTALL_DIR]/frameworks/opf/templates/3.0` folder.

Related information

[Working with Archive Files](#) on page 683

Chapter

9

Transforming Documents

Topics:

- [Transformation Scenarios](#)
- [WebHelp System Output](#)

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*.

Oxygen XML Developer plugin allows you to use transformation scenarios to publish XML content in various output formats (such as WebHelp, PDF, CHM, EPUB, JavaHelp, Eclipse Help, XHTML, etc.)

For transformations that are not included in your installed version of Oxygen XML Developer plugin, simply install the tool chain required to perform the specific transformation and process the files in accordance with the processor instructions. A multitude of target formats are possible. The basic condition for transformation to any format is that your source document is well-formed.



Note: You need to use the appropriate stylesheet according to the source definition and the desired output. For example, if you want to transform into an HTML format using a DocBook stylesheet, your source XML document should conform with the DocBook DTD.

Transformation Scenarios

A transformation scenario is a set of complex operations and settings that gives you the possibility to obtain outputs of multiple types (XML, HTML, PDF, EPUB, etc.) from the same source of XML files and stylesheets.

Executing a transformation scenario implies multiple actions, such as:

- Validating the input file.
- Obtaining intermediate output files (for example, formatting objects for the XML to PDF transformation).
- Using transformation engines to produce the output.

Before transforming an XML document in Oxygen XML Developer plugin, you need to define a transformation scenario to apply to that document. A scenario is a set of values for various parameters that define a transformation. It is not related to a particular document, but rather to a document type. Types of transformation scenarios include:

- **Scenarios that Apply to XML Files** - This type of scenario contains the location of an XSLT stylesheet that is applied on the edited XML document, as well as other transformation parameters.
- **Scenarios that Apply to XSLT Files** - This type of scenario contains the location of an XML document that the edited XSLT stylesheet is applied to, as well as other transform parameters.
- **Scenarios that Apply to XQuery Files** - This type of scenario contains the location of an XML source, that the edited XQuery file is applied to, as well as other transform parameters. When the XML source is a native XML database, the XML source field of the scenario is empty because the XML data is read with XQuery-specific functions, such as `document()`. When the XML source is a local XML file, the URL of the file is specified in the XML input field of the scenario.
- **Scenarios that Apply to SQL Files** - This type of scenario specifies a database connection for the database server that runs the SQL file that is associated with the scenario. The data processed by the SQL script is located in the database.
- **Scenarios that Apply to XProc Files** - This type of scenario contains the location of an XProc script, as well as other transform parameters.
- **DITA-OT Scenarios** - This type of scenario provides the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Developer plugin includes 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](#) (the [Enable oXygen consoles option](#) must be enabled in the [View preferences page](#)).

Built-in Transformation Scenarios

Oxygen XML Developer plugin included preconfigured built-in transformation scenarios that are used for common transformations. To obtain the desired output, use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the toolbar or the **XML** menu and choose one of the built-in scenarios for the current document.

You can use the  **Apply Transformation Scenario(s)** action even if the current document is not associated with a transformation scenario.

If the document contains an `xml-stylesheet` processing instruction that refers to an XSLT stylesheet (commonly used to display the document in web browsers), Oxygen XML Developer plugin prompts you to associate the document with a built-in transformation scenario.

The default transformation scenario is suggested based on the processing instruction from the edited document. The **XSL URL** field of the default transformation scenario contains the URL from the `href` attribute of the processing instruction. By default, the **Use xml-stylesheet declaration** checkbox 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}`.

DocBook 4 Transformation Scenarios

Default transformation scenarios allow you to convert DocBook 4 to DocBook 5 documents and transform DocBook documents to a variety of outputs, such as WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, and EPUB.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

DocBook 4 to WebHelp Output

DocBook 4 documents can be transformed into several types of WebHelp systems.

WebHelp Classic Output

To publish a DocBook 4 document as a *WebHelp Classic* system, follow these steps:

1. Click the  **Configure Transformation Scenario(s)** action from the toolbar.
2. Select the **DocBook WebHelp Classic** scenario from the **DocBook 4** section.
3. Click **Apply associated**.

When the **DocBook WebHelp Classic** transformation is complete, the output is automatically opened in your default browser.

WebHelp Classic with Feedback Output

To publish a DocBook 4 document as a *WebHelp Classic with Feedback* system, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp Classic 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 Classic 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. Follow the instructions to complete the system deployment. For more information, see [Deploying a Feedback-Enabled WebHelp System](#) on page 563.

To watch our video demonstration about the feedback-enabled WebHelp system, go to https://www.oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

WebHelp Classic Mobile Output

To publish a DocBook 4 document as a *WebHelp Classic Mobile* system, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp Classic Mobile** scenario from the **DocBook 4** section.
3. Click **Apply associated**.

When the **DocBook WebHelp Classic Mobile** transformation is complete, the output is automatically opened in your default browser.

Customizing WebHelp Transformation Scenarios

To customize a DocBook WebHelp transformation scenario, you can edit various parameters, including the following most commonly used ones:

`args.default.language`

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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**).

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.

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is **false**).

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.logo.image.target.url

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

webhelp.logo.image

Specifies a path to an image displayed as a logo in the left side of the output header.

webhelp.product.id (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: **< > / \ ' () { } = ; * % + &**.

webhelp.product.version (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.skin.css

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.

For more information about all the DocBook transformation parameters, go to <http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html>.

Related information

[WebHelp Classic Output](#) on page 456

[WebHelp Classic With Feedback Output](#) on page 458

[WebHelp Classic Mobile System](#) on page 634

[Customizing WebHelp Classic Systems](#) on page 613

[Customizing WebHelp Classic Mobile Systems](#) on page 634

DocBook to PDF Output Customization

When the default layout and output of the DocBook to PDF transformation needs to be customized, follow these steps:

1. Create a custom version of the DocBook title spec file.

You should start from a copy of the file

`[OXYGEN_INSTALL_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_INSTALL_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 this:

```
<xsl:import href="dir-name/mytitlepages.xsl"/>
```

4. Insert a 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 to duplicate the transformation scenario **DocBook PDF** that is included with Oxygen XML Developer plugin and set the customization layer in [the XSL URL property of the scenario](#).

Related information

The book [DocBook XSL: The Complete Guide](#) by Bob Stayton contains more details about customizing the PDF output. [Video demonstration for creating a DocBook customization layer in Oxygen XML Developer plugin](#).

DocBook to DITA Transformation

Oxygen XML Developer plugin includes a built-in transformation scenario that is designed to convert DocBook content to DITA. This transformation scenario is based upon a DITA Open Toolkit plugin that is available at sourceforge.net.

To convert a DocBook document to DITA, follow these steps:

1. Use one of the following two methods to begin the transformation process:
 - To apply the transformation scenario to a newly opened file, use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the toolbar or the **XML** menu.
 - To customize the transformation or change the scenario that is associated with the document, use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu.
2. Select the **DocBook to DITA** transformation scenario in the **DocBook 4** or **DocBook 5** section.
3. Click the **Apply associated** button to run the transformation.

Step Result: The transformation will convert as many of the DocBook elements into equivalent DITA elements as it can recognize in its mapping process. For elements that cannot be mapped, the transformation will insert XML comments so that you can see which elements could not be converted.

4. Adjust the resulting DITA composite to suit your needs. You may have to remove comments, fix validation errors, adjust certain attributes, or split the content into individual topics.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

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. Open the **Configure Transformation Scenario(s)** dialog box, select the **DocBook EPUB** transformation scenario, and click **Edit**.
4. In the **Parameters** tab, 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.

5. Run the transformation scenario.

DocBook 5 Transformation Scenarios

Default transformation scenarios allow you to transform DocBook 5 documents to a variety of outputs, such as WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, and EPUB.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

DocBook 5 to WebHelp Output

DocBook 5 documents can be transformed into several types of WebHelp systems.

WebHelp Classic Output

To publish a DocBook 5 document as a *WebHelp Classic* system, follow these steps:

1. Click the  **Configure Transformation Scenario(s)** action from the toolbar.
2. Select the **DocBook WebHelp Classic** scenario from the **DocBook 5** section.
3. Click **Apply associated**.

When the **DocBook WebHelp Classic** transformation is complete, the output is automatically opened in your default browser.

WebHelp Classic with Feedback Output

To publish a DocBook 5 document as a *WebHelp Classic with Feedback* system, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp Classic 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 Classic 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. Follow the instructions to complete the system deployment. For more information, see [Deploying a Feedback-Enabled WebHelp System](#) on page 563.

To watch our video demonstration about the feedback-enabled WebHelp system, go to https://www.oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

WebHelp Classic Mobile Output

To publish a DocBook 5 document as a *WebHelp Classic Mobile* system, follow these steps:

1. Click  **Configure Transformation Scenarios**.
2. Select the **DocBook WebHelp Classic Mobile** scenario from the **DocBook 5** section.
3. Click **Apply associated**.

When the **DocBook WebHelp Classic Mobile** transformation is complete, the output is automatically opened in your default browser.

Customizing WebHelp Transformation Scenarios

To customize a DocBook WebHelp transformation scenario, you can edit various parameters, including the following most commonly used ones:

`args.default.language`

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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**).

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.

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is **false**).

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.logo.image.target.url

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

webhelp.logo.image

Specifies a path to an image displayed as a logo in the left side of the output header.

webhelp.product.id (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: **< > / \ ' () { } = ; * % + &**.

webhelp.product.version (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.skin.css

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.

For more information about all the DocBook transformation parameters, go to <http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html>.

Related information

[WebHelp Classic Output](#) on page 456

[WebHelp Classic With Feedback Output](#) on page 458

[WebHelp Classic Mobile System](#) on page 634

[Customizing WebHelp Classic Systems](#) on page 613

[Customizing WebHelp Classic Mobile Systems](#) on page 634

DocBook to PDF Output Customization

When the default layout and output of the DocBook to PDF transformation needs to be customized, follow these steps:

1. Create a custom version of the DocBook title spec file.

You should start from a copy of the file

`[OXYGEN_INSTALL_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_INSTALL_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 this:

```
<xsl:import href="dir-name/mytitlepages.xsl"/>
```

4. Insert a 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 to duplicate the transformation scenario **DocBook PDF** that is included with Oxygen XML Developer plugin and set the customization layer in [the XSL URL property of the scenario](#).

Related information

The book [DocBook XSL: The Complete Guide](#) by Bob Stayton contains more details about customizing the PDF output. [Video demonstration for creating a DocBook customization layer in Oxygen XML Developer plugin](#).

DocBook to DITA Transformation

Oxygen XML Developer plugin includes a built-in transformation scenario that is designed to convert DocBook content to DITA. This transformation scenario is based upon a DITA Open Toolkit plugin that is available at sourceforge.net.

To convert a DocBook document to DITA, follow these steps:

1. Use one of the following two methods to begin the transformation process:
 - To apply the transformation scenario to a newly opened file, use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the toolbar or the **XML** menu.
 - To customize the transformation or change the scenario that is associated with the document, use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu.
2. Select the **DocBook to DITA** transformation scenario in the **DocBook 4** or **DocBook 5** section.
3. Click the **Apply associated** button to run the transformation.

Step Result: The transformation will convert as many of the DocBook elements into equivalent DITA elements as it can recognize in its mapping process. For elements that cannot be mapped, the transformation will insert XML comments so that you can see which elements could not be converted.

4. Adjust the resulting DITA composite to suit your needs. You may have to remove comments, fix validation errors, adjust certain attributes, or split the content into individual topics.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

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. Open the **Configure Transformation Scenario(s)** dialog box, select the **DocBook EPUB** transformation scenario, and click **Edit**.
4. In the **Parameters** tab, 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.

5. Run the transformation scenario.

DITA Topic Transformation Scenarios

The following default transformation scenarios are available for individual 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.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

DITA Map Transformation Scenarios

The following types of transformations scenarios are available for DITA maps:

- Predefined transformation scenarios allow you to transform a DITA map to a variety of outputs, such as 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 plugins from the DITA-OT/plugins directory.
- **DITA Map Metrics Report** - Use this type of transformation scenario if you want to generate a DITA map statistics report. They contain 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.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

DITA Map to WebHelp Output

DITA maps can be transformed into a variety of WebHelp systems designed to suit your specific needs. This section contains the procedures for obtaining the output for the variants of the WebHelp system.

Related information

[WebHelp System Output](#) on page 557

WebHelp Responsive Output

To publish a DITA map as a *WebHelp Responsive* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Responsive** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- [Templates Tab](#) - This tab contains a set of predefined skins that you can use for the layout of your WebHelp system output.
- [Parameters Tab](#) - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
- [Filters Tab](#) - This tab allows you to filter certain content elements from the generated output.
- [Advanced Tab](#) - This tab allows you to specify some advanced options for the transformation scenario.
- [Output Tab](#) - This tab allows you to configure options that are related to the location where the output is generated.

4. Click **Apply associated** to process the transformation.

When the **DITA Map WebHelp Responsive** transformation is complete, the output is automatically opened in your default browser.

Parameters for Customizing WebHelp Responsive Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

webhelp.reload.stylesheet

Set this parameter to `true` if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is `false`.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as `'*'` (matches zero or more characters) or `'?'` (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.show.changes.and.comments

When set to `yes`, user comments, replies to comments, and tracked changes are published in the WebHelp output. The default value is `no`.

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). For example, 0.3, 0.5, or 0.8. 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`.

Parameters Specific to WebHelp Responsive Output

webhelp.fragment.after.body

In the generated output it displays a given XHTML fragment after the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.logo_and_title

In the generated output it displays a given XHTML fragment after the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.main.page.search

In the generated output it displays a given XHTML fragment after the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.toc_or_tiles

In the generated output it displays a given XHTML fragment after the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.top_menu

In the generated output it displays a given XHTML fragment after the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.body

In the generated output it displays a given XHTML fragment before the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.logo_and_title

In the generated output it displays a given XHTML fragment before the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.main.page.search

In the generated output it displays a given XHTML fragment before the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.toc_or_tiles

In the generated output it displays a given XHTML fragment before the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.top_menu

In the generated output it displays a given XHTML fragment before the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.footer

In the generated output it displays a given XHTML fragment as the page footer. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.head

In the generated output it displays a given XHTML fragment as a page header. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.welcome

In the generated output it displays a given XHTML fragment as a welcome message (or title). The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.merge.nested.topics.related.links

Specifies if the related links from nested topics will be merged with the links in the parent topic. Thus the links will be moved from the topic content to the related links component and all of the links from the same group (for example, *Related Tasks*, *Related References*, *Related Information*) are merged into a single group. The default value is **yes**.

webhelp.show.breadcrumb

Specifies if the breadcrumb component will be presented in the output. The default value is **yes**.

webhelp.show.child.links

Specifies if child links will be generated in the output for all topics that have subtopics. The default value is **no**.

webhelp.show.indexterms.link

Specifies if an icon that links to the index will be presented in the output. The default value is **yes**.

webhelp.show.main.page.tiles

Specifies if the tiles component will be presented in the main page of the output. For a *tree* style layout, this parameter should be set to **no**.

webhelp.show.main.page.toc

Specifies if the table of contents will be presented in the main page of the output. The default value is **yes**.

webhelp.show.navigation.links

Specifies if navigation links will be presented in the output. The default value is **yes**.

webhelp.show.print.link

Specifies if a print link or icon will be presented within each topic in the output. The default value is **yes**.

webhelp.show.related.links

Specifies if the related links component will be presented in the WebHelp Responsive output. The default value is **yes**. The `webhelp.merge.nested.topics.related.links` parameter can be used in conjunction with this one to merge the related links from nested topics into the links in the parent topic.

webhelp.show.side.toc

Specifies if a side table of contents will be presented on the right side of each topic in the output. The default value is **yes**.

webhelp.show.top.menu

Specifies if a menu will be presented at the top of the main page in the output. The default value is **yes**.

webhelp.top.menu.depth

Specifies the maximum depth level of the topics that will be included in the top menu. The default value is 2.

Related information

[Customizing the WebHelp Responsive Output](#) on page 578

[WebHelp Responsive System](#) on page 558

WebHelp Responsive with Feedback Output

To publish a DITA map as a *WebHelp Responsive with Feedback* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Responsive with Feedback** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- [Templates Tab](#) - This tab contains a set of predefined skins that you can use for the layout of your WebHelp system output.
- [Parameters Tab](#) - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
- [Filters Tab](#) - This tab allows you to filter certain content elements from the generated output.
- [Advanced Tab](#) - This tab allows you to specify some advanced options for the transformation scenario.
- [Output Tab](#) - This tab allows you to configure options that are related to the location where the output is generated.

4. Click **Apply associated** to begin the transformation.
5. Enter the documentation product ID (value for the `webhelp.product.id` parameter) and the documentation version (value for the `webhelp.product.version` parameter).

When the **DITA Map WebHelp Responsive 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. Follow the [instructions to complete the system deployment](#).

Parameters for Customizing WebHelp Responsive with Feedback Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

webhelp.reload.stylesheet

Set this parameter to `true` if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is `false`.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as `'*'` (matches zero or more characters) or `'?'` (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.show.changes.and.comments

When set to `yes`, user comments, replies to comments, and tracked changes are published in the WebHelp output. The default value is `no`.

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). For example, 0.3, 0.5, or 0.8. 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`.

Parameters Specific to WebHelp Responsive with Feedback Output**webhelp.fragment.after.body**

In the generated output it displays a given XHTML fragment after the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.logo_and_title

In the generated output it displays a given XHTML fragment after the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.main.page.search

In the generated output it displays a given XHTML fragment after the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.toc_or_tiles

In the generated output it displays a given XHTML fragment after the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.after.top_menu

In the generated output it displays a given XHTML fragment after the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.body

In the generated output it displays a given XHTML fragment before the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.logo_and_title

In the generated output it displays a given XHTML fragment before the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.main.page.search

In the generated output it displays a given XHTML fragment before the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.toc_or_tiles

In the generated output it displays a given XHTML fragment before the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.before.top_menu

In the generated output it displays a given XHTML fragment before the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.footer

In the generated output it displays a given XHTML fragment as the page footer. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.head

In the generated output it displays a given XHTML fragment as a page header. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.fragment.welcome

In the generated output it displays a given XHTML fragment as a welcome message (or title). The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

webhelp.merge.nested.topics.related.links

Specifies if the related links from nested topics will be merged with the links in the parent topic. Thus the links will be moved from the topic content to the related links component and all of the links from the same group (for example, *Related Tasks*, *Related References*, *Related Information*) are merged into a single group. The default value is **yes**.

webhelp.show.breadcrumb

Specifies if the breadcrumb component will be presented in the output. The default value is **yes**.

webhelp.show.child.links

Specifies if child links will be generated in the output for all topics that have subtopics. The default value is **no**.

webhelp.show.indexterms.link

Specifies if an icon that links to the index will be presented in the output. The default value is **yes**.

webhelp.show.main.page.tiles

Specifies if the tiles component will be presented in the main page of the output. For a *tree* style layout, this parameter should be set to **no**.

webhelp.show.main.page.toc

Specifies if the table of contents will be presented in the main page of the output. The default value is **yes**.

webhelp.show.navigation.links

Specifies if navigation links will be presented in the output. The default value is **yes**.

webhelp.show.print.link

Specifies if a print link or icon will be presented within each topic in the output. The default value is **yes**.

webhelp.show.related.links

Specifies if the related links component will be presented in the WebHelp Responsive output. The default value is **yes**. The `webhelp.merge.nested.topics.related.links` parameter can be used in conjunction with this one to merge the related links from nested topics into the links in the parent topic.

webhelp.show.side.toc

Specifies if a side table of contents will be presented on the right side of each topic in the output. The default value is **yes**.

webhelp.show.top.menu

Specifies if a menu will be presented at the top of the main page in the output. The default value is **yes**.

webhelp.top.menu.depth

Specifies the maximum depth level of the topics that will be included in the top menu. The default value is 2.

Related information

[Customizing the WebHelp Responsive Output](#) on page 578

[WebHelp Responsive with Feedback System](#) on page 561

WebHelp Classic Output

To publish a DITA map as a *WebHelp Classic* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Classic** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- **Skins Tab** - This tab contains a set of predefined skins that you can use for the layout of your WebHelp system output.
- **Parameters Tab** - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
- **Filters Tab** - This tab allows you to filter certain content elements from the generated output.
- **Advanced Tab** - This tab allows you to specify some advanced options for the transformation scenario.
- **Output Tab** - This tab allows you to configure options that are related to the location where the output is generated.

4. Click **Apply associated** to process the transformation.

When the **DITA Map WebHelp Classic** transformation is complete, the output is automatically opened in your default browser.

To further customize this transformation, you can edit various parameters, including the following most commonly used ones:

Parameters for Customizing WebHelp Classic Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

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.

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.google.search.results

A file path 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>`

webhelp.google.search.script

A file path that specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google.

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.logo.image.target.url

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

webhelp.logo.image

Specifies a path to an image displayed as a logo in the left side of the output header.

webhelp.reload.stylesheet

Set this parameter to `true` if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is `false`.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as `'*'` (matches zero or more characters) or `'?'` (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.show.changes.and.comments

When set to `yes`, user comments, replies to comments, and tracked changes are published in the WebHelp output. The default value is `no`.

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). For example, 0.3, 0.5, or 0.8. 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`.

Related information

[Customizing WebHelp Classic Systems](#) on page 613

WebHelp Classic With Feedback Output

To publish a DITA map as a *WebHelp Classic with Feedback* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Classic with Feedback** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- **Skins Tab** - This tab contains a set of predefined skins that you can use for the layout of your WebHelp system output.
 - **Parameters Tab** - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
 - **Filters Tab** - This tab allows you to filter certain content elements from the generated output.
 - **Advanced Tab** - This tab allows you to specify some advanced options for the transformation scenario.
 - **Output Tab** - This tab allows you to configure options that are related to the location where the output is generated.
4. Click **Apply associated** to begin the transformation.
 5. Enter the documentation product ID (value for the `webhelp.product.id` parameter) and the documentation version (value for the `webhelp.product.version` parameter).

When the **DITA Map WebHelp Classic 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. Follow the [instructions to complete the system deployment](#).

To watch our video demonstration about the feedback-enabled WebHelp system, go to https://www.oxygenxml.com/demo/Feedback_Enabled_WebHelp.html.

Parameters for Customizing WebHelp Classic with Feedback Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

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.

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.google.search.results

A file path 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>`

webhelp.google.search.script

A file path that specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google.

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.logo.image.target.url

Specifies a target URL that is set on the logo image. When you click the logo image, you will be redirected to this address.

webhelp.logo.image

Specifies a path to an image displayed as a logo in the left side of the output header.

webhelp.reload.stylesheet

Set this parameter to **true** if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is **false**.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as `'*'` (matches zero or more characters) or `'?'` (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

webhelp.search.ranking

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.show.changes.and.comments

When set to `yes`, user comments, replies to comments, and tracked changes are published in the WebHelp output. The default value is `no`.

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). For example, 0.3, 0.5, or 0.8. 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`.

Related information

[Customizing WebHelp Classic Systems](#) on page 613

[WebHelp Classic with Feedback System](#) on page 605

WebHelp Classic Mobile Output

To publish a DITA map as a *WebHelp Classic Mobile* system, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Classic Mobile** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- [Parameters Tab](#) - This tab includes numerous parameters that can be set to customize your WebHelp system output. See the [Parameters section](#) below for details about the most commonly used parameters for WebHelp Responsive transformations.
- [Filters Tab](#) - This tab allows you to filter certain content elements from the generated output.
- [Advanced Tab](#) - This tab allows you to specify some advanced options for the transformation scenario.
- [Output Tab](#) - This tab allows you to configure options that are related to the location where the output is generated.

4. Click **Apply associated** to process the transformation.

When the **DITA Map WebHelp Classic Mobile** transformation is complete, the output is automatically opened in your default browser.

Parameters for Customizing WebHelp Classic Mobile Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

args.default.language

This parameter is used if the language is not detected in the DITA map. The default value is `en-us`.

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`).

fix.external.refs.com.oxygenxml (Only supported when the DITA OT transformation process is started from Oxygen XML Developer 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).

use.stemming

Controls whether or not you want to include stemming search algorithms into the published output (default setting is `false`).

webhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

webhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

webhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

webhelp.google.search.results

A file path 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>`

webhelp.google.search.script

A file path that specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google.

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.reload.stylesheet

Set this parameter to `true` if you have out of memory problems when generating WebHelp. It will increase processing time but decrease the memory footprint. The default value is `false`.

webhelp.search.custom.excludes.file

The path of the file that contains name patterns for HTML files that should not be indexed by the WebHelp search engine. Each exclude pattern must be on a new line. The patterns are considered to be relative to the output directory, and they accept wildcards such as '*' (matches zero or more characters) or '?' (matches one character). For more information about the patterns, see <https://ant.apache.org/manual/dirtasks.html#patterns>.

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

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). For example, 0.3, 0.5, or 0.8. 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`.

Related information

[Customizing WebHelp Classic Mobile Systems](#) on page 634

[WebHelp Classic Mobile System](#) on page 634

DITA Map to PDF Output Customization

Oxygen XML Developer plugin comes bundled with the DITA Open Toolkit that provides a mechanism for converting DITA maps to PDF output. There are numerous ways that you can customize the PDF output and the topics in this section discuss some of those possibilities.

Creating a DITA Map to PDF Transformation Scenario

To create a *DITA Map to PDF* transformation scenario, follow these steps:

1. Click the  **Configure Transformation Scenario(s)** button.
2. Select **DITA Map PDF** and click the **Edit** button (or use the **Duplicate** button if your framework is read-only).
3. Use the various tabs to configure the transformation scenario. In the **Parameters** tab, you can use a variety of parameters to customize the output. For example, the following parameters are just a few of the most commonly used ones:
 - `show.changes.and.comments` - If set to `yes`, user comments, replies to comments, and tracked changes are published in the PDF output.
 - `customization.dir` - Specifies the path to a customization directory.
4. Click **OK** and then the **Apply Associated** button to run the transformation scenario.

Creating a Customization Directory for PDF Output

DITA Open Toolkit PDF output can be customized in several ways:

- Creating a DITA Open Toolkit plugin that adds extensions to the PDF plugin. More details can be found in the [DITA Open Toolkit Documentation](#).
- Creating a customization directory and using it from the PDF transformation scenario. A small example of this procedure can be found below.

How to Create a Customization Directory for PDF Output

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 Developer plugin and *uncomment* this line:

```
<!-- uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"/-->
```

It now 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 Developer plugin and create the template called createFrontCoverContents for DITA-OT 2.3.3 (or createFrontMatter_1.0 for DITA-OT 1.8.5).



Tip: You can copy the same template from *DITA_OT_DIR*\plugins\org.dita.pdf2\xsl\fo\front-matter.xsl and modify it in whatever way necessary to achieve your specific goal. This new template in the custom.xsl file will override the same template from *DITA_OT_DIR*\plugins\org.dita.pdf2\xsl\fo\front-matter.xsl.

DITA OT 2.3.3 Example:

For example, if you are using DITA OT 2.3.3, the custom.xsl could look like this:

```
<?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="2.0">

  <xsl:template name="createFrontCoverContents">
    <!-- 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>

    <!-- Load the image logo -->
    <fo:block text-align="center" width="100%">
      <fo:external-graphic
        src="url({concat($artworkPrefix,
          '/Customization/OpenTopic/common/artwork/logo.png')})"
      />
    </fo:block>
  </xsl:template>
</xsl:stylesheet>
```

DITA OT 1.8.5 Example:

For example, if you are using DITA OT 1.8.5, the `custom.xsl` could look like this:

```
<?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>
      </fo:flow>
    </fo:page-sequence>
    <xsl:call-template name="createNotices"/>
  </xsl:template>
</xsl:stylesheet>
```

7. Edit the **DITA Map PDF** transformation scenario and in the **Parameters** tab, set the `customization.dir` parameter to `C:\Customization`.

Related information

[Automatic PDF plugin customization generator by Jarno Elovirta](#).

[DITA OT Documentation - PDF Customization Plugin](#)

Customizing the Header and Footer in PDF Output

The XSLT stylesheet `DITA_OT_DIR/plugins/org.dita.pdf2/xsl/fo/static-content.xsl` contains templates that output the static header and footers for various parts of the PDF such as 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 that 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 (such as *Body odd footer*) that 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](#).

Adding a Watermark to PDF Output

To add a watermark to the PDF output of a DITA map transformation, create a DITA-OT customization directory and use it from a DITA Map to PDF transformation scenario, as in the following procedure:

1. Copy the entire directory: `DITA_OT_DIR\plugins\org.dita.pdf2\Customization` to another location (for instance, `C:\Customization`).
2. Copy your watermark image (for example, `watermark.png`) to:
`C:\Customization\common\artwork\watermark.png`.
3. Rename the `C:\Customization\catalog.xml.orig` file to: `C:\Customization\catalog.xml`.
4. Open the `catalog.xml` in Oxygen XML Developer plugin and *uncomment* this line:

```
<!-- uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"/-->
```

The uncommented line should look 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 `C:\Customization\fo\xsl\custom.xsl` file in Oxygen XML Developer plugin to overwrite two XSLT templates:
 - The first template is located in the XSLT stylesheet `DITA_OT_DIR\plugins\org.dita.pdf2\xsl\fo\static-content.xsl` and we override it specifying a watermark image for every page in the PDF content, using a `block-container` element that references the watermark image file:

```
<fo:static-content flow-name="odd-body-header">
  <fo:block-container absolute-position="absolute"
    top="-2cm" left="-3cm" width="21cm" height="29.7cm"
    background-image="{concat($artworkPrefix,
'/Customization/OpenTopic/common/artwork/watermark.png')}">
    <fo:block/>
  </fo:block-container>
  <fo:block xsl:use-attribute-sets="__body_odd_header">
    <xsl:call-template name="insertVariable">
      <xsl:with-param name="theVariableID" select="'Body odd header'"/>
      <xsl:with-param name="theParameters">
        <prodname>
          <xsl:value-of select="$productName"/>
        </prodname>
        <heading>
          <fo:inline xsl:use-attribute-sets="__body_odd_header_heading">
            <fo:retrieve-marker retrieve-class-name="current-header"/>
          </fo:inline>
        </heading>
        <pagenum>
          <fo:inline xsl:use-attribute-sets="__body_odd_header_pagenum">
            <fo:page-number/>
          </fo:inline>
        </pagenum>
      </xsl:with-param>
    </xsl:call-template>
  </fo:block>
</fo:static-content>
</xsl:template>
```

- The second template that we override is located in the XSLT stylesheet `DITA_OT_DIR\plugins\org.dita.pdf2\xsl\fo\commons.xsl` and is used for styling the first page of the output. We also override it by copying the original template content in our `custom.xsl` and adding the `block-container` element that references the watermark image file:

```
<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-container absolute-position="absolute"
        top="-2cm" left="-3cm" width="21cm" height="29.7cm"
        background-image="{concat($artworkPrefix,
'/Customization/OpenTopic/common/artwork/watermark.png')}">
        <fo:block/>
      </fo:block-container>
      <fo:block xsl:use-attribute-sets="__frontmatter">
        <!-- set the title -->
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

```

<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>

<!--<xsl:call-template name="createPreface"/>-->

</fo:flow>
</fo:page-sequence>
<xsl:if test="not($retain-bookmap-order)">
  <xsl:call-template name="createNotices" />
</xsl:if>
</xsl:template>

```

7. Edit your **DITA Map PDF** transformation scenario. In the **Parameters** tab, set the customization.dir parameter to C:\Customization.

Force Page Breaks Between Two Block Elements in PDF Output

Suppose that 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 (for example, **DITA_OT_DIR/plugins/pdf-page-break**).
3. In this new folder, create a new **plugin.xml** file with the following 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:template>
</xsl:stylesheet>
```

Customizing note Images in PDF

To customize the images that appear next to each type of note in the PDF output, use a [PDF customization folder](#) with the following procedure:

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 Map PDF** transformation scenario and in the **Parameters** tab set the path for the `customization.dir` property to point to the customization folder.

Show Comments and Tracked Changes in PDF Output

To make comments and tracked changes (stored within your DITA topics) appear in the PDF output, follow these steps:

1. Click the  **Configure Transformation Scenario(s)** button.
2. Select **DITA Map PDF** and click the **Edit** button (or use the **Duplicate** button if your framework is read-only).
3. In the **Parameters** tab, set the value of the `show.changes.and.comments` parameter to **yes**.
4. Click **OK** and then the **Apply Associated** button to run the transformation scenario.

Result: Comment threads and tracked changes will now appear in the PDF output. Details about each comment or change will be available in the footer section for each page.

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 [Add a Font to the Built-in FO Processor](#) on page 554.

DITA OT PDF Font Mapping

The DITA OT contains a file `DITA_OT_DIR/plugins/org.dita.pdf2/cfg/fo/font-mappings.xml` that maps logical fonts used in the XSLT stylesheets to physical fonts that 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 this:

```
<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. Therefore, 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**.

Related information

<http://www.elovirta.com/2016/02/18/font-configuration-in-pdf2.html>

DITA Map to PDF WYSIWYG Transformation

Oxygen XML Developer plugin comes bundled with a DITA OT plugin that converts DITA maps to PDF using a CSS layout processor. Oxygen XML Developer plugin supports the following processors (not included in the Oxygen XML Developer 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 Developer 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.
2. Select **DITA Map PDF - WYSIWYG - Experimental**.
3. 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**, user comments, replies to comments, and tracked changes are published in the PDF output. The default value is **no**.
 - `dita.css.list` - Allows you to specify a list of CSS URLs to be used by the PDF processor. The files must have URL syntax and be separated using semicolons.
 - `args.css` - Allows you to specify a list of CSS URLs to be used in addition to those specified in the `dita.css.list` parameter.
4. Click **OK** and run the transformation scenario.

Customizing the Styles (for Output and Editing)

If you need to change the styles in the associated CSS, make sure you install Oxygen XML Developer plugin in a folder where you have full read and write privileges (for instance, your user home directory). This is due to the fact that the installed files are usually placed in a read-only folder (for instance, in Windows, Oxygen XML Developer plugin is installed in the **Program Files** folder where the users do not have change rights).

To change the styles of an element you need to create an additional CSS file that will store the customization rules. Once you have created this file, you need to instruct the editor how to use this additional CSS.

- Use the `args.css` parameter that allows you to specify a list of CSS URLs to be used in addition to the `dita.css.list` parameter:
 1. Configure a *DITA map to PDF WYSIWYG* transformation scenario, as described in the procedure [above](#).
 2. In the **Parameters** tab, specify the path to your custom CSS files in the `args.css` parameter.
 3. Click **OK** and run the transformation scenario.

This method is appropriate if you just want to apply the styling customization to the output.

How to Make Remote Resources Appear in the Output When Using the Prince Processor

If your documentation references external resources (such as images that are stored on a Web-based repository) and your system is behind an HTTP(S) proxy, you may find that they do not appear in the output when using the **Prince Print with CSS** processor. To solve this, follow this procedure:

1. Edit the `build.xml` file that is located in `DITA_OT_DIR/plugins/com.oxygenxml.pdf.css`.
2. There are two instances in the file where a pair of arguments are commented out:

```
<!-- Please remove the comment wrapping the following two arguments
if you are behind a proxy and your documentation refers remote resources,
for example images.

Note: You also need to remove the backslash '\' that appears before the
property name: "http-proxy". That backslash is there because a sequence of
two dashes ('-') is not permitted inside a comment.
-->
<!--
  <arg value="--\-http-proxy=${http.proxyHost}:${http.proxyPort}"/>
  <arg value="--\-http-proxy=${https.proxyHost}:${https.proxyPort}"/>
-->
```

3. Remove the comment in both instances.
4. Make sure you also remove the slash that appears before the property name `http-proxy` in each instance.

Step Result: The arguments should now look like this:

```
<arg value="--http-proxy=${http.proxyHost}:${http.proxyPort}"/>
<arg value="--http-proxy=${https.proxyHost}:${https.proxyPort}"/>
```

5. Save the `build.xml` file.
6. Run the *DITA map to PDF WYSIWYG transformation scenario*.

Result: Your external resources should now appear in your output.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

Compiled HTML Help (CHM) Output Format

To perform a **Compiled HTML Help (CHM)** transformation Oxygen XML Developer plugin needs Microsoft HTML Help Workshop to be installed on your computer. Oxygen XML Developer plugin automatically detects HTML Help Workshop and uses it.



Note: HTML Help Workshop might fail if the files used for transformation contain accents in their names, due to different encodings used when writing the `.hhp` and `.hhc` files. If the transformation fails to produce the CHM output but the `.hhp` (HTML Help Project) file is already generated, you can manually try to build the CHM output using HTML Help Workshop.

Changing the Output Encoding

Oxygen XML Developer plugin uses the `htmlhelp.locale` parameter to correctly display specific characters of different languages in the output of the **Compiled HTML Help (CHM)** transformation. The **Compiled HTML Help (CHM)** default scenario that comes bundled with Oxygen XML Developer plugin has the `htmlhelp.locale` parameter set to `en-US`.

The default value of the `htmlhelp.locale` is `en-US`. To customize this parameter, go to  **Configure Transformation Scenarios** and click the  **Edit** button. In the parameter tab search for the `htmlhelp.locale` parameter and change its value to the desired language tag.

The format of the `htmlhelp.locale` parameter is `LL-CC`, where `LL` represents the language code (`en`, for example) and `CC` represents the country code (`US`, for example). The language codes are contained in the ISO 639-1 standard and the country codes are contained in the ISO 3166-1 standard. For further details about language tags, go to <http://www.rfc-editor.org/rfc/rfc5646.txt>.

Kindle Output Format

Oxygen XML Developer plugin requires KindleGen to generate Kindle output from DITA maps. To install KindleGen for use by Oxygen XML Developer plugin, follow these steps:

1. Go to www.amazon.com/kindleformat/kindlegen and download the zip file that matches your operating system.
2. Unzip the file on your local disk.
3. Start Oxygen XML Developer plugin and open a DITA map in the **DITA Maps Manager** view.
4. In the **DITA Maps Manager** view, open the  **Configure Transformation Scenario(s)** dialog box.
5. Select the **DITA Map Kindle** transformation and press the **Edit** button to edit it.
6. Go to **Parameters** tab and set the `kindlegen.executable` parameter as the path to the *KindleGen* directory.
7. Accept the changes.

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.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

TEI ODD Transformation Scenarios

The following default transformations are available for TEI ODD document types:

- **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

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

TEI P4 Transformation Scenarios

The following default transformations are available for TEI P4 documents:

- **TEI HTML** - Transforms a TEI document into an HTML document.
- **TEI P4 - TEI P5 Conversion** - Convert a TEI P4 document into a TEI P5 document.
- **TEI PDF** - Transforms a TEI document into a PDF document using the Apache FOP engine.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

TEI P5 Transformation Scenarios

The following default transformations are available for TEI P5 documents:

- **TEI P5 XHTML** - Transforms a TEI P5 document into an XHTML document.
- **TEI P5 PDF** - Transforms a TEI P5 document into a PDF document using the Apache FOP engine.
- **TEI EPUB** - Transforms a TEI P5 document into an EPUB output. The EPUB output will contain any images referenced in the TEI XML document.
- **TEI DOCX** - Transforms a TEI P5 document into a DOCX (OOXML) document. The DOCX document will contain any images referenced in the TEI XML document.
- **TEI ODT** - Transforms a TEI P5 document into an ODT (ODF) document. The ODT document will contain any images referenced in the TEI XML document.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

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.

Related information

[Editing a Transformation Scenario](#) on page 543

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

Creating New Transformation Scenarios

Defining a transformation scenario is the first step in the process of transforming a document. This section includes information on the types of scenarios that are available in Oxygen XML Developer plugin and how to create each type of transformation.

XML Transformation with XSLT

This type of transformation 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.

To create an **XML transformation with XSLT** scenario, use one of the following methods:

- Use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu. Then click the **New** button and select **XML transformation with XSLT**.
- Use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the 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 to see if transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **XML transformation with XSLT**.

All three methods open the **New Scenario** dialog box.

The upper part of the dialog box allows you to specify the **Name** of the transformation scenario.

The lower part of the dialog box contains several tabs that allow you to configure the options that control the transformation.

XSLT Tab

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **XSLT** tab contains the following options:

XML URL

Specifies the source XML file. You can specify the path by using the text field, its history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list. 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.x and a custom URI resolver is configured in the [advanced Saxon preferences page](#), the XML input of the transformation is passed to that URI resolver. 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 this field.

XSL URL

Specifies the source XSL file that the transformation will use. You can specify the path by using the text field, its history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list. 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.

Use "xml-stylesheet" declaration

If enabled, the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction. By default, this option is disabled and the transformation applies the XSLT stylesheet that is specified in the **XSL URL** field.

Transformer

This drop-down menu presents all the transformation engines available to Oxygen XML Developer plugin for performing a transformation. These include the built-in engines and [the external engines defined in the Custom Engines preferences page](#). The engine you choose 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 that preferences page.

Parameters

Opens a [Configure parameters dialog box](#) that allows you to configure the XSLT parameters used in the current transformation. In this dialog box, you can also configure the parameters for [additional XSLT stylesheets](#). 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 a [dialog box for configuring the XSLT extension jars or classes](#) that define extension Java functions or extension XSLT elements used in the transformation.

Additional XSLT stylesheets

Opens a [dialog box for adding XSLT stylesheets](#) that are applied on the main stylesheet specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document.

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 resulting dialog box includes a table that 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, and you can use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with their 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.

Below the table, 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.

Related information

[Editor Variables](#) on page 99

XSLT Extensions

The **Extensions** button is used to specify the jars and classes that contain extension functions called from the XSLT 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 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.

Additional XSLT Stylesheets

Use the **Additional XSLT Stylesheets** button in the **XSLT** tab to display a list of additional XSLT stylesheets to be used in the transformation and you can add files to the list or edit existing entries. 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.

Advanced Saxon HE/PE/EE XSLT Transformation Options

The XSLT 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 [those in the Saxon HE/PE/EE preferences page](#) but they are configured as a specific set of transformation options for each transformation scenario, while the values set in the preferences page apply as global options. The advanced options configured in a transformation scenario override the global options defined in the preferences page.

The advanced options for Saxon 9.6.0.7 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

Mode ("-im")

A Saxon-specific option that sets the initial mode for the transformation.

Template ("-it")

A Saxon-specific option that sets the name of the initial XSLT template to be executed.

Use a configuration file ("-config")

Sets a Saxon 9.6.0.7 configuration file that is executed for XSLT transformation and validation processes.

Debugger trace into XPath expressions (applies to debugging sessions)

Instructs the [XSLT Debugger](#) to *step into* XPath expressions.

Version warnings ("-versmsg")

Warns you when the transformation is applied to an XSLT 1.0 stylesheet.

Line numbering ("-l")

Line numbers where errors occur are included in the output messages.

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")

Specifies whether or not the source document will be validated against their associated DTD. You can choose from the following:

- **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")

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")

Allows you to choose how the *strip whitespaces* operation is handled. You can choose 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")

Allows you to set the optimization level. It is the value is an integer in the range of 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.

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.

The following advanced options are specific for Saxon 9.6.0.7 Professional Edition (PE) and Enterprise Edition (EE) only:

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 Developer plugin only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about [executing such a stylesheet on Saxonica's website](#).

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 `http://[URL]`). It ensures that the stylesheet cannot call arbitrary Java methods and thus gain privileged access to resources on your machine.

The advanced options that are specific for Saxon 9.6.0.7 Enterprise Edition (EE) are as follows:

XML Schema version

Use this option to change the default XML Schema version for this transformation. To change the default XML Schema version globally, [open the Preferences dialog box](#) and go to **XML > XML Parser > XML Schema** and use the [Default XML Schema version option](#).

Validation of the source file ("-val")

Requests schema-based validation of the source file and of any files read using `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 result 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 for non-fatal errors are written (wherever 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>.

FO Processor Tab (XSLT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **FO Processor** tab contains the following options:

Perform FO Processing

Specifies whether or not an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.

Input

Choose between the following options to specify which input file to use:

- **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. The available options depend on the selected processor type.

Processor

Specifies the FO processor to be used for the transformation. It can be the built-in Apache FOP processor or an [external processor](#).

Output Tab (XSLT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

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. You can specify the path by using the text field, the  [Insert Editor Variables](#) button, or the  **Browse** button.

Open in Browser/System Application

If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, in Windows *PDF* files are often opened in *Acrobat Reader*).



Note: To set the web browser that is used for displaying HTML/XHTML pages, go to **Window > Preferences > General > Web Browser** and specify it there.

- **Output file** - When **Open in Browser/System Application** is selected, you can use this button to automatically open the default output file at the end of the transformation.
- **Other location** - When **Open in Browser/System Application** is selected, you can use this option to open the file specified in this field at the end of the transformation. You can specify the path by using the text field, the  [Insert Editor Variables](#) button, or the  **Browse** button.

Open in editor

When this is enabled, at the end of the transformation, the default output file 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

You can choose to view the results in one of the following:

- **XML** - If this is selected, Oxygen XML Developer plugin displays the transformation result in an XML viewer panel at the bottom of the application window with [syntax highlighting](#).
- **XHTML** - This option can only be selected if **Open in Browser/System Application** is disabled. If selected, Oxygen XML Developer plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.



Important: When transforming very large documents, you should be aware that enabling this feature may result in very long processing times. 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 selecting the **Open in Browser/System Application** option instead.

- **Image URLs are relative to** - If **Show in results view as XHTML** is selected, this option specifies the path used to resolve image paths contained in the transformation result. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.

Oxygen XML Developer plugin Browser View

The Oxygen XML Developer 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.

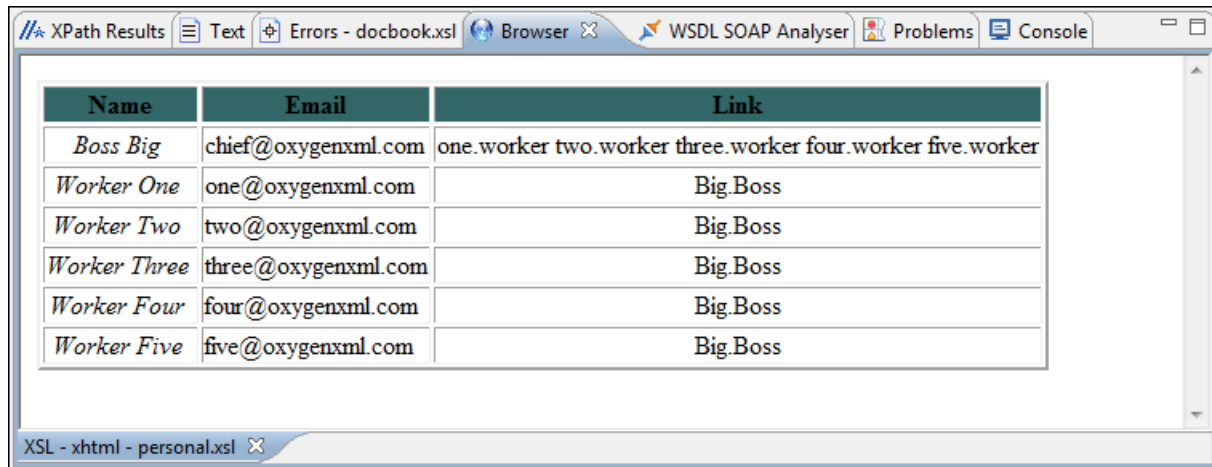


Figure 277: Browser View

Oxygen XML Developer plugin Text View

The Oxygen XML Developer 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.

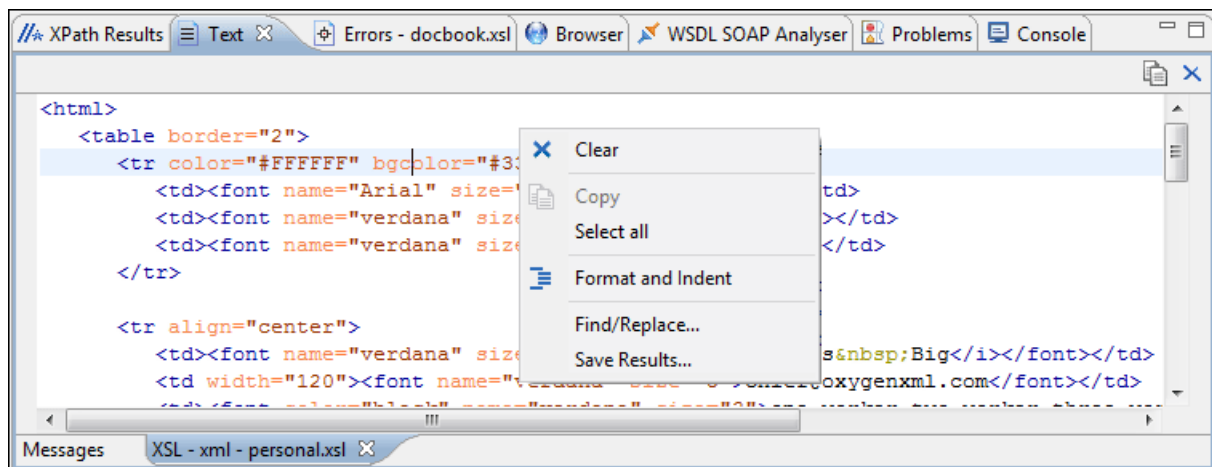


Figure 278: Text View

Text View Contextual Menu Actions

The following actions are available when the contextual menu is invoked in this view:

- ✕ **Clear**
Removes all content from the view.
-  **Copy**
Copies the selected content to the clipboard.

Select All

Selects all content in the view.



Format and Indent

Formats (pretty prints) the content in this view.

Find/Replace

Opens the [Find/Replace dialog box](#) that allows you to perform search and replace operations.

Save Results

Saves the content in the view to a file in text format.

XML Transformation with XQuery

This type of transformation specifies the transform parameters and location of an XQuery file that is applied to the edited XML document.

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:

- Use the **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu. Then click the **New** button and select **XML transformation with XQuery**.
- Use the **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the 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 to see if transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XML transformation with XQuery**.

All three methods open the **New Scenario** dialog box.

The upper part of the dialog box allows you to specify the **Name** of the transformation scenario.

The lower part of the dialog box contains several tabs that allow you to configure the options that control the transformation.

XQuery Tab

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **XQuery** tab contains the following options:

XML URL

Specifies the source XML file. You can specify the path by using the text field, its history drop-down, the **Insert Editor Variables** button, or the browsing tools in the ▾ **Browse** drop-down list. 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.x and a custom URI resolver is configured in the [advanced Saxon preferences page](#), the XML input of the transformation is passed to that URI resolver.

XQuery URL

Specifies the source XQuery file to be used for the transformation. You can specify the path by using the text field, its history drop-down, the **Insert Editor Variables** button, or the browsing tools in the ▾ **Browse** drop-down list. 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.

Transformer

This drop-down menu presents all the transformation engines available to Oxygen XML Developer plugin for performing a transformation. These include the built-in engines and [the external engines defined in the Custom Engines preferences](#)

[page](#). The engine you choose 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 that preferences page.

Parameters

Opens the [Configure parameters dialog box](#) for configuring the XQuery parameters. You can use the buttons in this dialog box to 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.

Extensions

Opens a [dialog box for configuring the XQuery extension jars or classes](#) that define extension Java functions or extension XSLT elements used in the transformation.

XQuery Parameters

The global parameters of the XQuery file used in a transformation scenario can be configured by using the **Parameters** button in the **XQuery** tab.

The resulting dialog box includes a table that displays all the parameters of the current XQuery file, along with their descriptions and current values. You can also add, edit, and remove parameters, and use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with their name in bold.

If the **XPath** column is checked, the parameter value is evaluated as an XPath expression before starting the XQuery 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 XQuery file location. You can use full paths or editor variables (such as `${cfdu}` [current file directory]) to specify other locations: `doc('${cfdu}/test.xml')//*`
2. Only XPath functions are allowed.

Below the table, 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.

Related information

[Editor Variables](#) on page 99

XQuery Extensions

The **Extensions** button is used to specify the jars and classes that contain extension functions called from the 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 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.

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 [those in the Saxon HE/PE/EE preferences page](#) but they are configured as a specific set of transformation options for each transformation scenario, while the values set in the preferences page apply as global options. The advanced options configured in a transformation scenario override the global options defined in the preferences page.

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")

Allows you to choose how the *strip whitespaces* operation is handled. You can choose 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")

Allows you to set the optimization level. It is the value is an integer in the range of 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 enabled (default value), Saxon runs the XQuery transformation with the XQuery 3.0 support.

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.

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 9.6.0.7 configuration file that is used for XQuery transformation and validation scenarios.

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 `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 result 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 for non-fatal errors are written (wherever 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>.

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 an XQuery Update are generated. This option is available when the **Enable XQuery update** option is enabled.

FO Processor Tab (XQuery Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **FO Processor** tab contains the following options:

Perform FO Processing

Specifies whether or not an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.

Input

Choose between the following options to specify which input file to use:

- **XQuery result as input** - The FO processor is applied to the result of the XQuery transformation that is 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. The available options depend on the selected processor type.

Processor

Specifies the FO processor to be used for the transformation. It can be the built-in Apache FOP processor or an [external processor](#).

Output Tab (XQuery Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Output** tab contains the following options:

Present as a sequence

Enabling this option will reduce the time necessary to fetch the full results, as it will only fetch the first chunk of the results.

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. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.

Open in Browser/System Application

If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, in Windows *PDF* files are often opened in *Acrobat Reader*).



Note: To set the web browser that is used for displaying HTML/XHTML pages, go to **Window > Preferences > General > Web Browser** and specify it there.

- **Output file** - When **Open in Browser/System Application** is selected, you can use this button to automatically open the default output file at the end of the transformation.
- **Other location** - When **Open in Browser/System Application** is selected, you can use this option to open the file specified in this field at the end of the transformation. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.

Open in editor

When this is enabled, at the end of the transformation, the default output file 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

You can choose to view the results in one of the following:

- **XML** - If this is selected, Oxygen XML Developer plugin displays the transformation result in an XML viewer panel at the bottom of the application window with [syntax highlighting](#).
- **XHTML** - This option can only be selected if **Open in Browser/System Application** is disabled. If selected, Oxygen XML Developer plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.



Important: When transforming very large documents, you should be aware that enabling this feature may result in very long processing times. 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 selecting the **Open in Browser/System Application** option instead.

- **Image URLs are relative to** - If **Show in results view as XHTML** is selected, this option specifies the path used to resolve image paths contained in the transformation result. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.

DITA OT Transformation

This type of transformation specifies the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Developer plugin includes a built-in version of Ant and a built-in version of DITA-OT, but other versions can be set in the scenario.

To create a **DITA OT Transformation** scenario, use one of the following methods:

- Use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu. Then click the **New** button and select **DITA OT Transformation**.
- Use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the 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 to see if transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **DITA OT Transformation**.

All three methods open the **DITA Transformation Type** dialog box that presents the list of possible outputs.

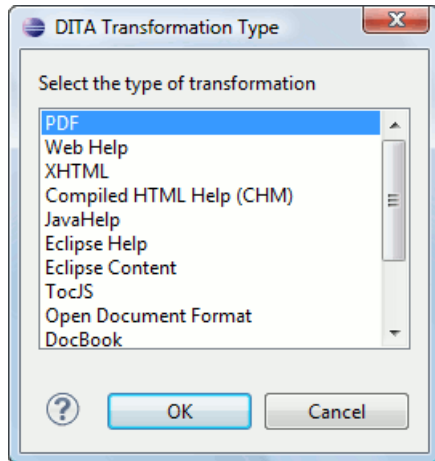


Figure 279: DITA Transformation Type Dialog Box

Select the desired type of output and click **OK**. This opens the **New Scenario** dialog box.

The upper part of the dialog box allows you to specify the **Name** of the transformation scenario.

The lower part of the dialog box contains several tabs that allow you to configure the options that control the transformation. Some of these tabs are only available for certain output types (for example, a **Skins** tab is only available for **WebHelp Classic** and **WebHelp Classic with Feedback** output types, a **Templates** tab is available only for **WebHelp Responsive** and **WebHelp Responsive with Feedback**, and a **FO Processor** tab is available for PDF output).

Skins Tab (DITA OT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Skins** tab is available for DITA OT transformations with **WebHelp Classic** or **WebHelp Classic with Feedback** output types and it provides a set of predefined skins that you can use as a base for your WebHelp system output.

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 your output.

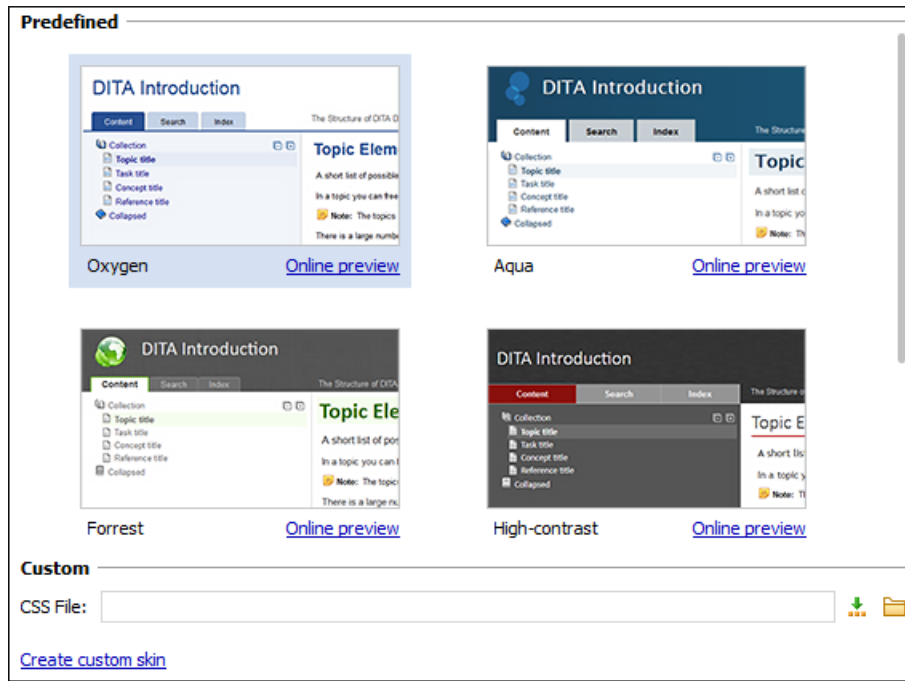


Figure 280: Skins Tab

The **Skins** tab includes the following sections:

Predefined Skins

This section presents the predefined skins that are included in Oxygen XML Developer plugin. The predefined skins cover a wide range of chromatic themes, ranging from a very light one to a high-contrast variant. To see how the *skin* looks when applied on a sample documentation project that is stored on the Oxygen XML Developer plugin website, press the **Online preview** link.

Custom Skins

You can use this section to customize the look of the output.

CSS File

You can set this field to point to a custom CSS stylesheet or customized skin. 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.

Create custom skin

Use this link to open the [WebHelp Skin Builder](#) tool.

Templates Tab (DITA OT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Templates** tab is available for DITA OT transformations with **WebHelp Responsive** or **WebHelp Responsive with Feedback** output types and it provides a set of predefined *skins* that you can use as a base for the layout of your WebHelp system output.

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 your output. You can choose predefined skins in a *tile* style of layout or a *tree* style of layout, and you can also [add your own customized skins](#).

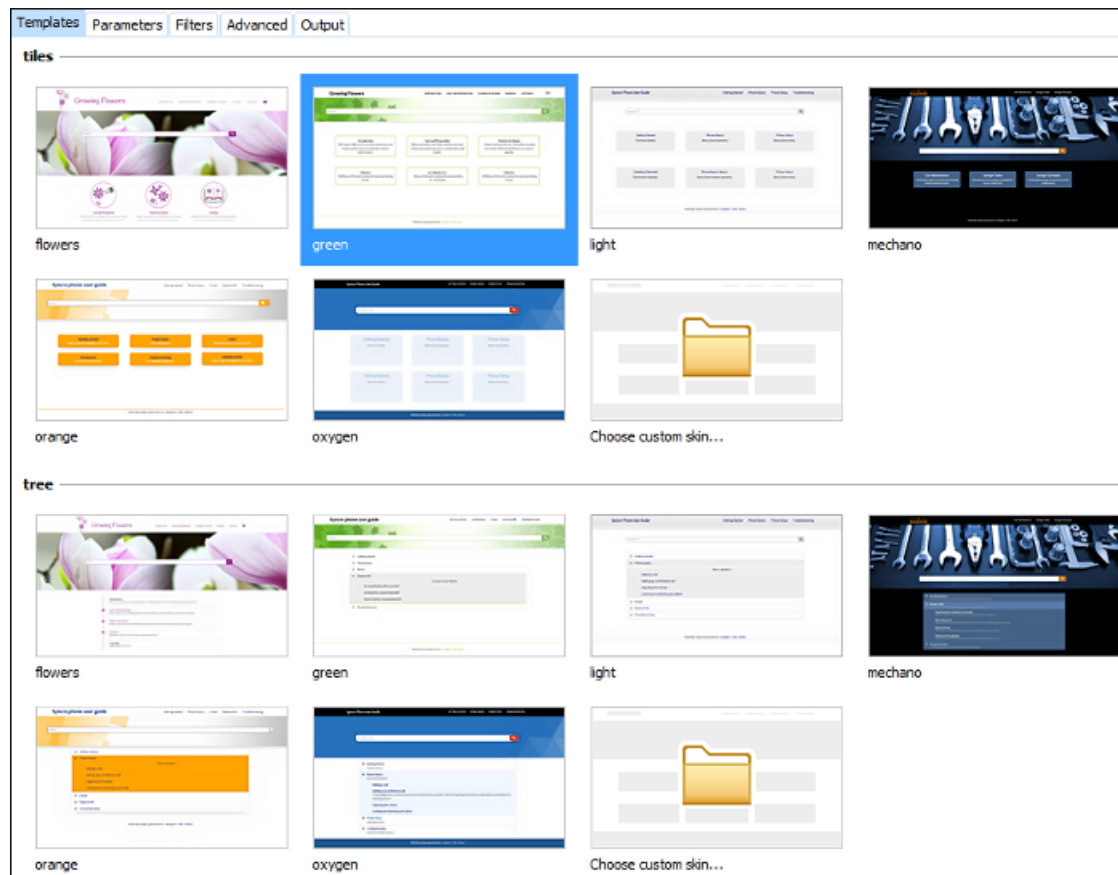


Figure 281: Templates Tab

The **Templates** tab comes by default with the following predefined collections of skins:

Tiles

This section presents the predefined skins that are arranged in a *tiles* style of layout. These predefined skins include a variety of themes, ranging from a very light one to a high-contrast variant, and various styles. If you select **Choose custom skin**, you can select a custom CSS stylesheet to be used as your template.

Tree

This section presents the predefined skins that are arranged in a *tree* style of layout. These predefined skins include a variety of themes, ranging from a very light one to a high-contrast variant, and various styles. If you select **Choose custom skin**, you can select a custom CSS stylesheet to be used as your template.

When you add a new collection of skins, this tab will list them.

FO Processor Tab (DITA OT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **FO Processor** tab is available for DITA OT transformations with a **PDF** output type.

This tab allows you to select an FO Processor to be used for the transformation.

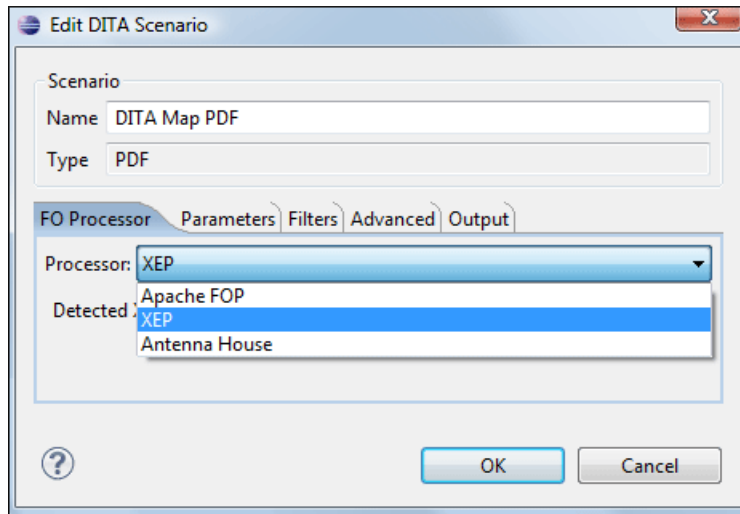


Figure 282: FO Processor Configuration Tab

You can choose one of the following processors:

Apache FOP

The default processor that comes bundled with Oxygen XML Developer plugin.

XEP

The [RenderX XEP](#) processor. If XEP is already installed, Oxygen XML Developer plugin 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 `DITA_OT_DIR/plugins/org.dita.pdf2/lib` directory of the Oxygen XML Developer plugin installation directory.

Antenna House

The [Antenna House](#) (AH Formatter) processor. If Antenna House is already installed, Oxygen XML Developer plugin 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 Oxygen XML Developer plugin preferences pages.

To further customize the PDF output obtained from the Antenna House processor, follow these steps:

1. **Edit** the transformation scenario.
2. Open the [Parameters tab](#).
3. Add the `env.AXF_OPT` parameter and point to the Antenna House configuration file.

Related information

[FO Processors Preferences](#) on page 85

[XSL-FO Processors](#) on page 554

Parameters Tab (DITA OT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Parameters** tab allows you to configure the parameters sent to the DITA-OT build file.

The table in this tab 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, and you can use the text

box to filter or search for a specific term in the entire parameters collection. Note that edited parameters are displayed with their 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.

Below the table, 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. You can specify the **Value** of the parameter by using the  [Insert Editor Variables](#) button or the  **Browse** button.

Unset

Resets the selected parameter to its default value. Available only for edited parameters with set values.

Edit

Opens the **Edit Parameter** dialog box that allows you to change the value of the selected parameter or its description.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

Related information

[DITA Open Toolkit Documentation](#)

Filters Tab (DITA Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Filters** tab allows you to add filters to remove certain content elements from the generated output.

You can choose one of the following options 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. You can specify the path by using the text field, its history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list. You can find out more about constructing a *DITaval* file in the [DITA Documentation](#).

 **Attention:** If a filter file is specified in the `args.filter` parameter (in [the Parameters tab](#)), that file takes precedence over a *DITaval* file specified here.

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.

Advanced Tab (DITA OT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Advanced** tab allows you to specify advanced options for the transformation scenario.

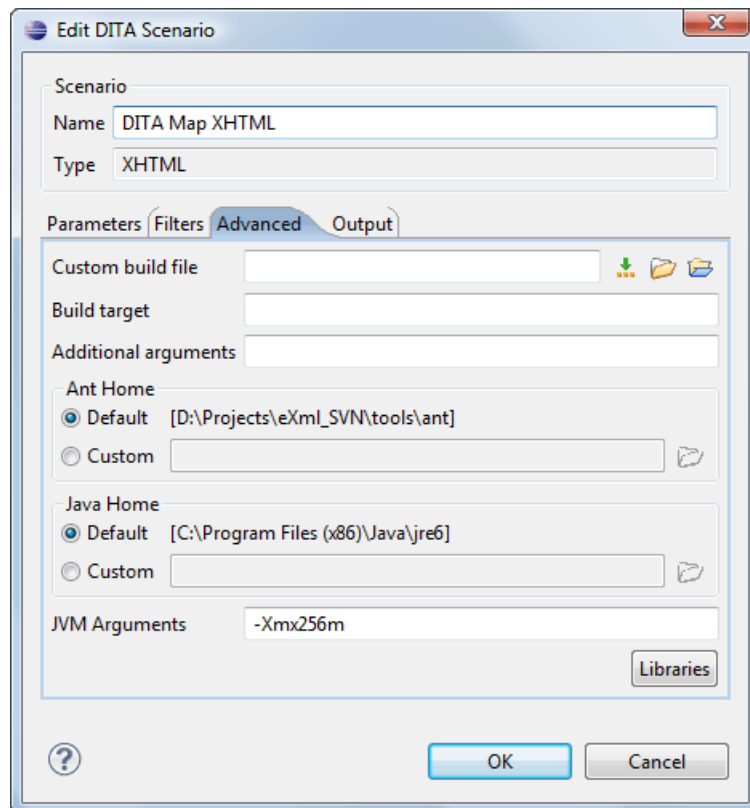


Figure 283: Advanced Settings Tab

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. You can specify the path by using the text field, the [Insert Editor Variables](#) button, or the **Browse** 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 transformation (such as `-verbose`).

Ant Home

You can choose between the default or custom Ant installation to run the transformation.

Java Home

You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by Oxygen XML Developer plugin.



Note: It may be possible that the used Java version is incompatible with the DITA Open Toolkit engine. For example, DITA OT 1.8 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 transformation, 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 Developer plugin adds libraries (as high priority) 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.

Output Tab (DITA OT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Output** tab allows you to configure options that are related to the location where the output is generated.

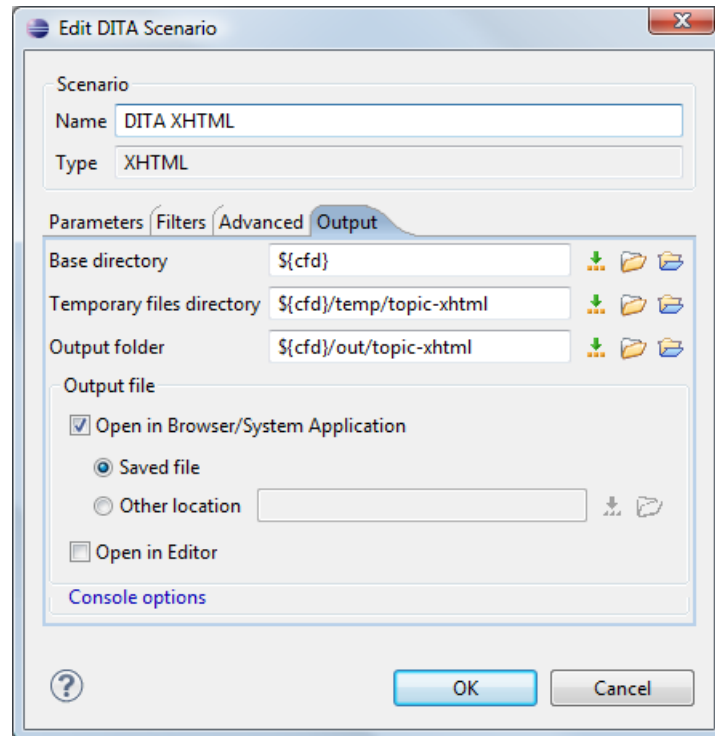


Figure 284: Output Settings Tab

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. You can specify the path by using the text field, the [Insert Editor Variables](#) button, or the **Browse** button.

Temporary files directory

This directory is used to store pre-processed temporary files until the final output is obtained. You can specify the path by using the text field, the [Insert Editor Variables](#) button, or the **Browse** button.

Output directory

The folder where the content of the final output is stored. You can specify the path by using the text field, the [Insert Editor Variables](#) button, or the **Browse** button.

Note: If the DITA map or topic is opened from a remote location or a ZIP file, the parameters must specify absolute paths.

Open in Browser/System Application

If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, in Windows *PDF* files are often opened in *Acrobat Reader*).



Note: To set the web browser that is used for displaying HTML/XHTML pages, go to **Window > Preferences > General > Web Browser** and specify it there.

- **Output file** - When **Open in Browser/System Application** is selected, you can use this button to automatically open the default output file at the end of the transformation.
- **Other location** - When **Open in Browser/System Application** is selected, you can use this option to open the file specified in this field at the end of the transformation. You can specify the path by using the text field, the **Insert Editor Variables** button, or the **Browse** button.

Open in editor

When this is enabled, at the end of the transformation, the default output file 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).

Troubleshooting DITA Transformation Errors

If a DITA transformation results in errors or warnings, the information is displayed in the message panel at the bottom of the editor. The information includes the severity, description of the problem, the name of the resource, and the path of the resource.

To help prevent and solve DITA transformation problems, follow these steps:

1. Validate your DITA documents by using the **Validate** action from the **Validation** toolbar drop-down menu, the **XML** menu, or from the **Validate** menu when invoking the contextual menu in the **Navigator** view.
2. If this action results in validation errors, solve them prior to executing the transformation. Also, you should pay attention to the warning messages because they may identify problems in the transformation.
3. [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.
6. If you need to contact the oXygen technical support team, they will need you to send the entire transformation scenario execution log. To obtain it:
 - a. Go to **Options > Preferences > DITA** preferences page and set the **Show console output** option to **Always**.
 - b. Execute the transformation scenario again. The console output messages are displayed in the **DITA OT** view.
 - c. Copy the entire log, save it in a text file, then send it to the oXygen technical support team.
 - d. After your issue has been solved, go back to the **Options > Preferences > DITA** preferences page and set the **Show console output** option to **When build fails**.

Ant Transformation

This type of transformation allows you to configure the options and parameters of an Ant build script.

An Ant transformation scenario is usually associated with an Ant build script. Oxygen XML Developer plugin runs an Ant transformation scenario as an external process that executes the Ant build script with the built-in Ant distribution (Apache Ant version 1.8.2) that is included 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:

- Use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu. Then click the **New** button and select **ANT transformation**.
- Use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the 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 to see if transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **ANT transformation**.

All three methods open the **New Scenario** dialog box.

The upper part of the dialog box allows you to specify the **Name** of the transformation scenario.

The lower part of the dialog box contains several tabs that allow you to configure the options that control the transformation.

Options Tab (Ant Transformations)

When you *create a new transformation scenario* or *edit an existing one*, a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Options** tab allows you to specify the following options:

Working directory

The path of the current directory of the Ant external process. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.

Build file

The Ant script file that is the input of the Ant external process. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** 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 transformation (such as `-verbose`).

Ant Home

You can choose between the default or custom Ant installation to run the transformation.

Java Home

You can choose between the default or custom Java installation to run the transformation. The default path is the Java installation that is used by Oxygen XML Developer plugin.

JVM Arguments

This parameter allows you to set specific parameters for the Java Virtual Machine used by Ant. For example, if it is set to `-Xmx384m`, the transformation process is allowed to use 384 megabytes of memory. When performing a large transformation, you may want to increase the memory allocated to the Java Virtual Machine. This will help avoid *Out of Memory* error messages (**OutOfMemoryError**).

Libraries

By default, Oxygen XML Developer plugin adds libraries (as high priority) 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.

Parameters Tab (Ant Transformations)

When you *create a new transformation scenario* or *edit an existing one*, a configuration dialog box allows you to customize the transformation with various options in several tabs.

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, and use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with their 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.

Below the table, 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. You can specify the **Value** of the parameter by using the  *Insert Editor Variables* button or the  **Browse** button.

Edit

Opens the **Edit Parameter** dialog box that allows you to change the value of the selected parameter or its description.

Delete

Removes the selected parameter from the list. It is enabled only for new parameters that have been added to the list.

Output Tab (Ant Transformations)

When you *create a new transformation scenario* or *edit an existing one*, a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Output** tab contains the following options:

Open

Allows you to specify the file to open automatically when the transformation is finished. This is usually the output file of the Ant process. You can specify the path by using the text field, the  *Insert Editor Variables* button, or the  **Browse** 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, in Windows *PDF* files are often opened in *Acrobat Reader*).
- **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).

Show console output

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

This type of transformation specifies the parameters and location of an XML document that the edited XSLT stylesheet is applied on. This scenario is useful when you develop an XSLT document and the XML document is in its final form.

To create an **XSLT transformation** scenario, use one of the following methods:

- Use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu. Then click the **New** button and select **XSLT transformation**.
- Use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the 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 to see if transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **XSLT transformation**.

All three methods open the **New Scenario** dialog box.

The upper part of the dialog box allows you to specify the **Name** of the transformation scenario.

The lower part of the dialog box contains several tabs that allow you to configure the options that control the transformation.

XSLT Tab

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **XSLT** tab contains the following options:

XML URL

Specifies the source XML file. You can specify the path by using the text field, its history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list. 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.x and a custom URI resolver is configured in the [advanced Saxon preferences page](#), the XML input of the transformation is passed to that URI resolver. 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 this field.

XSL URL

Specifies the source XSL file that the transformation will use. You can specify the path by using the text field, its history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list. 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.

Use "xml-stylesheet" declaration

If enabled, the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction. By default, this option is disabled and the transformation applies the XSLT stylesheet that is specified in the **XSL URL** field.

Transformer

This drop-down menu presents all the transformation engines available to Oxygen XML Developer plugin for performing a transformation. These include the built-in engines and [the external engines defined in the Custom Engines preferences page](#). The engine you choose 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 that preferences page.

Parameters

Opens a [Configure parameters dialog box](#) that allows you to configure the XSLT parameters used in the current transformation. In this dialog box, you can also configure the parameters for [additional XSLT stylesheets](#). 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 a [dialog box for configuring the XSLT extension jars or classes](#) that define extension Java functions or extension XSLT elements used in the transformation.

Additional XSLT stylesheets

Opens a [dialog box for adding XSLT stylesheets](#) that are applied on the main stylesheet specified in the **XSL URL** field. This is useful when a chain of XSLT stylesheets must be applied to the input XML document.

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 resulting dialog box includes a table that 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, and you can use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with their 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.

Below the table, 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.

Related information

[Editor Variables](#) on page 99

XSLT Extensions

The **Extensions** button is used to specify the jars and classes that contain extension functions called from the XSLT 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 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.

Additional XSLT Stylesheets

Use the **Additional XSLT Stylesheets** button in the **XSLT** tab to display a list of additional XSLT stylesheets to be used in the transformation and you can add files to the list or edit existing entries. 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.

Advanced Saxon HE/PE/EE XSLT Transformation Options

The XSLT 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 [those in the Saxon HE/PE/EE preferences page](#) but they are configured as a specific set of transformation options for each transformation scenario, while the values set in the preferences page apply as global options. The advanced options configured in a transformation scenario override the global options defined in the preferences page.

The advanced options for Saxon 9.6.0.7 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

Mode ("-im")

A Saxon-specific option that sets the initial mode for the transformation.

Template ("-it")

A Saxon-specific option that sets the name of the initial XSLT template to be executed.

Use a configuration file ("-config")

Sets a Saxon 9.6.0.7 configuration file that is executed for XSLT transformation and validation processes.

Debugger trace into XPath expressions (applies to debugging sessions)

Instructs the [XSLT Debugger](#) to *step into* XPath expressions.

Version warnings ("-versmsg")

Warns you when the transformation is applied to an XSLT 1.0 stylesheet.

Line numbering ("-l")

Line numbers where errors occur are included in the output messages.

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")

Specifies whether or not the source document will be validated against their associated DTD. You can choose from the following:

- **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")

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")

Allows you to choose how the *strip whitespaces* operation is handled. You can choose 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")

Allows you to set the optimization level. It is the value is an integer in the range of 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.

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.

The following advanced options are specific for Saxon 9.6.0.7 Professional Edition (PE) and Enterprise Edition (EE) only:

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 Developer plugin only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about [executing such a stylesheet on Saxonica's website](#).

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 `http://[URL]`). It ensures that the stylesheet cannot call arbitrary Java methods and thus gain privileged access to resources on your machine.

The advanced options that are specific for Saxon 9.6.0.7 Enterprise Edition (EE) are as follows:

XML Schema version

Use this option to change the default XML Schema version for this transformation. To change the default XML Schema version globally, [open the Preferences dialog box](#) and go to **XML > XML Parser > XML Schema** and use the [Default XML Schema version option](#).

Validation of the source file ("-val")

Requests schema-based validation of the source file and of any files read using `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 result 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 for non-fatal errors are written (wherever 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>.

FO Processor Tab (XSLT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **FO Processor** tab contains the following options:

Perform FO Processing

Specifies whether or not an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.

Input

Choose between the following options to specify which input file to use:

- **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. The available options depend on the selected processor type.

Processor

Specifies the FO processor to be used for the transformation. It can be the built-in Apache FOP processor or an [external processor](#).

Output Tab (XSLT Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

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. You can specify the path by using the text field, the  [Insert Editor Variables](#) button, or the  **Browse** button.

Open in Browser/System Application

If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, in Windows *PDF* files are often opened in *Acrobat Reader*).



Note: To set the web browser that is used for displaying HTML/XHTML pages, go to **Window > Preferences > General > Web Browser** and specify it there.

- **Output file** - When **Open in Browser/System Application** is selected, you can use this button to automatically open the default output file at the end of the transformation.
- **Other location** - When **Open in Browser/System Application** is selected, you can use this option to open the file specified in this field at the end of the transformation. You can specify the path by using the text field, the  [Insert Editor Variables](#) button, or the  **Browse** button.

Open in editor

When this is enabled, at the end of the transformation, the default output file 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

You can choose to view the results in one of the following:

- **XML** - If this is selected, Oxygen XML Developer plugin displays the transformation result in an XML viewer panel at the bottom of the application window with [syntax highlighting](#).
- **XHTML** - This option can only be selected if **Open in Browser/System Application** is disabled. If selected, Oxygen XML Developer plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.



Important: When transforming very large documents, you should be aware that enabling this feature may result in very long processing times. 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 selecting the **Open in Browser/System Application** option instead.

- **Image URLs are relative to** - If **Show in results view as XHTML** is selected, this option specifies the path used to resolve image paths contained in the transformation result. You can specify the path by using the text field, the  [Insert Editor Variables](#) button, or the  **Browse** button.

XProc Transformation

This type of transformation specifies the parameters and location of an XProc script.

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:

- Use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu. Then click the **New** button and select **XProc transformation**.
- Use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the 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 to see if transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **XProc transformation**.

All three methods open the **New Scenario** dialog box.

The upper part of the dialog box allows you to specify the **Name** of the transformation scenario.

The lower part of the dialog box contains several tabs that allow you to configure the options that control the transformation.

XProc Tab

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **XProc** tab contains the following options:

XProc URL

Specify the source XSL file to be used by the transformation. You can specify the path by using the text field, its history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list. 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.

Processor

Allows you to select the XProc engine to be used for the transformation. You can select the built-in *Calabash* engine or a custom engine that is [configured in the Preferences dialog box](#).

Inputs Tab (XProc Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

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 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. The *built-in editor variables* and *custom editor variables* can be used to specify the URL.

Edit

Opens an **Edit** dialog box that allows you to modify the selected port and its URL. The *built-in editor variables* and *custom editor variables* can be used to specify the URL.

Delete

Removes the selected port from the list. It is enabled only for new ports that have been added to the list.

Parameters Tab (XProc Transformations)

When you *create a new transformation scenario* or *edit an existing one*, a configuration dialog box allows you to customize the transformation with various options in several tabs.

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.

Outputs Tab (XProc Transformations)

When you *create a new transformation scenario* or *edit an existing one*, a configuration dialog box allows you to customize the transformation with various options in several tabs.

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. You can specify the path by using the text field, its history drop-down, the  *Insert Editor Variables* button, or the browsing tools in the  **Browse** drop-down list.

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.

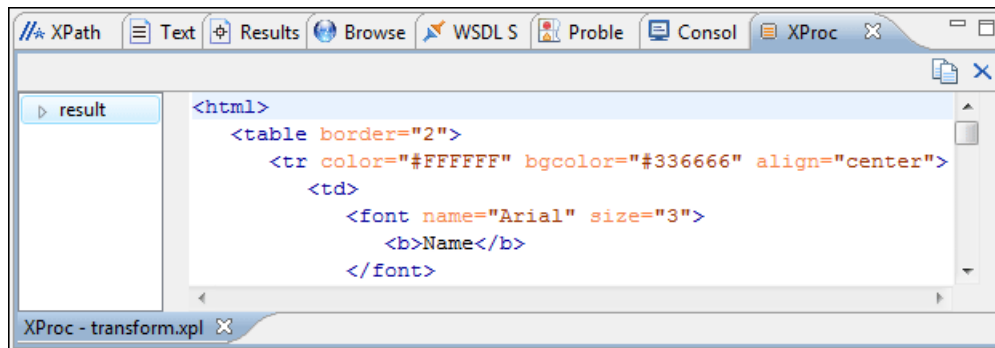


Figure 285: XProc Transformation Results View

Options Tab (XProc Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

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.

XQuery Transformation

This type of transformation specifies the parameters and location of an XML source that the edited XQuery file is applied on.



Note: When the XML source is a native XML database, the 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 input field of the scenario.

To create an **XQuery transformation** scenario, use one of the following methods:

- Use the **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu. Then click the **New** button and select **XQuery transformation**.
- Use the **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the 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 to see if transformation scenarios are associated with the edited document by hovering your cursor over the **Apply Transformation Scenario** button.

- Go to **Window > Show View** and select **Transformation Scenarios** to display this view. Click the **New** button and select **XQuery transformation**.

All three methods open the **New Scenario** dialog box.

The upper part of the dialog box allows you to specify the **Name** of the transformation scenario.

The lower part of the dialog box contains several tabs that allow you to configure the options that control the transformation.

XQuery Tab

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **XQuery** tab contains the following options:

XML URL

Specifies the source XML file. You can specify the path by using the text field, its history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list. 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.x and a custom URI resolver is configured in the [advanced Saxon preferences page](#), the XML input of the transformation is passed to that URI resolver.

XQuery URL

Specifies the source XQuery file to be used for the transformation. You can specify the path by using the text field, its history drop-down, the  [Insert Editor Variables](#) button, or the browsing tools in the  **Browse** drop-down list. 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.

Transformer

This drop-down menu presents all the transformation engines available to Oxygen XML Developer plugin for performing a transformation. These include the built-in engines and [the external engines defined in the Custom Engines preferences page](#). The engine you choose 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 that preferences page.

Parameters

Opens the [Configure parameters dialog box](#) for configuring the XQuery parameters. You can use the buttons in this dialog box to 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.

Extensions

Opens a [dialog box for configuring the XQuery extension jars or classes](#) that define extension Java functions or extension XSLT elements used in the transformation.

XQuery Parameters

The global parameters of the XQuery file used in a transformation scenario can be configured by using the **Parameters** button in the **XQuery** tab.

The resulting dialog box includes a table that displays all the parameters of the current XQuery file, along with their descriptions and current values. You can also add, edit, and remove parameters, and use the **Filter** text box to search for a specific term in the entire parameters collection. Note that edited parameters are displayed with their name in bold.

If the **XPath** column is checked, the parameter value is evaluated as an XPath expression before starting the XQuery 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 XQuery file location. You can use full paths or editor variables (such as `${cfdu}` [current file directory]) to specify other locations: `doc('${cfdu}/test.xml')//*`
2. Only XPath functions are allowed.

Below the table, 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.

Related information

[Editor Variables](#) on page 99

XQuery Extensions

The **Extensions** button is used to specify the jars and classes that contain extension functions called from the 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 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.

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 [those in the Saxon HE/PE/EE preferences page](#) but they are configured as a specific set of transformation options for each transformation scenario, while the values set in the preferences page apply as global options. The advanced options configured in a transformation scenario override the global options defined in the preferences page.

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")

Allows you to choose how the *strip whitespaces* operation is handled. You can choose 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")

Allows you to set the optimization level. It is the value is an integer in the range of 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 enabled (default value), Saxon runs the XQuery transformation with the XQuery 3.0 support.

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.

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 9.6.0.7 configuration file that is used for XQuery transformation and validation scenarios.

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 `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 result 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 for non-fatal errors are written (wherever 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>.

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 an XQuery Update are generated. This option is available when the **Enable XQuery update** option is enabled.

FO Processor Tab (XQuery Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **FO Processor** tab contains the following options:

Perform FO Processing

Specifies whether or not an FO processor is applied (either the built-in Apache FOP engine or an external engine defined in **Preferences**) during the transformation.

Input

Choose between the following options to specify which input file to use:

- **XQuery result as input** - The FO processor is applied to the result of the XQuery transformation that is 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. The available options depend on the selected processor type.

Processor

Specifies the FO processor to be used for the transformation. It can be the built-in Apache FOP processor or an [external processor](#).

Output Tab (XQuery Transformations)

When you [create a new transformation scenario](#) or [edit an existing one](#), a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Output** tab contains the following options:

Present as a sequence

Enabling this option will reduce the time necessary to fetch the full results, as it will only fetch the first chunk of the results.

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. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.

Open in Browser/System Application

If enabled, Oxygen XML Developer plugin automatically opens the result of the transformation in a system application associated with the file type of the result (for example, in Windows *PDF* files are often opened in *Acrobat Reader*).



Note: To set the web browser that is used for displaying HTML/XHTML pages, go to **Window > Preferences > General > Web Browser** and specify it there.

- **Output file** - When **Open in Browser/System Application** is selected, you can use this button to automatically open the default output file at the end of the transformation.
- **Other location** - When **Open in Browser/System Application** is selected, you can use this option to open the file specified in this field at the end of the transformation. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.

Open in editor

When this is enabled, at the end of the transformation, the default output file 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

You can choose to view the results in one of the following:

- **XML** - If this is selected, Oxygen XML Developer plugin displays the transformation result in an XML viewer panel at the bottom of the application window with [syntax highlighting](#).

- **XHTML** - This option can only be selected if **Open in Browser/System Application** is disabled. If selected, Oxygen XML Developer plugin displays the transformation result in a built-in XHTML browser panel at the bottom of the application window.
-  **Important:** When transforming very large documents, you should be aware that enabling this feature may result in very long processing times. 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 selecting the **Open in Browser/System Application** option instead.
- **Image URLs are relative to** - If **Show in results view as XHTML** is selected, this option specifies the path used to resolve image paths contained in the transformation result. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.

SQL Transformation

This type of 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.

To create an **SQL transformation** scenario, use one of the following methods:

- Use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu. Then click the **New** button and select **SQL transformation**.
- Use the  **Apply Transformation Scenario(s)** (**Alt + Shift + T, T (Command + Alt + T, T on OS X)**) action from the 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 to see if transformation scenarios are associated with the edited document by hovering your cursor over the  **Apply Transformation Scenario** button.

- Go to **Window > Show View** and select  **Transformation Scenarios** to display this view. Click the **New** button and select **SQL transformation**.

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 specify the path by using the text field, its history drop-down, the  **Insert Editor Variables** button, or the browsing tools in the  **Browse** drop-down list.

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 add or configure parameters for the transformation.

Editing a Transformation Scenario

Editing a transformation scenario is useful if you need to configure some of its parameters.

 **Note:** You can edit transformation scenarios that are defined at project level only. To edit a transformation scenario that is associated with a predefined document type, [duplicate it and edit the duplicated scenario](#).

To configure an existing transformation scenario, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C (Command + Alt + T, C on OS X)**) action from the toolbar or the **XML** menu.

Step Result: The [Configure Transformation Scenario\(s\) dialog box](#) is opened.

2. Select the particular transformation scenario and click the **Edit** button at the bottom of the dialog box or from the contextual menu.



Tip: You could also select the scenario and the  **Edit** button in the **Transformation Scenarios** view to achieve the same result.

Result: This will open an **Edit scenario** configuration dialog box that allows you to configure various options in several tabs, depending on the type of transformation scenario that was selected.

Transformation Types

The [Configure Transformation Scenario\(s\) dialog box](#) contains a **Type** column that shows you the transformation type for each of the listed scenarios. Each type of transformation contains includes some tabs with various configuration options.

The following is a list of the transformation types and their particular tabs (click the name of each tab below to see details about all the options that are available):

- **DITA OT** - This type of transformation includes configurable options in the following tabs:
 - [Skins Tab](#) (Available for **WebHelp Classic** and **WebHelp Classic with Feedback**)
 - [Templates Tab](#) (Available for **WebHelp Responsive** and **WebHelp Responsive with Feedback**)
 - [FO Processor Tab](#) (Available for PDF output)
 - [Parameters Tab](#)
 - [Filters Tab](#)
 - [Advanced Tab](#)
 - [Output Tab](#)
- **ANT** - This type of transformation includes configurable options in the following tabs:
 - [Options Tab](#)
 - [Parameters Tab](#)
 - [Output Tab](#)
- **XSLT** - This type of transformation includes configurable options in the following tabs:
 - [XSLT Tab](#)
 - [FO Processor Tab](#)
 - [Output Tab](#)
- **XProc** - This type of transformation includes configurable options in the following tabs:
 - [XProc Tab](#)
 - [Inputs Tab](#)
 - [Parameters Tab](#)
 - [Outputs Tab](#)
 - [Options Tab](#)
- **XQuery** - This type of transformation includes configurable options in the following tabs:
 - [XQuery Tab](#)
 - [FO Processor Tab](#)
 - [Output Tab](#)

Related information

[Creating New Transformation Scenarios](#) on page 509

[Duplicating a Transformation Scenario](#) on page 545

[Configure Transformation Scenario\(s\) Dialog Box](#) on page 545

Duplicating a Transformation Scenario

Duplicating a transformation scenario is useful for creating a scenario that is similar to an existing one or to edit a predefined transformation scenario.

To configure an existing transformation scenario, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C** (**Command + Alt + T, C on OS X**)) action from the toolbar or the **XML** menu.

Step Result: The *Configure Transformation Scenario(s) dialog box* is opened.

2. Select the particular transformation scenario and click the **Duplicate** button at the bottom of the dialog box or from the contextual menu.



Tip: You could also select the scenario and the  **Duplicate** button in the **Transformation Scenarios** view to achieve the same result.

Result: This will open an **Edit scenario** configuration dialog box that allows you to configure various options in several tabs, depending on the type of transformation scenario that was selected. For information about all the specific options in the various tabs, see the *Transformation Types section*.

Related information

[Creating New Transformation Scenarios](#) on page 509

[Editing a Transformation Scenario](#) on page 543

Configure Transformation Scenario(s) Dialog Box

You can use the **Configure Transformation Scenarios(s)** dialog box for *editing exiting transformation scenarios* or *creating new ones*.

To open this dialog box, use the  **Configure Transformation Scenario(s)** (**Alt + Shift + T, C** (**Command + Alt + T, C on OS X**)) action from the toolbar or the **XML** menu.

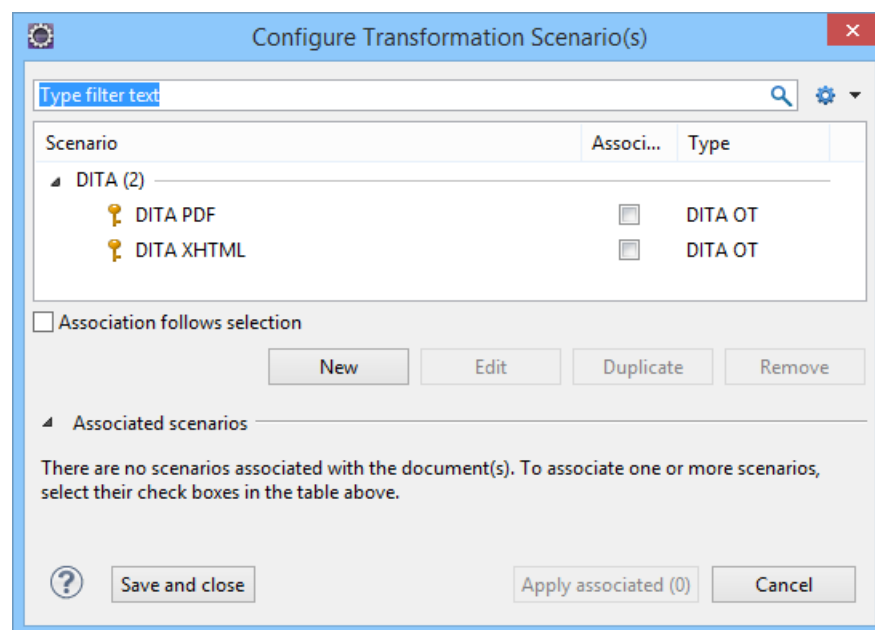


Figure 286: Configure Transformation Scenario(s) Dialog Box

The top section of the dialog box contains a filter that allows you to search through the scenarios list and the  **Settings** button allows you to configure the following options:

Show all scenarios

Select this option to display all the available scenarios, regardless of the document they are associated with.

Show only the scenarios available for the editor

Select this option to only display the scenarios that Oxygen XML Developer plugin can 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 Developer plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:

- Keep or replace the existing scenario.
- Keep both scenarios.



Note: When you keep both scenarios, Oxygen XML Developer plugin adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export selected scenarios individually. Oxygen XML Developer plugin creates a `scenarios` file that contains the scenarios that you export. This is useful if you want to share scenarios with others or export them to another computer.

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 or not 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 types of transformation scenarios that are available in Oxygen XML Developer plugin, see the [Transformation Types 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 also includes the following actions:

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 checkbox 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](#).

Edit

This button opens the **Edit Scenario** dialog box that allows you to configure the options of the transformations scenario. For information about all the specific options in the various tabs, see the [Transformation Types section](#).



Note: If you try to edit a transformation scenario associated with a defined document type, Oxygen XML Developer plugin displays a warning message to inform you that this is not possible and gives you the option to create a [duplicate transformation scenario](#) to edit instead.

Duplicate

Use this button to create a [duplicate transformation scenario](#).

Remove

Use this button to remove transformation scenarios.



Note: Removing scenarios associated with a defined document type is not allowed.

The **Edit**, **Duplicate**, and **Remove** actions are also available in the contextual menu of the transformation scenarios listed in the middle section of the dialog box (along with  **Import scenarios** and  **Export selected scenarios**).

Related information

[Editing a Transformation Scenario](#) on page 543

[Duplicating a Transformation Scenario](#) on page 545

Apply Batch Transformations

A transformation action can be applied on a batch of selected files [from the contextual menu of the Navigator 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. Select the files you want to transform and from the contextual menu, select **Transform** >  **Configure Transformation Scenario(s)** to choose one or more transformation scenarios to be applied on all the files in the logical folder.
2. Use Oxygen XML Developer plugin [editor variables](#) to specify the input and output files. This ensures that each file from the selected set of resources is processed and that the output is not overwritten by the subsequent processing.
 - a) 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.
 - b) Edit the transformation scenario to make sure the appropriate editor variable is assigned for the output file. For example, for an *XML transformation with XSLT*, switch to the **Output** tab and set the path of the output file using a construct of editor variables, such as `${cfd}/${cfn}.html`.
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.

Related information

[Editor Variables](#) on page 99

Sharing 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, select **Window > Show View > Transformation Scenarios**.

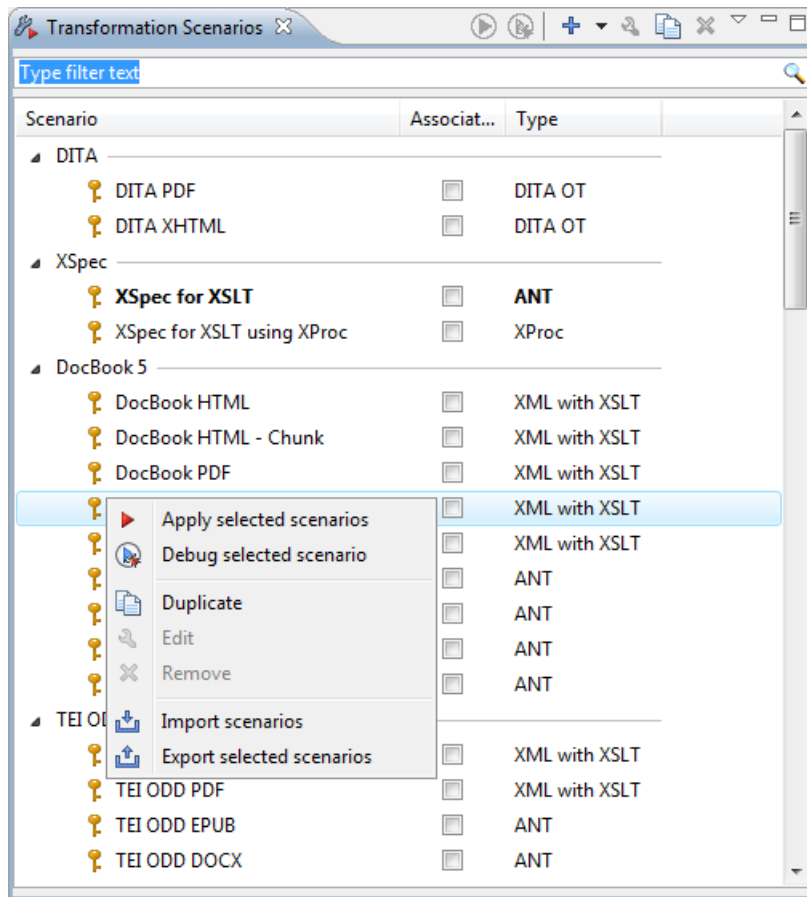


Figure 287: Transformation Scenarios view

Oxygen XML Developer plugin supports multiple scenarios association. To associate multiple scenarios with a document, enable the check-boxes in front of each scenario. You can also associate multiple scenarios with a document from the [Configure Transformation Scenario\(s\) dialog box](#).

The **Transformation Scenarios** view presents both global and project-level scenarios. By default, Oxygen XML Developer plugin presents the items in the following order:

1. Scenarios that match the current framework.
2. Scenarios that match the current project.
3. Scenarios that match other frameworks.

Toolbar/Contextual Menu Actions and Options

The following actions and options are available on the toolbar or in the contextual menu:

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).

▾ **New**

This drop-down menu contains a list of the [scenarios that you can create](#). Oxygen XML Developer plugin determines the most appropriate scenarios for the current type of file and displays them at the beginning of the list, followed by the rest of the scenarios.

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 that allows you to configure various options in several tabs, depending on the type of transformation scenario that was selected. For information about all the specific options in the various tabs, see the [Transformation Types section](#).

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 Developer plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:

- Keep or replace the existing scenario.
- Keep both scenarios.



Note: When you keep both scenarios, Oxygen XML Developer plugin adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export transformation and validation scenarios individually. Oxygen XML Developer plugin creates a `scenarios` file that contains the scenarios that you export.

Settings

This drop-down menu allows you to configure the following options (many of these options are also available if you right-click the name of a column):

Show all scenarios

Select this option to display all the available scenarios, regardless of the document they are associated with.

Show only the scenarios available for the editor

Select this option to only display the scenarios that Oxygen XML Developer plugin can 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 Developer plugin ignores it. If a conflict appears (an imported scenario has the same name as an existing one), you can choose between two options:

- Keep or replace the existing scenario.
- Keep both scenarios.



Note: When you keep both scenarios, Oxygen XML Developer plugin adds `imported` to the name of the imported scenario.



Export selected scenarios

Use this option to export selected scenarios individually. Oxygen XML Developer plugin creates a **scenarios** file that contains the scenarios that you export. This is useful if you want to share scenarios with others or export them to another computer.

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.

Related information

[Editing a Transformation Scenario](#) on page 543

[Creating New Transformation Scenarios](#) on page 509

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. Its value should have a URL syntax like this: `file:///c:/path/to/DITA_OT_DIR/plugins/org.dita.pdf2/cfg`.
9. Debug the transformation scenario.

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_INSTALL_DIR]/lib/xproc/calabash/engine.xml` file.
2. Uncomment the `<system-property name="com.xmlcalabash.fo-processor" value="com.xmlcalabash.util.FoXEP"/>` system property.
3. Uncomment the `<system-property name="com.renderx.xep.CONFIG" file=".../tools/xep/xep.xml"/>` system property. Edit the `file` attribute to point to the configuration file that is usually located in the XEP installation folder.
4. Uncomment the references to the XEP libraries. Edit them to point to the matching library names from the XEP installation directory.

5. Restart Oxygen XML Developer plugin.

Integration of an External XProc Engine

The Javadoc documentation of the XProc API is available for download from the application website as a zip file [xprocAPI.zip](#).

To create an XProc integration project, follow these steps:

1. Move the `oxygen.jar` file from `[OXYGEN_INSTALL_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 (otherwise, `false`).

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_INSTALL_DIR]/lib/xproc`.
6. Place the `engine.xml` and all the libraries necessary to run the new integration in that folder.

XSLT Processors

This section explains how to configure an XSLT processor and extensions for such a processor in Oxygen XML Developer plugin.

Supported XSLT Processors

Oxygen XML Developer plugin includes the following XSLT processors:

- **Xalan 2.7.1** - [Xalan-Java](#) is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements XSL Transformations (XSLT) Version 1.0 and XML Path Language (XPath) Version 1.0.
- **Saxon 6.5.5** - [Saxon 6.5.5](#) is an XSLT processor that implements the Version 1.0 XSLT and XPath with a number of powerful extensions. This version of Saxon also includes many of the new features that were first defined in the XSLT 1.1 working draft, but for conformance and portability reasons these are not available if the stylesheet header specifies `version="1.0"`.
- **Saxon 9.6.0.7 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 Developer plugin. Saxon EE includes an XML Schema processor, and schema-aware XSLT, XQuery, and XPath processors.

The validation in schema aware transformations is done according to the W3C XML Schema 1.0 or 1.1. This can be [configured in Preferences](#).



Note: Oxygen XML Developer plugin implements a Saxon framework that allows you to create Saxon configuration files. Two templates are available: **Saxon collection catalog** and **Saxon configuration**. Both of these templates support content completion, element annotation, and attribute annotation.



Note: Saxon can use the *ICU-J localization library* (`saxon9-icu.jar`) to add support for sorting and date/number formatting in a wide variety of languages. This library is not included in the Oxygen XML Developer plugin

installation kit. However, Saxon will use the default collation and localization support available in the currently used JRE. To enable this capability follow these steps:

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 Developer plugin.

- **Saxon-CE (Client Edition)** is Saxonica's implementation of XSLT 2.0 for use on web browsers. Oxygen XML Developer plugin provides support for editing stylesheets that contain Saxon-CE extension functions and instructions. This support improves the validation, content completion, and syntax highlighting.



Note: Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Developer plugin only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about [executing such a stylesheet on Saxonica's website](#).



Note: A specific template, named **Saxon-CE 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 Developer plugin is 2.7.6 and the `Libxslt` version is 1.1.26.

Oxygen XML Developer plugin uses `Libxslt` through its command line tool (`Xsltproc`). The XSLT processor is included in the distribution kit of the stand-alone version for Windows and Mac OS X. Since there are differences between various Linux distributions, on Linux you must install `Libxslt` on your machine as a separate application and set the `PATH` variable to contain the `Xsltproc` executable.

If you do not have the `Libxslt` library already installed, you should copy the following files from Oxygen XML Developer plugin stand-alone installation directory to the root of the `com.oxygenxml.editor_18.1` plugin:

- on Windows: `xsltproc.exe`, `zlib1.dll`, `libxslt.dll`, `libxml2.dll`, `libexslt.dll`, `iconv.dll`
- on Linux: `xsltproc`, `libexslt.so.0`, `libxslt.so.1`, `libxml2.so.2`
- on Mac OS X: `xsltproc.mac`, `libexslt`, `libxslt`, `libxml`



Note: The `Xsltproc` processor can be configured from the [XSLTPROC options page](#).



CAUTION: There is a known problem where file paths that contain spaces are not handled correctly in the LIBXML processor. For example, the built-in XML catalog files of the predefined document types (DocBook, TEI, DITA, etc.) are not handled properly by LIBXML if Oxygen XML Developer plugin is installed in the default location on Windows (C:\Program Files). This is because the built-in XML catalog files are stored in the `[OXYGEN_INSTALL_DIR]/frameworks` subdirectory of the installation directory, and in this case it contains a space character.

- **MSXML 4.0** - [MSXML 4.0](#) is available only on Windows platforms. It can be used for [transformation](#) and [validation of XSLT stylesheets](#).

Oxygen XML Developer plugin uses the Microsoft XML parser through its command line tool `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 Developer plugin performs XSLT transformations and validations using .NET Framework's XSLT implementation (`System.Xml.Xsl.XslTransform` class) through the `nxslt` command line utility. The `nxslt` version included in Oxygen XML Developer plugin is 1.6.

You should have the .NET Framework version 1.0 already installed on your system. Otherwise, you will get the following warning: MSXML.NET requires .NET Framework version 1.0 to be installed. Exit code: 128.

You can get the .NET Framework version 1.0 from the [Microsoft website](#).

- **.NET 1.0** - A transformer based on the System.Xml 1.0 library available in the .NET 1.0 and .NET 1.1 frameworks from Microsoft (<http://msdn.microsoft.com/xml/>). It is available only on Windows.

You should have the .NET Framework version 1.0 or 1.1 already installed on your system. Otherwise, you will get the following warning: MSXML.NET requires .NET Framework version 1.0 to be installed. Exit code: 128.

You can get the .NET Framework version 1.0 from the [Microsoft website](#).

- **.NET 2.0** - A transformer based on the System.Xml 2.0 library available in the .NET 2.0 framework from [Microsoft](#). It is available only on Windows.

You should have the .NET Framework version 2.0 already installed on your system. Otherwise, you will get the following warning: MSXML.NET requires .NET Framework version 2.0 to be installed. Exit code: 128.

You can get the .NET Framework version 2.0 from the [Microsoft website](#).

Configuring Custom XSLT Processors

Oxygen XML Developer plugin allows you to configure custom processors to be used for running XSLT and XQuery transformations.

To add a new custom processor, follow these steps:

1. **Open the [Preferences dialog box](#)** and go to **XML > XSLT-FO-XQuery > Custom Engines**.
2. Click the  **New** button at the bottom of the dialog box.
3. Configure the [parameters for the custom engine](#).
4. Click **OK**.



Note:

The output messages of a custom processor are displayed in an output view at the bottom of the application window. If an output message follows [the format of an Oxygen XML Developer plugin linked message](#), clicking it highlights the location of the message in an editor panel containing the file referenced in the message.

Related information

[Custom Engines Preferences](#) on page 83

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 Developer plugin.

Built-in XSL-FO Processor

The Oxygen XML Developer plugin installation package is distributed with the [Apache FOP](#) that is a Formatting Objects processor for rendering your XML documents to PDF. *FOP* is a print and output independent formatter driven by XSL Formatting Objects. *FOP* is implemented as a Java application that reads a formatting object tree and renders the resulting pages to a specified output.

Other FO processors can be configured in [the Preferences dialog box](#).

Add a Font to the Built-in FO Processor - 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 that has 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 Developer plugin install directory.
- c) Create the following script file in the Oxygen XML Developer plugin installation directory.

For OS X and Linux create a file `ttfConvert.sh`:

```
#!/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:$LIB/commons-logging-1.1.3.jar
export CP=$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-1.1.3.jar
set CP=%CP%;%LIB%\commons-io-1.3.1.jar
set CP=%CP%;%LIB%\xmlgraphics-commons-1.5.jar
set CP=%CP%;%LIB%\xml-apis.jar
set CMD=java -cp "%CP%" org.apache.fop.fonts.apps.TTFReader
set FONT_DIR=C:\Windows\Fonts
%CMD% %FONT_DIR%\Arialuni.ttf Arialuni.xml
```

The paths specified in the file are relative to the Oxygen XML Developer plugin installation directory. If you decide to create it in other directory, change the file paths accordingly.

The `FONT_DIR` can be something 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>
      <font>
        <font metrics-url="Arialuni.xml" kerning="yes"
              embed-url="file:/Library/Fonts/Arialuni.ttf">
          <font-triplet name="Arialuni" style="normal"
                    weight="normal"/>
        </font>
      </font>
    </renderer>
  </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">
  ...
  <font>
    <font metrics-url="Arialuni.xml" kerning="yes"
          embed-url="file:/Library/Fonts/Arialuni.ttf">
      <font-triplet name="Arialuni" style="normal"
                    weight="normal"/>
    </font>
    <font 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>
  </font>
  ...
</fop>
```

More details about the FOP configuration file are available on the FOP website.

- b) [Open the Preferences dialog box](#), go to **XML > XSLT/FO/XQuery > FO Processors**, and enter the path of the FOP configuration file in the **Configuration file** text field.
4. Set the font on the document content.

This is usually done with XSLT stylesheet parameters and depends on the document type processed by the stylesheet.

For DocBook documents, you can start with the predefined scenario called **DocBook PDF**, [edit the XSLT parameters](#), and set the font name (in our example **Arialuni**) to the parameters `body.font.family` and `title.font.family`.

For TEI documents, you can start with the predefined scenario called **TEI PDF**, [edit the XSLT parameters](#), and set the font name (in our example **Arialuni**) to the parameters `bodyFont` and `sansFont`.

For DITA to PDF transformations using DITA-OT modify the following two files:

- `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` that has the attribute `mime="application/pdf"`:

```
<renderer mime="application/pdf">
  ...
  <font>
    <font metrics-url="Arialuni.xml" kerning="yes"
          embed-url="file:/Library/Fonts/Arialuni.ttf">
      <font-triplet name="Arialuni" style="normal"
                    weight="normal"/>
    </font>
  </font>
</renderer>
```

```

        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_INSTALL_DIR]/lib/fop` directory.

To add support for hyphenation, follow these steps:

1. Create a folder called `fop` in the `[OXYGEN_INSTALL_DIR]/lib` folder.
2. Download the compiled JAR from [OFFO](#).
3. Copy the `fop-hyph.jar` file into the `[OXYGEN_INSTALL_DIR]/lib/fop` folder.
4. Restart Oxygen XML Developer plugin.

Adding Support for PDF Images

To add support for PDF images, follow these steps:

1. Create a folder called `fop` in the `[OXYGEN_INSTALL_DIR]/lib` folder.
2. Download the [fop-pdf-images](#) JAR libraries.
3. Copy the libraries into the `[OXYGEN_INSTALL_DIR]/lib/fop` folder.
4. Restart Oxygen XML Developer 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.

WebHelp System Output

Oxygen XML Developer plugin allows you to obtain WebHelp Classic and WebHelp Responsive outputs. This section contains information about the WebHelp system, its variants, and ways to customize it to better fit your specific needs.

Table 31: WebHelp System Feature Matrix

Features	WebHelp System Variants				
	Responsive w/ Feedback	Responsive	Classic w/ Feedback	Classic	Classic Mobile
Desktop Systems	✓	✓	✓	✓	

Features	WebHelp System Variants				
	Responsive w/ Feedback	Responsive	Classic w/ Feedback	Classic	Classic Mobile
Mobile Devices	✓	✓			✓
Predefined Skins	✓	✓	✓	✓	
Predefined Templates	✓	✓			
Search Capabilities	✓	✓	✓	✓	✓
Modern Layout	✓	✓			
Adaptable to Any Screen Size	✓	✓			
Comments Section	✓		✓		
DITA Documents	✓	✓	✓	✓	✓
DocBook Documents			✓	✓	✓
Tri-Pane Frames or Frameless Version			✓	✓	

WebHelp Responsive System

WebHelp is a form of online help that consists of a series of web pages (XHTML format). Its advantages include platform independence, ability to update content continuously, and it can be viewed using a regular web browser. The Oxygen XML Developer plugin WebHelp system includes several variants to suit your specific needs. The *WebHelp Responsive* variant features a very flexible layout, is designed to adapt to any screen size, and is available for DITA document types.

This type of WebHelp system can be generated by using the [DITA Map WebHelp Responsive transformation scenario](#).

Layout

The layout of the *WebHelp Responsive* system is platform independent and is able to adapt to any screen size. It is highly customizable and relies on a [template mechanism](#) that allows you to control the position of various functional [template components](#) to suit your particular requirements.

You can select from several different styles of layouts (for example, by default, you can select either a *tiles* or *tree* style of layout). Furthermore, each of these layouts include a collection of skins that you can choose from, or you can customize your own.

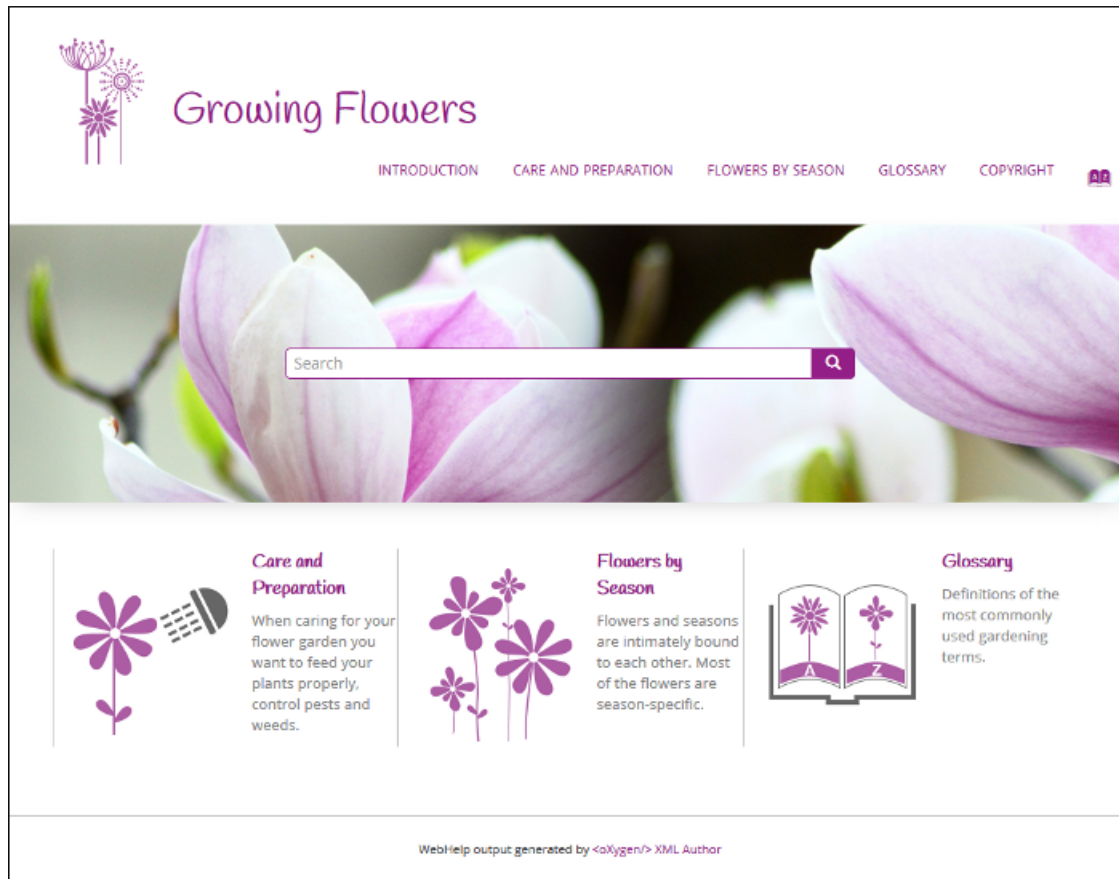


Figure 288: WebHelp Responsive Output on a Normal Screen

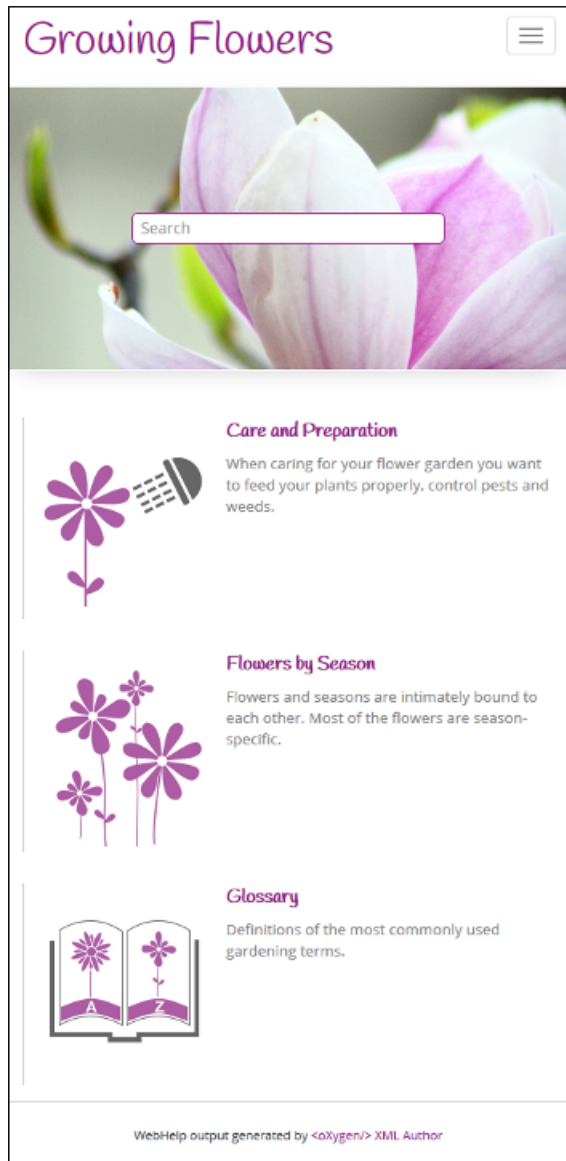


Figure 289: WebHelp Responsive Output on a Narrow Screen

Search Feature

When you enter search terms in the **Search** field, the results are displayed in the results page. When you click on a result, the topic is opened in the main pane and the search results are highlighted.

The **Search** feature is also 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

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.



Note: For DITA content, if you have topics that are referenced multiple times in your DITA map structure, they will be found by the search engine that same number of times in the search results. The first occurrence will have a **Similar results** link displayed below the search result and clicking this link will display the other occurrences. Note that the content may be different in some of the occurrences if they have different profiling applied than the others.

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 [Editing Scoring Values of Tag Elements in Search Results](#) on page 588.

This output format is compatible with the most recent versions of the following common browsers:

- Edge
- Internet Explorer (IE 8 or newer)
- Chrome
- Firefox
- Safari
- Opera

WebHelp Responsive with Feedback System

WebHelp is a form of online help that consists of a series of web pages (XHTML format). Its advantages include platform independence, ability to update content continuously, and it can be viewed using a regular web browser. The Oxygen XML Developer plugin WebHelp system includes several variants to suit your specific needs. The *WebHelp Responsive with Feedback* variant features a very flexible layout, is designed to adapt to any screen size, and includes a feedback system that allows your users to make comments and allows you to manage and reply to them. This variant is available for DITA document types.

This type of WebHelp system can be generated by using the [DITA Map WebHelp Responsive with Feedback transformation scenario](#) and following the [instructions for deploying the system](#).

Layout

The layout of the *WebHelp Responsive with Feedback* system is platform independent and is able to adapt to any screen size. It is highly customizable and relies on a [template mechanism](#) that allows you to control the position of various functional [template components](#) to suit your particular requirements.

You can select from several different styles of layouts (for example, by default, you can select either a *tiles* or *tree* style of layout). Furthermore, each of these layouts include a collection of skins that you can choose from, or you can customize your own.

The *WebHelp Responsive with Feedback* system also contains a **Comments** section at the bottom of the pane. This section is where you can interact with users through a comment system.

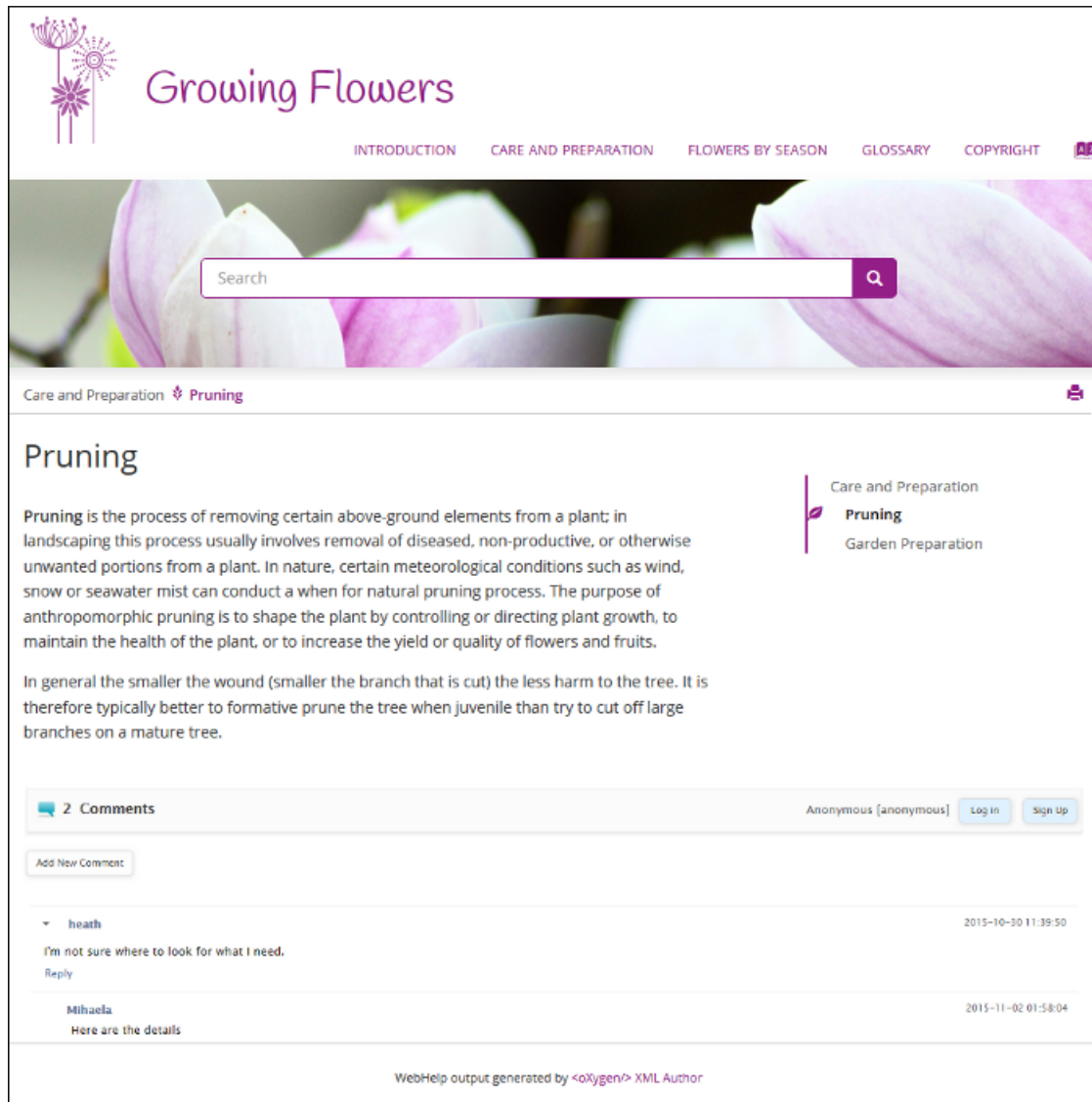


Figure 290: WebHelp Output

Managing Comments

To add a new comment, click the **Add New Comment** button, or click **Reply** to add a comment to an existing thread. You can click on the **Log in** button on the right side of this bar to be authenticated as a user 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.

After you log in, your name and user name are displayed in the **Comments** bar, along with the **Log off** and **Edit** buttons. Click the **Edit** button to open the **User Profile** dialog box where 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 Classic 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.

If you are an administrator, you can manage user information and comments. For more information, see [Managing Users and Comments in a Feedback-Enabled WebHelp System](#) on page 566.

Search Feature

When you enter search terms in the **Search** field, the results are displayed in the results page. When you click on a result, the topic is opened in the main pane and the search results are highlighted.

The **Search** feature is also 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

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 [Editing Scoring Values of Tag Elements in Search Results](#) on page 588.

This output format is compatible with the most recent versions of the following common browsers:

- Edge
- Internet Explorer (IE 8 or newer)
- Chrome
- Firefox
- Safari
- Opera

Deploying a Feedback-Enabled WebHelp System

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Developer 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, Edge, Internet Explorer, Safari, Opera.

Create WebHelp with Feedback Database

The *WebHelp with Feedback* 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. Access the *phpMyAdmin* instance running on your server.
2. Click *Databases* (in the right frame) and then create a *database*. You can give it any name you want (for example *comments*).
3. Create a user with connection privileges for this database.
4. 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 with Feedback Output

If you have a web server configured with PHP and MySQL, you can deploy the *WebHelp with Feedback* 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/](http://[YOUR_SERVER]/myProductWebHelp/)). You are redirected to the 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_INSTALL_DIR]/feedback/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 Authentication** 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 multiple *WebHelp with Feedback* deployments. If a topic is available in multiple 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 with Feedback* 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 with Feedback* 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 three different WebHelp outputs, you have three 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 visible in other specified *products*.

Related information

[Managing Users and Comments in a Feedback-Enabled WebHelp System](#) on page 566

Refreshing the Content of a Feedback-Enabled WebHelp System

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.

Fortunately, you can refresh just the content without losing the comments or the initial system configuration. To do so, follow these steps:

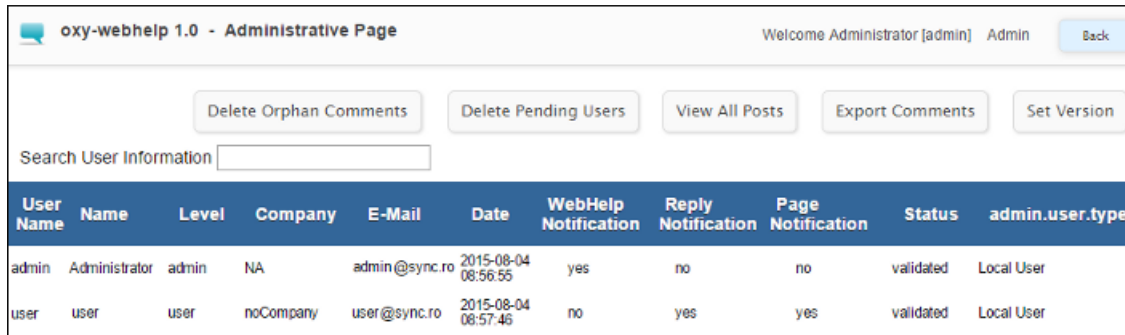
1. Execute the transformation scenario that produces the *WebHelp with Feedback* output directory.
2. Go to the output directory (specified in the **Output** tab of the transformation scenario), locate the `\feedback\resources\php\config\config.php` file, and delete it.
3. Locate the `\feedback\install` 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 Feedback-Enabled WebHelp 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 page.



User Name	Name	Level	Company	E-Mail	Date	WebHelp Notification	Reply Notification	Page Notification	Status	admin.user.type
admin	Administrator	admin	NA	admin@sync.ro	2015-08-04 08:56:55	yes	no	no	validated	Local User
user	user	user	noCompany	user@sync.ro	2015-08-04 08:57:45	no	yes	yes	validated	Local User

Figure 291: Administrative Page

3. Use this page 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 Responsive Template Mechanism

The WebHelp Responsive template mechanism is the base of the system and it is responsible for defining its output. It consists of a set of HTML template files and other additional resources (such as images, CSS, and JavaScript files). Each of the HTML template files contain one or more template components (such as title, table of contents, search input, etc.) whose placement inside the template will define the final layout of the output.

This mechanism allows you to create multiple layouts simply by creating templates that define the location of where the various components will be displayed.

Creating WebHelp Responsive Templates

To create a new WebHelp Responsive template, follow these steps:

1. Locate the following folder in your DITA-OT directory (*DITA_OT_DIR*):

```
DITA_OT_DIR/plugins/com.oxygenxml.webhelp/templates/dita/
```

2. Duplicate the `bootstrap` folder and rename it to whatever you want your new template to be identified as (for example, `myTemplate`).
3. Customize the structure of the new template according to your needs. For example, if you only want to keep one of the template variants, open the `myTemplate/variants` folder and delete all of its subdirectories, except for that one (for instance, the `tiles` directory). Keep in mind that the structure of the template directory is important. The names of folders at certain levels correspond to the names of templates and skins, while components and resources are defined and referenced in certain files or folders at specific locations within the directory structure. For more information, see [WebHelp Responsive Template Directory Structure](#) on page 568.
4. You can also customize the structure of the skins within the template variants. For example, if you only want to keep one of the skins in the `tiles` variant, open the `myTemplate/variants/tiles` folder and delete all of its subdirectory skins, except for that one (for instance, the `light` directory). You can also edit the `skin.css` file that is located in the `skin` directory to customize the styling. If your customization of the CSS file requires additional resources (such as images, fonts, or other CSS files), they need to be placed in the `resources` folder at the same level with the `skin.css` file.
5. To customize the components that appear in the WebHelp Responsive output, you can modify the HTML template files that define the output. For more information, see [WebHelp Responsive Template Files](#) on page 569.

Related information

[Customizing the WebHelp Responsive Output](#) on page 578

WebHelp Responsive Template Directory Structure

A certain directory structure is required for the WebHelp Responsive templates. The names of folders at certain levels correspond to the names of template variants and skins, components are defined in specific files, and various resources need to be located in specific locations within the directory structure.

The templates are stored in `DITA_OT_DIR/plugins/com.oxygenxml.webhelp/templates/dita`.

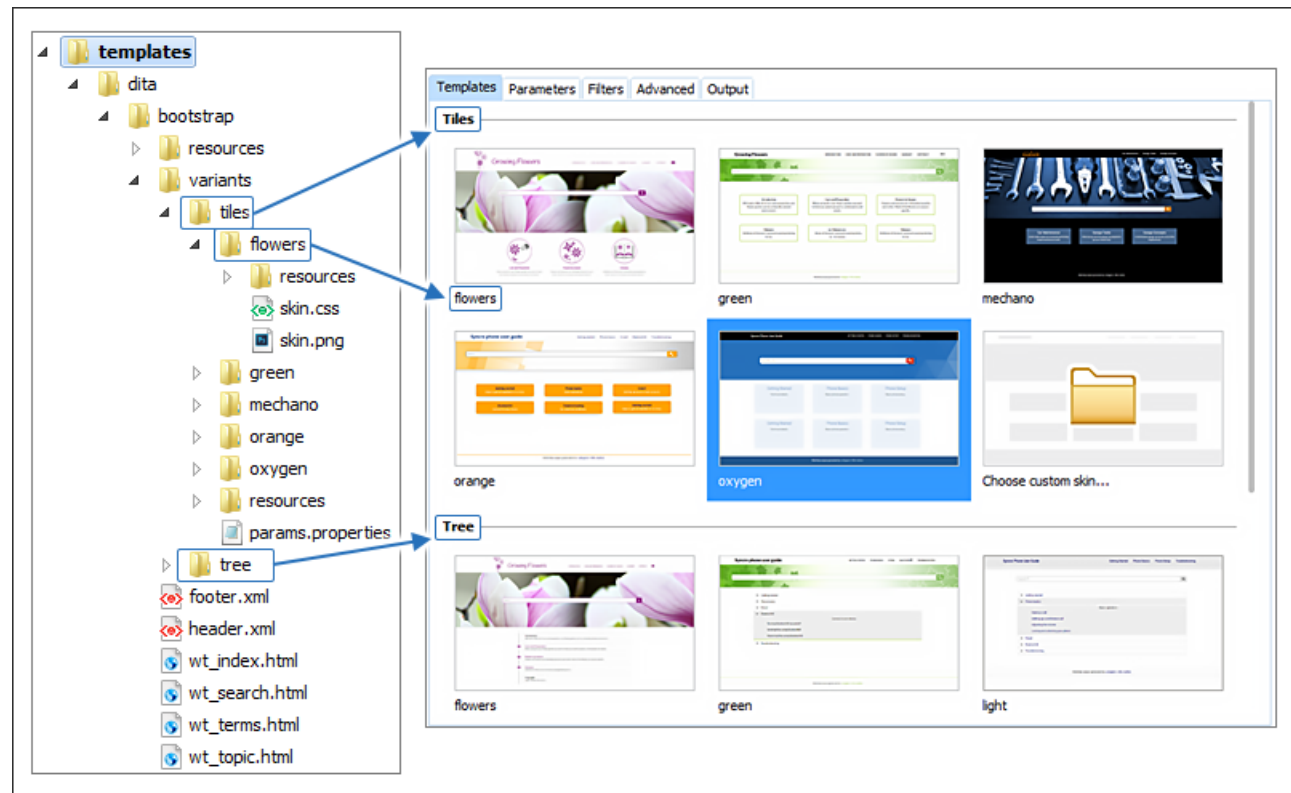


Figure 292: Templates Directory Structure

At the first level of the template directory we can find the following predefined files and folders:

- **resources Folder** - Contains all additional resources used by the template, such as images, CSS, and JavaScript files.
- **variants Folder** - Contains the template variants, including folders for each skin. See [Template Variants and Skins](#) on page 568.
- **Template Files** - The HTML template files that are used to generate the output:
 - `wt_index.html` - Used to produce the main home page of the WebHelp Responsive output. See [WebHelp Responsive Main Page Template](#) on page 569.
 - `wt_topic.html` - Used to generate the HTML pages associated with individual topics. See [WebHelp Responsive Topic Template](#) on page 571.
 - `wt_search.html` - Used to generate the HTML page that presents the search results. See [WebHelp Responsive Search Results Template](#) on page 572.
 - `wt_terms.html` - Used to generate the HTML page that presents the documentation index. See [WebHelp Responsive Index Terms Template](#) on page 572.

After the transformation scenario is executed, the resources and variants folders are copied in the `/oxygen-webhelp/template/` folder within the output directory (defined in the **Output** tab of the transformation scenario).

Template Variants and Skins

You could think of a *template* as being a set of WebHelp components that are placed in a predefined HTML layout. You can have multiple variants of the template. A WebHelp *template variant* is an instance of the template with a specific set of

parameters. For example, you could have two variants of the WebHelp main page, one that displays the topics in a *tiles* style of layout, and another one that displays the topics in a *tree* style.

Each variant has its own directory that corresponds to the name of the variant. The name of the variant is displayed in the user interface when the variants are displayed (for example, in the **templates** tab of the transformation scenario).

Each variant directory may contain the following resources:

- **Skin Directories** - These folders represent skin templates for the current variant.
- **params.properties File** - This file specifies the values for the parameters imposed by the variant.
- **resources Folder** - This is an optional directory that contains resources that are specific to the current variant (such as images, CSS files, etc.) They will be copied to the output directory.

Skins

A variant *skin* represents a CSS file that allows you to alter the styling of the template. The CSS associated with a skin must be named **skin.css** and it must be stored as a first child of the skin directory.

Each skin might need additional resources (images, fonts) that must be stored in the **resources** directory in the root folder for that particular skin. The name of the skin directory is displayed in the user interface when you choose a skin (in the **templates** tab of the transformation scenario).

Each skin directory can also contain a **skin.png** preview image that will be displayed in the user interface and a properties file that contains a URL for the online preview of the skin. This image file must be stored as a first child of the skin directory.

For information about creating or customizing skins, see [Create or Customize a WebHelp Responsive Skin](#) on page 581.

WebHelp Responsive Template Files

The HTML pages that comprise the output of a WebHelp Responsive system are obtained after processing HTML template files. These files correspond to the type of page they generate in the output.

There are four types of template pages and corresponding HTML template files:

1. **Main Page Template** (wt_index.html file) - Used to produce the main home page of the WebHelp Responsive output.
2. **Topic Template** (wt_topic.html file) - Used to generate the HTML pages associated with individual topics.
3. **Search Results Template** (wt_search.html file) - Used to generate the HTML page that presents the search results.
4. **Index Terms Template** (wt_terms.html file) - Used to generate the HTML page that presents the documentation index.

The HTML template files are located in the following directory:

```
DITA_OT_DIR/plugins/com.oxygenxml.webhelp/templates/dita/bootstrap/
```

WebHelp Responsive Main Page Template

The *Main Page Template* is used to generate the home page of the WebHelp Responsive output. The name of the template file is wt_index.html and it is located in the following directory:

```
DITA_OT_DIR/plugins/com.oxygenxml.webhelp/templates/dita/bootstrap/
```

The main function of the home page is to display top level information and provide links that help you easily navigate to any of the top level topics of the publication. These links can be rendered in either a *Tiles* or *Tree* style of layout. The HTML page produced for the home page also consists of various other components, such as a logo, title, menu, search field, or index link.

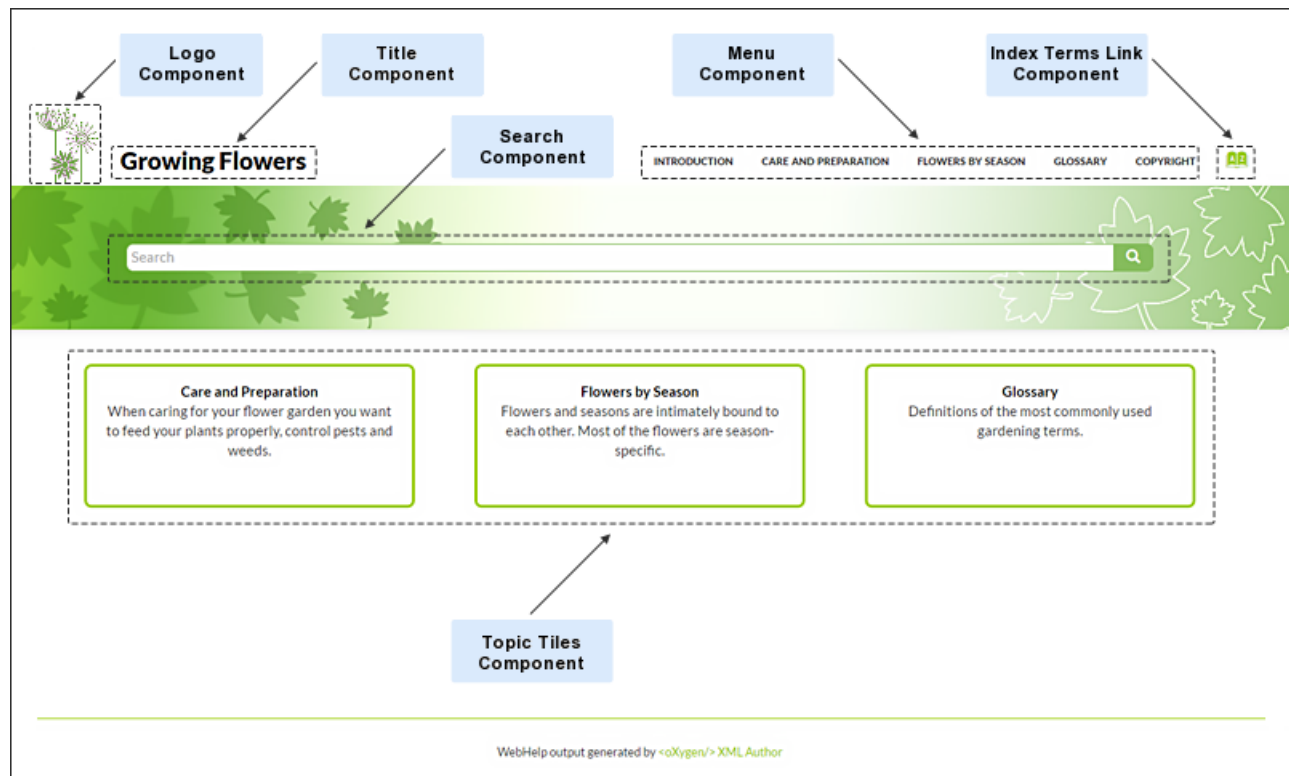


Figure 293: Example: Main Page Components of a Tiles Style of Layout

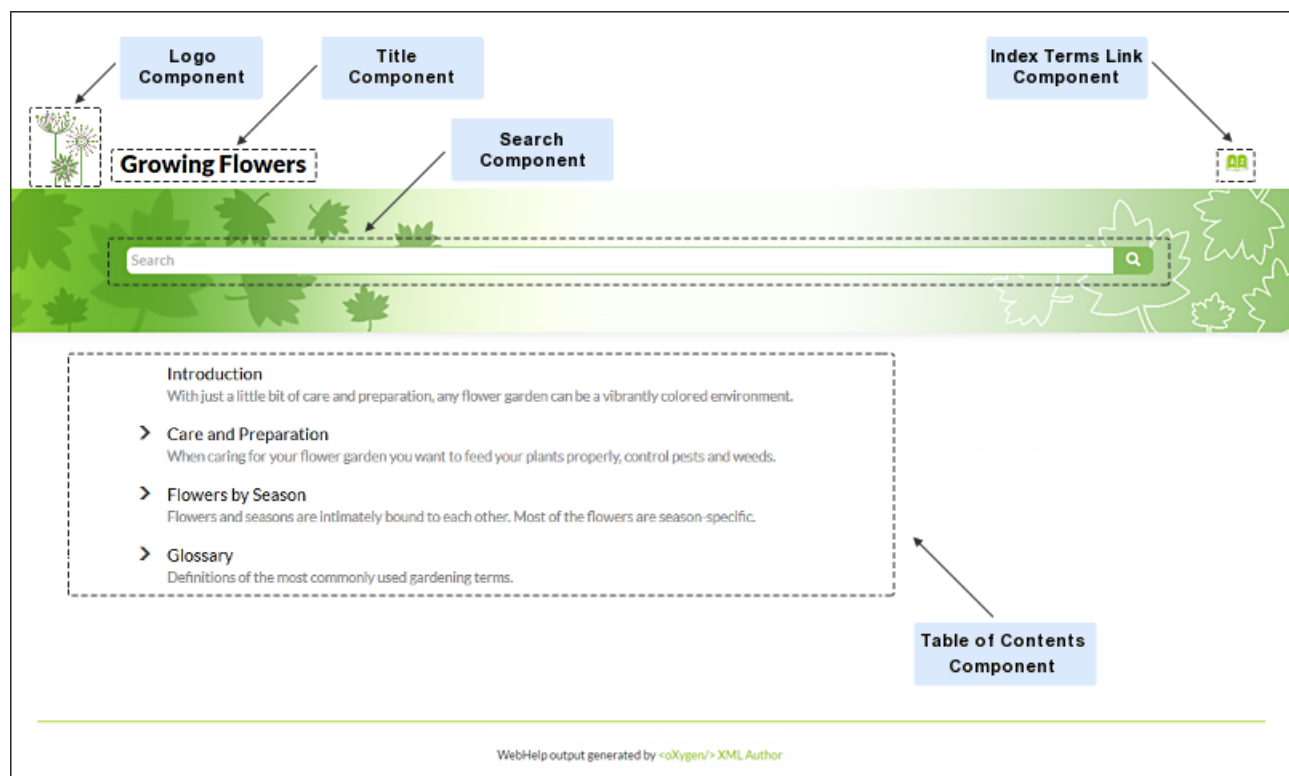


Figure 294: Example: Main Page Components of a Tree Style of Layout

The components that can be referenced in the `wt_index.html` template file are as follows:

- [Publication Title](#)
- [Publication Logo](#)
- [Search Input](#)

- [Print Link](#)
- [Main Page Menu](#)
- [Main Page Topic Tiles](#)
- [Main Page Table of Contents](#)
- [Index Terms Link](#)
- [Link to Skin Resources](#)

WebHelp Responsive Topic Template

The *Topic Template* is used to generate HTML pages for each topic in the WebHelp Responsive output. The name of the template file is `wt_topic.html` and it is located in the following directory:

```
DITA_OT_DIR/plugins/com.oxygenxml.webhelp/templates/dita/bootstrap/
```

The HTML pages produced for each topic consist of the topic content along with various other additional components, such as a title, menu, navigation breadcrumb, print icon, or side table of contents.

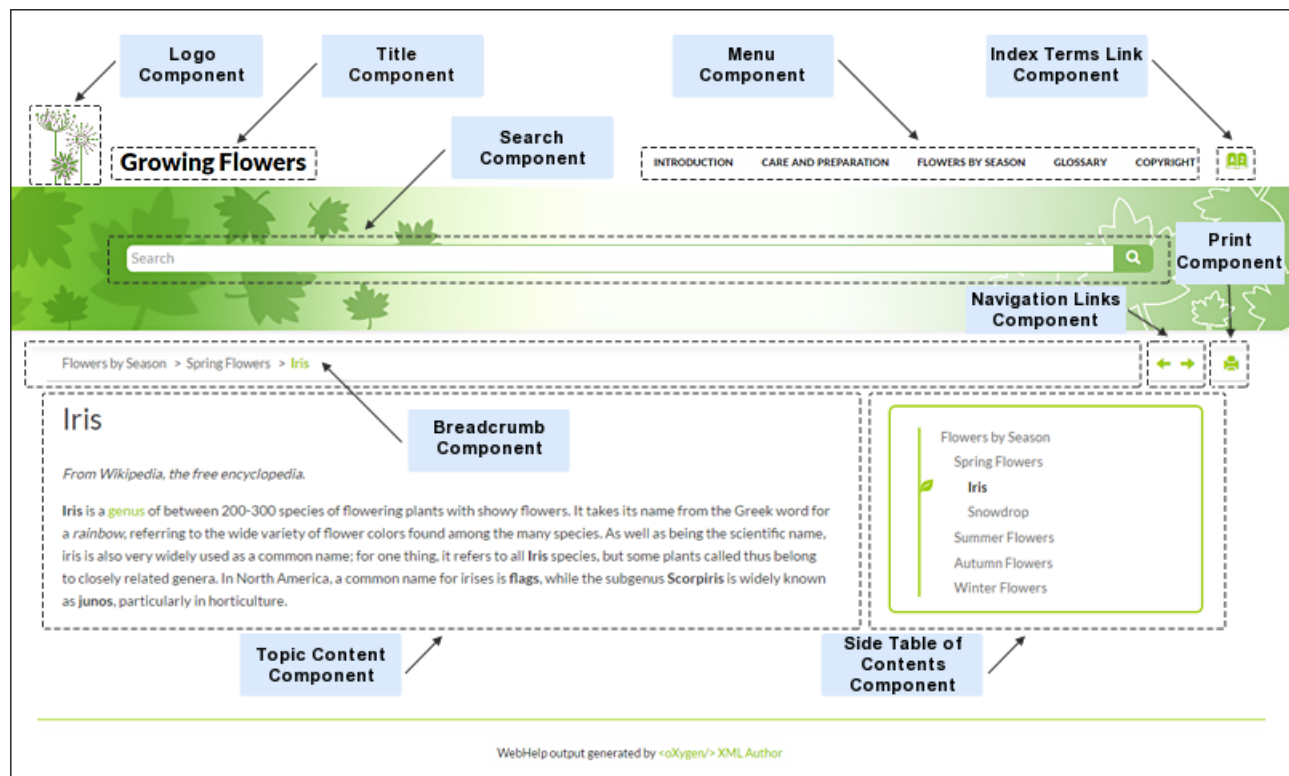


Figure 295: Example: Topic Page Components

The components that can be referenced in the `wt_topic.html` template file are as follows:

- [Publication Title](#)
- [Publication Logo](#)
- [Search Input](#)
- [Topic Breadcrumb](#)
- [Navigational Links](#)
- [Print Link](#)
- [Topic Content](#)
- [Topic Side TOC](#)
- [Topic Feedback](#)
- [Main Page Menu](#)
- [Index Terms Link](#)
- [Child Links](#)
- [Related Links](#)

- [Link to Skin Resources](#)

WebHelp Responsive Search Results Template

The *Search Results Template* is used to generate HTML pages that present search results in the WebHelp Responsive output. The name of the template file is `wt_search.html` and it is located in the following directory:

```
DITA_OT_DIR/plugins/com.oxygenxml.webhelp/templates/dita/bootstrap/
```

The HTML page that is produced consists of a search results component along with various other additional components, such as a title, menu, or index link.

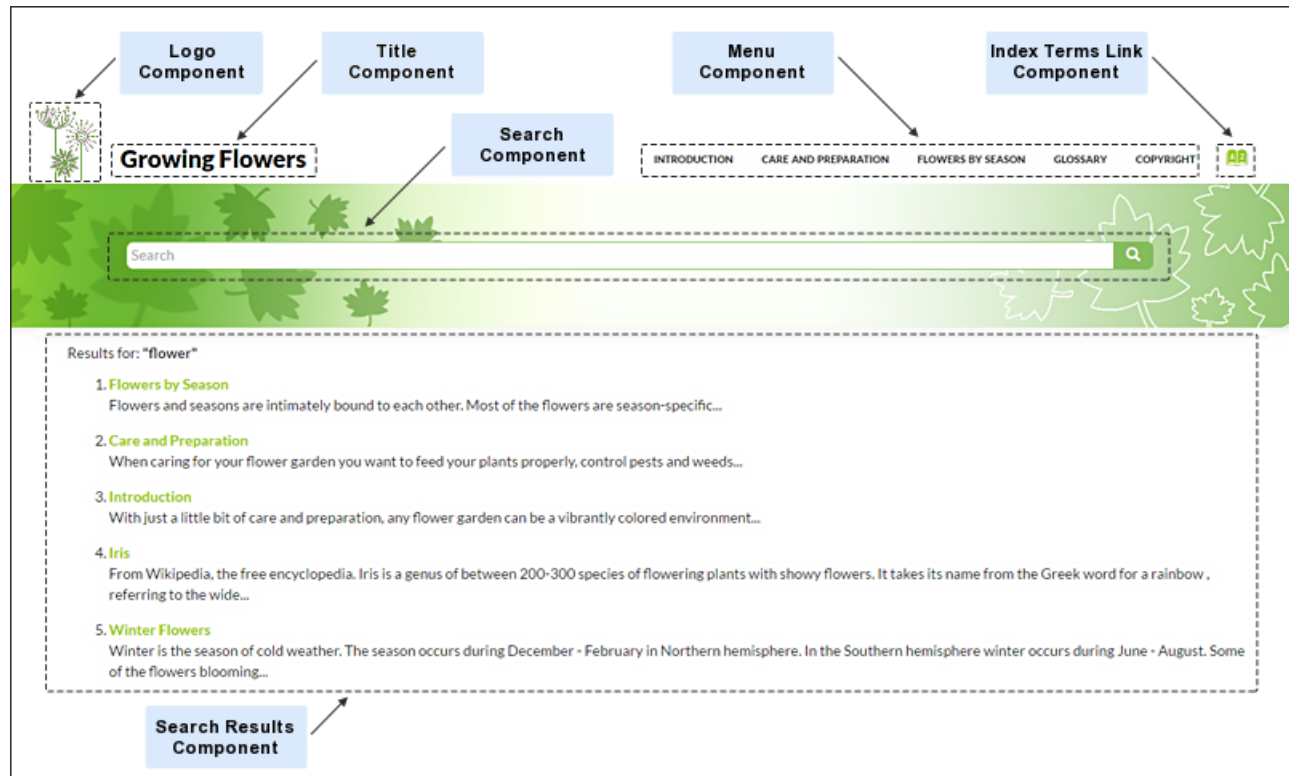


Figure 296: Example: Search Results Page Components

The components that can be referenced in the `wt_search.html` template file are as follows:

- [Publication Title](#)
- [Publication Logo](#)
- [Search Input](#)
- [Search Results](#)
- [Print Link](#)
- [Main Page Menu](#)
- [Index Terms Link](#)
- [Link to Skin Resources](#)

WebHelp Responsive Index Terms Template

The *Index Terms Template* is used to generate HTML pages that present index terms in the WebHelp Responsive output. The name of the template file is `wt_terms.html` and it is located in the following directory:

```
DITA_OT_DIR/plugins/com.oxygenxml.webhelp/templates/dita/bootstrap/
```

The HTML page that is produced consists of an index terms section along with various other additional components, such as a title, menu, or search field.

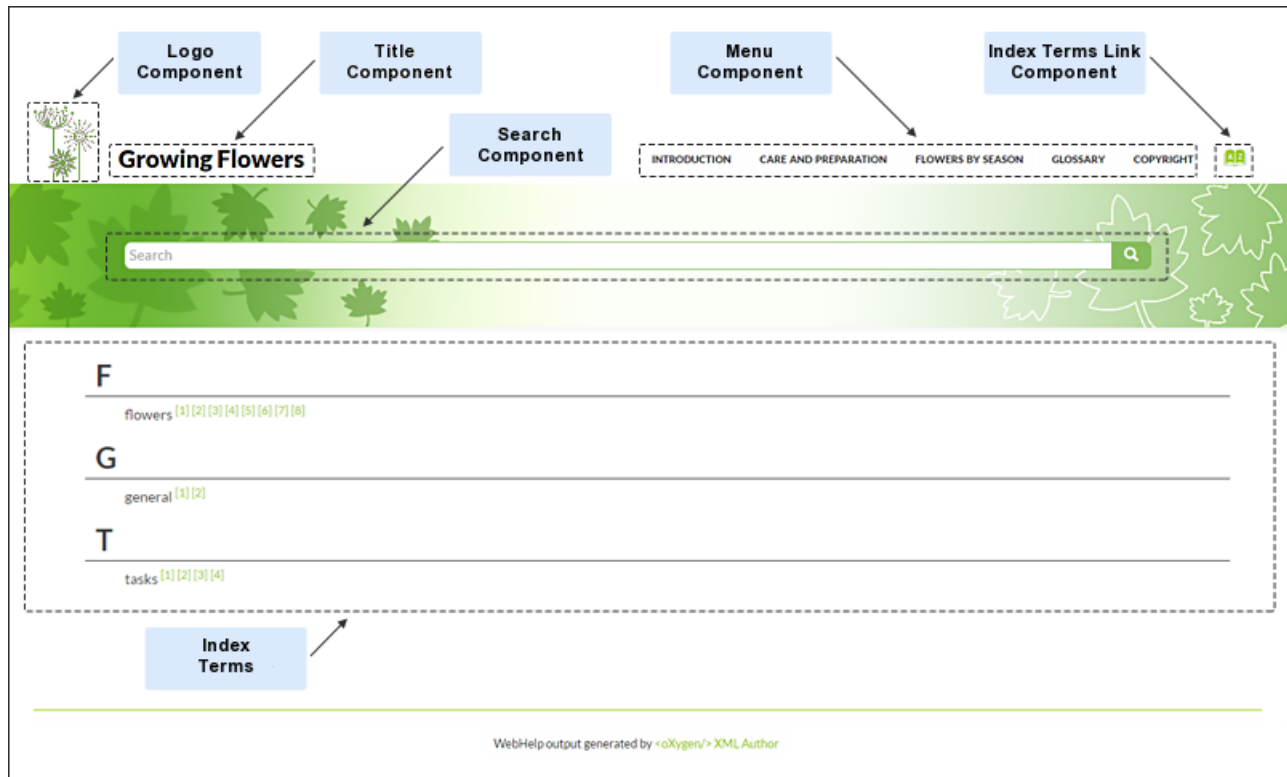


Figure 297: Example: Index Terms Page Components

The components that can be referenced in the `wt_terms.html` template file are as follows:

- [Publication Title](#)
- [Publication Logo](#)
- [Search Input](#)
- [Print Link](#)
- [Main Page Menu](#)
- [Index Terms Link](#)
- [Link to Skin Resources](#)

WebHelp Responsive Template Components

A WebHelp Responsive *template component* adds dynamics to a WebHelp *template page*. The rendering of a component depends on the context of where it is placed and its content depends on the transformed DITA Map.

Some of the template components can be used in all the [types of template pages](#), while some are only available for certain template pages. For instance, the *Publication Title* component can be used in all pages, but the *Navigation Breadcrumb* component can only be used in *Topic Template* pages.

To include a *template component* in the output of a particular type of *template page*, you have to reference a specific element in the particular [template file](#). All the elements associated with a template component should belong to the `http://www.oxygenxml.com/webhelp/components` namespace.

Every component could contain custom content or reference another component. To specify where the component content will be located in the output you can use the `component_content` element as a descendant of the component element.

Example:

```
<whc:webhelp_search_input class="navbar-form webhelp_main_page_search"
  role="form" >
  <whc:include_html href="{ $webhelp.include.before.search }"/>
  <div class="search_input">
    <whc:component_content/>
  </div>
</whc:webhelp_search_input>
```

The various components and where they can be used are as follows:

Publication Title [webhelp_publication_title]

This component generates the publication title in the output. To generate this component, the `webhelp_publication_title` element must be specified in the template file. It can be used in all four types of template pages:

- [Main Page Template \(wt_index.html file\)](#)
- [Topic Template \(wt_topic.html file\)](#)
- [Search Results Template \(wt_search.html file\)](#)
- [Index Terms Template \(wt_terms.html file\)](#)

In the template file, the `webhelp_publication_title` element can be specified as in the following example:

```
<whc:webhelp_publication_title
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_publication_title`.

Publication Logo [webhelp_logo]

This component generates a logo image in the output. To generate this component, the `webhelp_logo` element must be specified in the template file. It can be used in all four types of template pages:

- [Main Page Template \(wt_index.html file\)](#)
- [Topic Template \(wt_topic.html file\)](#)
- [Search Results Template \(wt_search.html file\)](#)
- [Index Terms Template \(wt_terms.html file\)](#)

In the template file, the `webhelp_logo` element can be specified as in the following example:

```
<whc:webhelp_logo
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In addition, you must also specify the path of the logo image in the `webhelp_logo.image` transformation parameter (in the **Parameters** tab in the transformation scenario). You can set the `webhelp_logo.image.target.url` parameter to generate a link to a URL when you click the logo image. If this parameter is not set, a link to the home page will automatically be generated.

In the output, you will find an element with the class: `wh_logo`.

Search Input [webhelp_search_input]

This component is used to generate the input widget associated with search function in the output. To generate this component, the `webhelp_search_input` element must be specified in the template file. It can be used in all four types of template pages:

- [Main Page Template \(wt_index.html file\)](#)
- [Topic Template \(wt_topic.html file\)](#)
- [Search Results Template \(wt_search.html file\)](#)
- [Index Terms Template \(wt_terms.html file\)](#)

In the template file, the `webhelp_search_input` element can be specified as in the following example:

```
<whc:webhelp_search_input
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `webhelp_search_input`.

Search Results [webhelp_search_results]

This component is used to generate a placeholder to signal where the search results will be presented in the output. To generate this component, the `webhelp_search_results` element must be specified in the template file. It can be used in the following type of template page:

- [Search Results Template \(wt_search.html file\)](#)

In the template file, the `webhelp_search_results` element can be specified as in the following example:

```
<whc:webhelp_search_results
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_search_results`.

Topic Breadcrumb [webhelp_breadcrumb]

This component generates a breadcrumb that displays the path of the current topic. To generate this component, the `webhelp_breadcrumb` element must be specified in the template file. It can be used in the following type of template page:

- [Topic Template \(wt_topic.html file\)](#)

In the template file, the `webhelp_breadcrumb` element can be specified as in the following example:

```
<whc:webhelp_breadcrumb
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_breadcrumb`. This element will contain a list with items that correspond to the topics in the path. The first item in the list has a link to the main page with the `home` class. The last item in the list corresponds to the current topic and has the `active` class set.

Navigational Links [webhelp_navigation_links]

This component generates navigation links to the next and previous topics. To generate this component, the `webhelp_navigation_links` element must be specified in the template file. It can be used in the following type of template page:

- [Topic Template \(wt_topic.html file\)](#)

In the template file, the `webhelp_navigation_links` element can be specified as in the following example:

```
<whc:webhelp_navigation_links
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_navigation_links`. This element will contain the links to the next and previous topics.

Topic Content [webhelp_topic_content]

This component generates the content of a topic and it represent the content of the HTML files as they are produced by the DITA-OT processor. To generate this component, the `webhelp_topic_content` element must be specified in the template file. It can be used in the following type of template page:

- [Topic Template \(wt_topic.html file\)](#)

In the template file, the `webhelp_topic_content` element can be specified as in the following example:

```
<whc:webhelp_topic_content
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_topic_content`.

Topic Side TOC [webhelp_side_toc]

This component generates a mini table of contents for the current topic. It will contain links to the children of current topic, its siblings, and all of its ancestors. To generate this component, the `webhelp_side_toc` element must be specified in the template file. It can be used in the following type of template page:

- [Topic Template \(wt_topic.html file\)](#)

In the template file, the `webhelp_side_toc` element can be specified as in the following example:

```
<whc:webhelp_side_toc
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```


In the output, you will find an element with the class: `wh_side_toc`. This element will contain links to the topics that are close to the current topic.

Topic Feedback [webhelp_feedback]

This component generates a placeholder for where the comments section will be presented. To generate this component, the `webhelp_feedback` element must be specified in the template file. It can be used in the following type of template page:

- [Topic Template \(wt_topic.html file\)](#)

In the template file, the `webhelp_feedback` element can be specified as in the following example:

```
<whc:webhelp_feedback
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

Child Links [webhelp_child_links]

For all topics with subtopics (child topics), this component generates a list of links to each child topic. To generate this component, the `webhelp_child_links` element must be specified in the template file. It can be used in the following type of template page:

- [Topic Template \(wt_topic.html file\)](#)

In the template file, the `webhelp_child_links` element can be specified as in the following example:

```
<whc:webhelp_child_links
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

Related Links [webhelp_related_links]

For all topics that contain related links, this component generates a list of related links that will appear in the output. To generate this component, the `webhelp_related_links` element must be specified in the template file. It can be used in the following type of template page:

- [Topic Template \(wt_topic.html file\)](#)

In the template file, the `webhelp_related_links` element can be specified as in the following example:

```
<whc:webhelp_related_links
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

Main Page Topic Tiles [webhelp_tiles]

This component generates the tiles section in the main page. This section will contain a tile for each root topic of the published documentation. Each topic tile has three sections that corresponds to the topic title, short description, and image. To generate this component, the `webhelp_tiles` element must be specified in the template file. It can be used in the following type of template page:

- [Main Page Template \(wt_index.html file\)](#)

In the template file, the `webhelp_tiles` element can be specified as in the following example:

```
<whc:webhelp_tiles
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_tiles`.

If you want to control the HTML structure that is generated for a WebHelp tile you can also specify the template for a tile by using the `whc:webhelp_tile` component, as in the following example:

```
<whc:webhelp_tile class="col-md-4">
  <!-- Place holder for tile's image -->
  <whc:webhelp_tile_image/>

  <div class="webhelp_tile_text">
    <!-- Place holder for tile's title -->
    <whc:webhelp_tile_title/>

    <!-- Place holder for tile's shordesc -->
```



```
<whc:webhelp_tile_shortdesc/>
</div>
</whc:webhelp_tile>
```

For information about customizing the tiles, see [Configuring the Tiles on the Main Page](#) on page 585.

Main Page Table of Contents [webhelp_main_page_toc]

This component generates a simplified Table of Contents. It is simplified because it contains only two levels from the documentation hierarchy. To generate this component, the `webhelp_main_page_toc` element must be specified in the template file. It can be used in the following type of template page:

- [Main Page Template \(wt_index.html file\)](#)

In the template file, the `webhelp_main_page_toc` element can be specified as in the following example:

```
<whc:webhelp_main_page_toc
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_main_page_toc`.

Main Page Menu [webhelp_topics_menu]

This component generates a menu with all the documentation topics. To generate this component, the `webhelp_topics_menu` element must be specified in the template file. It can be used in the following type of template page:

- [Main Page Template \(wt_index.html file\)](#)

In the template file, the `webhelp_topics_menu` element can be specified as in the following example:

```
<whc:webhelp_topics_menu
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_topics_menu`.

You can control the maximum level of topics that will be included in the menu using the `webhelp.top.menu.depth` transformation parameter (in the **Parameters** tab of the transformation scenario).

For information about customizing the menu, see [Customizing the Menu](#) on page 587.

Print Link [webhelp_print_link]

This component is used to generate a print icon that opens the print dialog box for your particular browser. To generate this component, the `webhelp_print_link` element must be specified in the template file. It can be used in all four types of template pages:

- [Main Page Template \(wt_index.html file\)](#)
- [Topic Template \(wt_topic.html file\)](#)
- [Search Results Template \(wt_search.html file\)](#)
- [Index Terms Template \(wt_terms.html file\)](#)

In the template file, the `webhelp_print_link` element can be specified as in the following example:

```
<whc:webhelp_print_link
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_print_link`.

Include HTML files [webhelp_include_html]

This component can be used to include custom HTML files. To generate this component, the `include_html` element must be specified in the template file. It can be used in all four types of template pages:

- [Main Page Template \(wt_index.html file\)](#)
- [Topic Template \(wt_topic.html file\)](#)
- [Search Results Template \(wt_search.html file\)](#)

- [Index Terms Template \(wt_terms.html file\)](#)

In the template file, the `include_html` element can be specified as in the following example:

```
<whc:include_html href="{wh.param}"/>
```

Where the `href` can have the following values:

- **any URL** - In this case, the file to be included is specified as an URL.
- **{oxygen-webhelp-template-dir}/file_to_include.html** - To include resources that are part of the template.
- **{webhelp.param}** - To include a resource whose path is specified through a WebHelp transformation scenario parameter. The value of this parameter can be a simple HTML fragment, in which case it will be copied to the output.

Index Terms Link [webhelp_indexterms_link]

This component can be used to generate a link to the index terms page (`indexterms.html`). If the published documentation does not contains any index terms, then the link will not be generated. To generate this component, the `webhelp_indexterms_link` element must be specified in the template file. It can be used in all four types of template pages:

- [Main Page Template \(wt_index.html file\)](#)
- [Topic Template \(wt_topic.html file\)](#)
- [Search Results Template \(wt_search.html file\)](#)
- [Index Terms Template \(wt_terms.html file\)](#)

In the template file, the `webhelp_indexterms_link` element can be specified as in the following example:

```
<whc:webhelp_indexterms_link
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_indexterms_link`. This element will contain a link to the `indexterms.html` page.

Link to Skin Resources [webhelp_skin_resources]

This component can be used to add a link to resources for the current WebHelp skin (such as the CSS file). To generate this component, the `webhelp_skin_resources` element must be specified in the template file. It can be used in all four types of template pages:

- [Main Page Template \(wt_index.html file\)](#)
- [Topic Template \(wt_topic.html file\)](#)
- [Search Results Template \(wt_search.html file\)](#)
- [Index Terms Template \(wt_terms.html file\)](#)

In the template file, the `webhelp_skin_resources` element can be specified as in the following example:

```
<whc:webhelp_skin_resources/>
```

In the output, you will find a link to the skin resources.

Customizing the WebHelp Responsive Output

To change the overall appearance of your WebHelp Responsive output, you can use several different customization methods or a combination of methods. If you are familiar with CSS and coding, you can style your WebHelp output through your own custom stylesheets. You can also customize your output by modifying existing templates, create your own, or by configuring certain options and parameters in the transformation scenario.

This section includes topics that explain various ways to customize your WebHelp Responsive system output, such as how to configure the tiles on the main page, add logos in the title area, integrate with social media, localizing the interface, and much more.

WebHelp Responsive Customization Methods

There are several methods that you can use to customize your WebHelp Responsive output. This topic provides information on each of the methods and highlights its advantages so that you can choose the best possible method based upon your needs.

Insert Custom XHTML Fragments in Predefined Placeholders

The WebHelp Responsive template contains a series of component placeholders. Some of these placeholders are left empty in the default output configurations, but you can use them to display custom content. This method is recommended if you want to use an existing skin and simply make some minor changes or additions in certain locations within the final output.

Advantages:

- This method is very easy, since the fragments for the placeholders can be specified in the transformation scenario.
- Advanced knowledge of CSS styling is not required for this method.

Each such placeholder has an associated parameter in the transformation scenario **Parameters** tab. These predefined empty placeholder parameters are illustrated and described below:

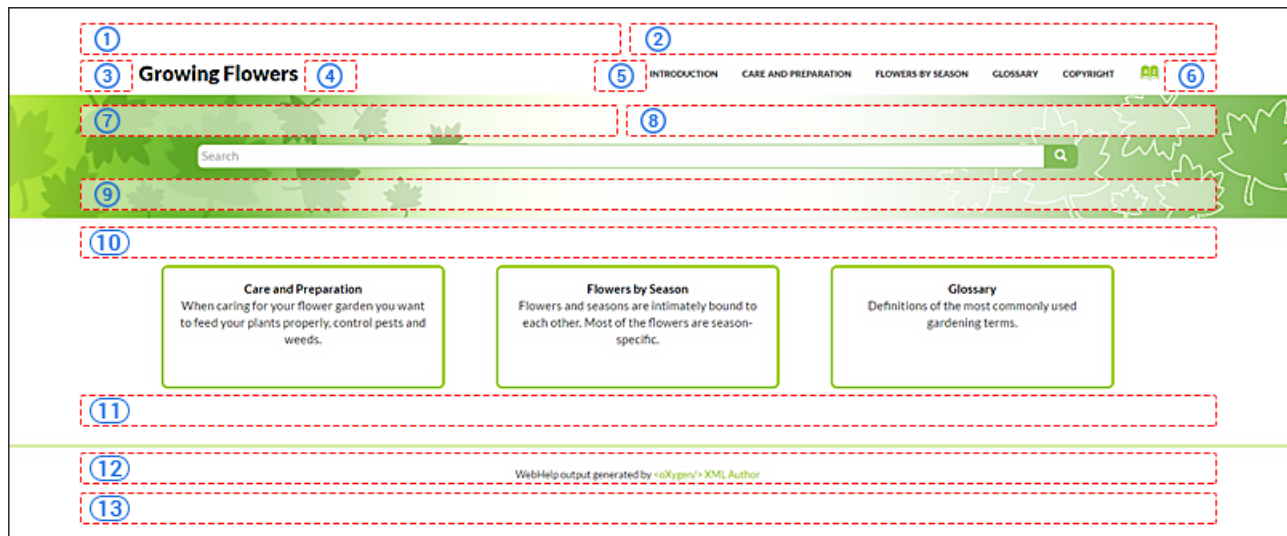


Figure 298: Predefined Placeholders Diagram

Each of these parameters can hold either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment:

1- webhelp.fragment.head

In the generated output it displays a given XHTML fragment as a page header. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

2- webhelp.fragment.before.body

In the generated output it displays a given XHTML fragment before the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

3- webhelp.fragment.before.logo_and_title

In the generated output it displays a given XHTML fragment before the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

4- webhelp.fragment.after.logo_and_title

In the generated output it displays a given XHTML fragment after the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

5- webhelp.fragment.before.top_menu

In the generated output it displays a given XHTML fragment before the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

6- webhelp.fragment.after.top_menu

In the generated output it displays a given XHTML fragment after the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

7- webhelp.fragment.before.main.page.search

In the generated output it displays a given XHTML fragment before the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

8- webhelp.fragment.welcome

In the generated output it displays a given XHTML fragment as a welcome message (or title). The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

9- webhelp.fragment.after.main.page.search

In the generated output it displays a given XHTML fragment after the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

10- webhelp.fragment.before.toc_or_tiles

In the generated output it displays a given XHTML fragment before the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

11- webhelp.fragment.after.toc_or_tiles

In the generated output it displays a given XHTML fragment after the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

12- webhelp.fragment.footer

In the generated output it displays a given XHTML fragment as the page footer. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

13- webhelp.fragment.after.body

In the generated output it displays a given XHTML fragment after the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

EXAMPLE:

To insert a message above the search field component in the main page of the output, follow this procedure:

1. Edit the *WebHelp Responsive* transformation scenario.
2. Go to **Parameters** tab and find the parameter associated with the place holder that you want to use. In this case, it is called `webhelp.fragment.welcome`.
3. Edit the parameter. Depending on the size of the content you want to add, you can insert one of the following:
 - A small well-formed XHTML fragment, such as: `<i>Welcome to our user guide</i>`.
 - A path to a file that contains well-formed XHTML content.

Customize WebHelp Output with a Custom CSS

This method is recommended if you just want to adapt an existing skin (that is very close to what you need) and only need to change the styling of the final output. For example, this might be the case if you simply want to change a color, or adjust some of the margins or paddings of certain components.

Advantages:

- This method could be used as a quick and easy way to make small styling changes.
- The custom CSS can be distributed with your project and shared with other members of your team.
- This method can be used for advanced and precise styling.

Additional Notes:

- The fonts, images, and other resources must be stored in a remote server location.
- This type of customization will not appear in the **Templates** tab of the transformation scenario. Instead, the custom CSS needs to be set as a parameter of an existing transformation scenario.

To set a custom CSS to a transformation scenario:

1. Edit the *WebHelp Responsive* transformation scenario.
2. Open the **Parameters** tab.
3. 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. Also, if your customization CSS requires additional resources, you can copy them to the generated output by specifying the [webhelp.custom.resources parameter](#).

If you are using DITA, you could also override the default XSLT templates that are used for WebHelp transformations by using some extension points. For information about this method, see [Overriding the XSLT Processing Step of a DITA WebHelp Transformation](#) on page 594.

Create or Customize a WebHelp Responsive Skin

This method is recommended if you want a design that is not similar to any of the predefined skins, or if you want to make a lot of changes to one of the existing skins. This method is also useful if you want to distribute additional resources (such as fonts and images) together with a custom CSS.

Advantages:

- The customized skin will be available in the **Templates** tab of the transformation scenario.
- The resources are encapsulated into the `skin` directory and can be shared with other team members, along with a custom CSS file.

Additional Notes:

- This method requires access to the installation folder, or the use of an external *DITA-OT* engine (with the [WebHelp plugin](#) installed).

To create a new WebHelp Responsive skin, follow this procedure:

1. Locate the following folder in your DITA-OT directory (*DITA_OT_DIR*):

```
DITA_OT_DIR/plugins/com.oxygenxml.webhelp/templates/dita/bootstrap/variants/
```

2. Here you can see some subdirectories corresponding to different variants for the same template. For instance, the default directories are `tiles` and `tree`.
3. In each of these variants, you will find a directory for each of the skins (for example, the default skins and their corresponding directories are: *flowers*, *green*, *light*, *mechano*, *orange*, etc.)
4. Duplicate one of the skin folders and rename it to whatever you want your new skin to be identified as.
5. Edit the `skin.css` file and customize it the way you want. If your customization of the CSS file requires additional resources (such as images, fonts, or other CSS files), they need to be placed in the `resources` folder at the same level with the `skin.css` file.

Result: Your new skin should now be included in the list of skins in the **Templates** tab of the transformation scenario.



Tip: During development, you may want to regularly test your customization. To shorten the publishing time of your tests, use a small set of test files (you could use one of the Oxygen XML Developer plugin sample projects). Also, you can use your web browser CSS inspector tool to lookup the CSS classes you want to modify.

Create a New WebHelp Responsive Template

This method can be used when you need to make significant structural changes to the WebHelp output. For example, if you want to move some components to other positions, or if you want to use a different responsive front-end framework than the default [Bootstrap framework](#) (for instance, if you want to switch to [ZURB Foundation](#)).

Advantages:

- This method allows you to fully customize the output.
- This method allows you to [change the structure of the generated HTML files](#).
- You can [create your own skins](#) for the new template.

Additional Notes:

- This method requires access to the installation folder, or the use of an external *DITA-OT* engine (with the [WebHelp plugin](#) installed).

To create a new WebHelp Responsive template, follow these steps:

1. Locate the following folder in your DITA-OT directory (*DITA_OT_DIR*):

```
DITA_OT_DIR/plugins/com.oxygenxml.webhelp/templates/dita/
```

2. Duplicate the `bootstrap` folder and rename it to whatever you want your new template to be identified as (for example, `myTemplate`).
3. Customize the structure of the new template according to your needs. For example, if you only want to keep one of the template variants, open the `myTemplate/variants` folder and delete all of its subdirectories, except for that one (for instance, the `tiles` directory). Keep in mind that the structure of the template directory is important. The names of folders at certain levels correspond to the names of templates and skins, while components and resources are defined and referenced in certain files or folders at specific locations within the directory structure. For more information, see [WebHelp Responsive Template Directory Structure](#) on page 568.
4. You can also customize the structure of the skins within the template variants. For example, if you only want to keep one of the skins in the `tiles` variant, open the `myTemplate/variants/tiles` folder and delete all of its subdirectory skins, except for that one (for instance, the `light` directory).
5. Edit the `skin.css` file that is located in the skin directory (for example, `myTemplate/variants/tiles/light`) and customize it the way you want. If your customization of the CSS file requires additional resources (such as images, fonts, or other CSS files), they need to be placed in the `resources` folder at the same level with the `skin.css` file.

Result: Your new templates and skins should now be included in the **Templates** tab of the transformation scenario.



Tip: During development you regularly need to test your customization. To shorten the publishing time of your test, use a small project (you could use one of the Oxygen XML Developer plugin sample projects). Also, you can use your web browser CSS inspector tool to lookup the CSS classes you want to modify.

For more information about WebHelp Responsive templates, see the [WebHelp Responsive Template Mechanism](#) on page 567 section.

Copying Additional Resources to WebHelp Output Directory

To copy additional resources (such as JavaScript, CSS or other resources) to the output directory of a WebHelp system, follow these steps:

1. Place all your resources in the same directory.
2. Edit the WebHelp transformation scenario.
3. Go to the **Parameters** tab.
4. Edit the value for the `webhelp.custom.resources` parameter and set it to the absolute path of the directory in step 1.
5. Click **OK** to save the changes to the transformation scenario.

All files from the new directory will be copied to the root of the WebHelp output directory.

Adding Custom HTML Content in WebHelp Responsive Output

You can add custom HTML content in the WebHelp Responsive output by inserting it in a well-formed XML file that will be referenced in the transformation scenario. This content may include references to additional JavaScript, CSS, and other types of resources, or such resources can be inserted inline within the HTML content that is inserted in the XML file.

To include custom HTML content in the WebHelp Responsive output files, follow these steps:

1. Insert the HTML content in a well-formed XML file. There are several things to consider in regards to this XML file:
 - a. **Well-Formedness** - If the file is not a [Well-formed XML document](#) (or fragments are not well-formed), the transformation will fail.

A common use case is if you want to include several `script` or `link` elements. In this case, the XML fragment has multiple root elements and to make it well-formed, you can wrap it in an `html` element. This element tag will be filtered out and only its children will be copied to the output documents. Similarly, you can wrap your content in `head`, `body`, `html/head`, or `html/body` elements.

- b. **Referencing Resources in the XML File** - You can include references to local resources (such as JavaScript or CSS files) by using the predefined `${oxygen-webhelp-output-dir}` macro to specify their paths relative to the output directory:

```
<html>
  <script type="text/javascript" src="${oxygen-webhelp-output-dir}/js/test.js"/>
  <link rel="stylesheet" type="text/css"
    href="${oxygen-webhelp-output-dir}/css/test.css" />
</html>
```

If you want that the path of your resource to be relative to the [templates directory](#), you can use the `${oxygen-webhelp-template-dir}` macro.

To copy the referenced resources to the output directory, follow the procedure in: [Copying Additional Resources to WebHelp Output Directory](#) on page 582.

- c. **Inline JavaScript or CSS Content** - If you want to include inline JavaScript or CSS content in the XML file, it is important to place this content inside an XML comment, as in the following examples:

JavaScript:

```
<script type="text/javascript">
  <!--
    /* Include JavaScript code here. */

    function myFunction() {
      return true;
    }
  -->
</script>
```

CSS:

```
<style>
  <!--
    /* Include CSS style rules here. */

    *{
      color:red
    }
  -->
</style>
```

2. Edit the WebHelp Responsive transformation scenario.
3. Go to the **Parameters** tab.
4. Edit the value of the [webhelp.fragment.head parameter](#) and set it to the absolute path of the XML file created in step 1. Your additional content will be included at the end of the `head` element of your output document.



Note: If you want to insert the content in another location within the output document, you can reference the XML file in any of the other parameters listed in [Insert Custom XHTML Fragments in Predefined Placeholders](#) on page 579.

5. Click **OK** to save the changes to the transformation scenario.

Related tasks

[Copying Additional Resources to WebHelp Output Directory](#) on page 582

Related information

[WebHelp Responsive Template Directory Structure](#) on page 568

[WebHelp Responsive Customization Methods](#) on page 578

Adding a Favicon in WebHelp Systems

You can add a custom *favicon* to your WebHelp system by simply using a parameter in the transformation scenario to point to your *favicon* image. This is available for DITA and DocBook WebHelp systems using **WebHelp Responsive**, **WebHelp Responsive with Feedback**, **WebHelp Classic**, **WebHelp Classic with Feedback**, or **WebHelp Classic Mobile** transformation scenarios.

To add a *favicon*, follow these steps:

1. [Edit the WebHelp transformation scenario](#) and open the **Parameters** tab.
2. Locate the `webhelp.favicon` parameter and enter the file path that points to the image that will be use as the *favicon*.
3. Run the transformation scenario.

Adding a Logo Image in the Title Area

To add a logo in the title area of your WebHelp output, follow this procedure:

1. Edit a WebHelp transformation scenario, then open the **Parameters** tab.
2. Specify the path to your logo in the `webhelp.logo.image` parameter.

3. If you also want to add a link to your website when you click the logo image, set the URL in the `webhelp.logo.image.target.url` parameter.
4. Run the transformation scenario.

Adding a Welcome Message in the Home Page

The main page of the WebHelp Responsive output contains a set of empty placeholders that can be used to display customized text fragments. These placeholders are available to you through [WebHelp Responsive transformation scenario parameters](#). One of these placeholders (identified through the `webhelp.fragment.welcome` parameter) was designed to display text content above the search box in the main page.

To add a customized welcome message in the main page of the WebHelp Responsive output, follow this procedure:

1. Edit a WebHelp Responsive transformation scenario.
2. Open the [Templates tab](#) and choose a *skin*.
3. Open the [Parameters tab](#) and edit the `webhelp.fragment.welcome` parameter. The value of this parameter can be one of the following:
 - A small well-formed XHTML fragment (such as: `<i>Welcome to the User Guide</i>`).
 - Path to a file that contains well-formed XHTML content.
4. Click **OK**, then click the **Apply associated** button to execute the transformation scenario.

Adding Video and Audio Objects in DITA WebHelp Output

You can insert references to video and audio media resources (such as videos, audio clips, or embedded HTML frames) in your DITA topics and then publish them to WebHelp output. The media objects can be played directly in all HTML5-based outputs, including WebHelp systems.

To add media objects in the WebHelp output generated from DITA documents, follow the procedures below.

Adding Videos to DITA WebHelp Output

1. Edit the DITA topic and insert a reference to the video by manually adding an `object` element, as in one of the following examples:

```
<object outputclass="video" type="video/mp4" data="MyVideo.mp4"/>
```

or, instead of the `data` attribute, you can specify the video using a parameter like this:

```
<object outputclass="video">
  <param name="src" value="videos/MyVideo.mp4"/>
</object>
```

2. Apply a *DITA to WebHelp* transformation scenario to obtain the output.

Result: The transformation converts the `object` element to an HTML5 `video` element.

```
<video controls="controls"><source type="video/mp4" src="MyVideo.mp4"></source>
</video>
```

Adding Audio Clips to DITA WebHelp Output

1. Edit the DITA topic and insert a reference to the audio clip by manually adding an `object` element, as in one of the following examples:

```
<object outputclass="audio" type="audio/mpeg" data="MyClip.mp3"/>
```

or, instead of the `data` attribute, you can specify the audio clip using a parameter like this:

```
<object outputclass="audio">
  <param name="src" value="audio/MyClip.mp3"/>
</object>
```

2. Apply a *DITA to WebHelp* transformation scenario to obtain the output.

Result: The transformation converts the `object` element to an HTML5 `audio` element.

```
<audio controls="controls"><source type="audio/mpeg" src="MyClip.mp3"></source>
</audio>
```

Adding Embedded HTML Frames (such as YouTube videos) to DITA WebHelp Output

1. Edit the DITA topic and insert a reference to the embedded object by manually adding an `object` element, as in one of the following examples:

```
<object outputclass="iframe" data="https://www.youtube.com/embed/m_vv2s5Trn4"/>
```

or, instead of the `data` attribute, you can specify the object using a parameter like this:

```
<object outputclass="iframe">
  <param name="src" value="http://www.youtube.com/embed/m_vv2s5Trn4"/>
</object>
```

2. Apply a *DITA to WebHelp* transformation scenario to obtain the output.

Result: The transformation converts the `object` element to an HTML5 `iframe` element.

```
<iframe controls="controls" src="https://www.youtube.com/embed/m_vv2s5Trn4">
</iframe>
```

Configuring the Tiles on the Main Page

The *tiles* version of the main page of the WebHelp Responsive output displays a tile for each topic found on the first level of the DITA map source. However, you might want to customize the way they look or even to hide some of them.

Depending on your particular setup, you can choose to customize these tiles either by setting metadata information in the DITA map or by customizing the CSS that is associated with the DITA map.

How to Hide Some of the Tiles

If your documentation is very large or there is a large number of topics on the first level, you might want to hide some of the tiles. Also, this might be useful if you only want to display the topics in the first page that are most relevant to your intended audience.

There are two methods for doing this. One of them involves editing the DITA map and marking the topics that do not need to be displayed as tiles, and another one that uses a small CSS customization level to hide some tiles identified by the id of the topic.

Editing the DITA Map

To edit the metadata in the DITA map to control which topics on the first level of the DITA map will not be displayed as a tile, follow these steps:

1. Open the DITA map in the **Text** editing mode of Oxygen XML Developer plugin.
2. Add the following metadata information in the `topicref` element (or any of its specializations) for each first-level topic that you do not want to be displayed as a tile:

```
<topicmeta>
  <data name="wh-tile">
    <data name="hide" value="yes"/>
  </data>
</topicmeta>
```

Customizing the CSS

To customize the CSS to control which topics on the first level of the DITA map will not be displayed as a tile, follow these steps:

1. Make sure you set an id on the topic you want to hide.

2. Create a new CSS file that contains a rule that hides the tile generated for the topic (identified in the following example by the topic id `growing-flowers`). The CSS file should have content that is similar to this:

```
.wh_tile [data-id='growing-flowers'] {  
  display:none;  
}
```

3. Use the [Customizing WebHelp Output with a Custom CSS](#) method to pass the CSS file you just created to the transformation scenario.

How to Add an Image to the Tiles

There are two methods that you can use to add an image to a tile. One of them involves editing the DITA map, and the other uses a CSS customization.

Editing the DITA Map

To edit the metadata in the DITA map to set an image to be displayed in a tile, follow these steps:

1. Open the DITA map in the **Text** editing mode of Oxygen XML Developer plugin.
2. Add the following metadata information in the `topicref` element (or any of its specializations) for each first-level topic that will have an image displayed in the corresponding tile:

```
<topicmeta>  
  <data name="wh-tile">  
    <data name="image" href="img/tile-image.png" format="png">  
      <data name="attr-width" value="64"/>  
      <data name="attr-height" value="64"/>  
    </data>  
  </data>  
</topicmeta>
```



Note: The `attr-width` and `attr-height` attributes can be used to control the size of the image, but they are optional.

Customizing the CSS

To customize the CSS to set an image to be displayed in a tile, follow these steps:

1. Make sure you set an id on the topic that you want the tile include an image.
2. Create a new CSS file that contains a rule that associates an image with a specific tile. The CSS file should have content that is similar to this:

```
.wh_tile[data-id='growing-flowers']> div {  
  background-image:url('resources/flower.png');  
}
```

3. Use the [Customizing WebHelp Output with a Custom CSS](#) or the [Create a WebHelp Responsive Skin](#) method to pass the CSS file you just created to the transformation scenario.

Change Numbering Styles for Ordered Lists

Ordered lists (o l) 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.

CSS Styling to Customize WebHelp Output

One way to customize WebHelp output is to use custom CSS styling. This method can be used to make small, simple styling changes or more advanced, precise changes. To implement the styling in your WebHelp output, you simply need to create the custom CSS file and reference it in your transformation scenario.

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.

Customizing the Menu

By default, the menu component is displayed in all WebHelp Responsive pages. However, you might want to hide it completely, or only display some of its menu entries.

How to Hide Some of the Menu Entries

There are two methods for doing this. One of them involves editing the DITA map and marking the topics that do not need to be included in the menu, and another one that uses a small CSS customization.

Editing the DITA Map

To edit the metadata in the DITA map to control which topics will not be displayed in the menu, follow these steps:

1. Open the DITA map in the **Text** editing mode of Oxygen XML Developer plugin.
2. Add the following metadata information in the `topicref` element (or any of its specializations) for each topic you do not want to be displayed in the menu:

```
<topicmeta>
  <data name="wh-menu">
    <data name="hide" value="yes"/>
  </data>
</topicmeta>
```

Customizing the CSS

To customize the CSS to control which topics will not be displayed in the menu, follow these steps:

1. Make sure you set an id on the topic that you do not want to include in the menu.
2. Create a new CSS file that contains a rule that hides the menu entry generated for the topic (identified by the topic id `growing-flowers` in the following example). The CSS file should have content that is similar to this:

```
.wh_top_menu *[data-id='growing-flowers'] {
  display:none;
}
```

3. Use the [Customizing WebHelp Output with a Custom CSS](#) method to pass the CSS file you just created to the transformation scenario.

How to Hide the Entire Menu

If you do not want to include a main menu in the pages of the WebHelp Responsive output, you can instruct the transformation scenario to skip the menu generation completely. To do so, follow this procedure:

1. Edit a WebHelp Responsive transformation scenario.
2. Open the [Templates tab](#) and choose a *skin*.
3. Open the [Parameters tab](#) and set the value of the `webhelp.show.top.menu` parameter to `no`.
4. Click **OK**, then click the **Apply associated** button to execute the transformation scenario.

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 Developer 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_INSTALL_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.

Exclude Certain DITA Topics from WebHelp Search Results

The WebHelp **Search** engine does not index DITA topics that have the `@search` attribute set to `no`. This is useful if you have topics in your DITA map structure that you do not want to be included in search results for your WebHelp system.

To exclude DITA topics from WebHelp search results, follow these steps:

1. Edit the DITA map and for any `topicref` that you want to exclude from search results, set the `search` attribute to `no`.
For example:

```
<topicref href="../../../topics/internal-topic1.dita" search="no"/>
```

2. Save your changes to the DITA map.
3. Re-run your WebHelp system transformation scenario.

Flag DITA Content in WebHelp Output

Flagging content in WebHelp output 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 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.

Integrating Social Media and Google Tools in WebHelp Output

Oxygen XML Developer 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 Responsive 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 Developer plugin, click the  **Configure Transformation Scenario(s)** action from the toolbar.
5. Select an existing WebHelp Responsive transformation scenario (depending on your needs, it may be with or without feedback) and click the **Duplicate** button to open the **Edit Scenario** dialog box.
6. Switch to the **Parameters** tab. Depending on where you want to display the button, edit *one of the parameters that begin with webhelp. fragment*. Set that parameter to reference the `facebook-widget.xml` file that you created earlier.
7. Click **Ok**.
8. Run the transformation scenario.

How to Add a Google+ Button in WebHelp Responsive Output

To add a Google+ widget to your WebHelp Responsive 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 Developer plugin, click the  **Configure Transformation Scenario(s)** action from the toolbar.
5. Select an existing WebHelp Responsive transformation scenario (depending on your needs, it may be with or without feedback) and click the **Duplicate** button to open the **Edit Scenario** dialog box.
6. Switch to the **Parameters** tab. Depending on where you want to display the button, edit *one of the parameters that begin with `webhelp.fragment`*. Set that parameter to reference the `google-plus-button.xml` file that you created earlier.
7. Click **Ok**.
8. Run the transformation scenario.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Add Tweet Button in WebHelp Responsive Output

To add a Twitter™ *Tweet* widget to your WebHelp Responsive 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 Developer plugin, click the  **Configure Transformation Scenario(s)** action from the toolbar.

5. Select an existing WebHelp Responsive transformation scenario (depending on your needs, it may be with or without feedback) and click the **Duplicate** button to open the **Edit Scenario** dialog box.
6. Switch to the **Parameters** tab. Depending on where you want to display the button, edit *one of the parameters that begin with webhelp.fragment*. Set that parameter to reference the `tweet-button.xml` file that you created earlier.
7. Click **Ok**.
8. Run the transformation scenario.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Integrate Google Analytics in WebHelp Responsive Output

To enable your WebHelp Responsive 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-XXXXXXX-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 Developer plugin, click the  **Configure Transformation Scenario(s)** action from the toolbar.
6. Select an existing WebHelp Responsive transformation scenario (depending on your needs, it may be with or without feedback) and click the **Duplicate** button to open the **Edit Scenario** dialog box.
7. Switch to the **Parameters** tab. Depending on where you want to display the button, edit *one of the parameters that begin with webhelp.fragment*. Set that parameter to reference the `googleAnalytics.html` file that you created earlier.
8. Click **Ok**.
9. Run the transformation scenario.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Integrate Google Search in WebHelp Responsive Output

You can integrate Google Search into your WebHelp Responsive 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 Developer plugin, click the  **Configure Transformation Scenario(s)** action from the toolbar.
6. Select an existing WebHelp Responsive transformation scenario (depending on your needs, it may be with or without feedback) 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 with the following content: `<div class="gcse-searchresults-only" data-autoSearchOnLoad="true" data-queryParameterName="searchQuery"></div>` (you must use the [HTML5 version for the GCSE](#)). For more information about other supported attributes, see [Google Custom Search: Supported Attributes](#).
 - b) Set the value of the `webhelp.google.search.results` parameter to the file path of the file you just created. If this parameter is not specified, the following code is used: `<div class="gcse-searchresults-only" data-autoSearchOnLoad="true" data-queryParameterName="searchQuery"></div>`.
9. Click **Ok**.
10. Run the transformation scenario.

Related information

[Integrating Social Media and Google Tools in WebHelp Output](#) on page 589

Localizing the Email Notifications of WebHelp with Feedback Systems

The feedback-enabled WebHelp systems use 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 WebHelp Classic with Feedback

1. Create the following directory:
`[OXYGEN_INSTALL_DIR]\frameworks\docbook\xs\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl`
2. Copy all English template files from
`[OXYGEN_INSTALL_DIR]\frameworks\docbook\xs\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_INSTALL_DIR]\frameworks\docbook\xs\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl`
 directory and translate the content into Dutch.
4. Start Oxygen XML Developer plugin and edit the **DocBook WebHelp Classic 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 WebHelp Classic with Feedback or WebHelp Responsive 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 Developer plugin and edit the **DITA Map WebHelp Classic with Feedback** or **DITA Map WebHelp Responsive 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.

Localizing 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. 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.
4. 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"/>`.
5. 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).
6. Run the transformation scenario to produce the WebHelp output.

Related information

[Searching Japanese Content in WebHelp Pages](#) on page 593

Searching Japanese Content in WebHelp Pages

To optimize the indexing of Japanese content in WebHelp pages, the Lucene Kuromoji Japanese analyzer can be used. This analyzer is included in the Oxygen XML Developer plugin installation kit.

Activating Japanese Indexing in DITA WebHelp Systems

To activate the Japanese indexing in your WebHelp system generated from DITA content, follow these steps:

1. Set the language for your content to Japanese with one of the following two methods:
 - Edit a **DITA to WebHelp** transformation scenario and in the [Parameters tab](#), set the value of the `args.default.language` parameter to `ja-jp`.
 - Set the `xml:lang` attribute on the root of the DITA map and the referenced topics to `ja-jp`.
2. For the analyzer to work properly, search terms that are entered into the WebHelp search text field must be separated by spaces.
3. Run the **DITA to WebHelp** transformation scenario to generate the output.

Optionally a Japanese user dictionary can be set with the [webhelp.search.japanese.dictionary](#) parameter.

Activating Japanese Indexing in DocBook WebHelp Systems

To activate the Japanese indexing in your WebHelp system generated from DocBook content, follow these steps:

1. Edit a **DocBook to WebHelp** transformation scenario and in the [Parameters](#) tab, set the value of the `l10n.gentext.default.language` parameter to `ja`.
2. Run the transformation scenario to generate the output.

Related tasks

[Localizing the Interface of WebHelp Output \(for DITA Map Transformations\)](#) on page 593

[Localizing the Interface of WebHelp Output \(for DocBook Transformations\)](#) on page 624

Overriding the XSLT Processing Step of a DITA WebHelp Transformation

There are several methods that you can use to customize your WebHelp output. Since WebHelp output is primarily obtained by running XSLT transformations over the DITA input files (through the **DITA Map WebHelp** transformation scenarios), one possibility would be to override the default XSLT templates that are used for WebHelp transformations by using some extension points.

XSLT-Import Extension Points

You can customize your WebHelp output by overriding the default XSLT templates that are used for each type of WebHelp transformation. This can be done by creating a DITA extension plugin that specifies one of the following extension points:

com.oxygenxml.webhelp.xsl.dita2webhelp

You can use this extension point to override a template of the XSLT stylesheet (`dita2webhelpImpl.xsl`) that produces an HTML file for each DITA topic. Depending on the type of WebHelp transformation you are currently using, the path to this stylesheet is as follows:

- **WebHelp Classic Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\desktop\dita2webhelpImpl.xsl`
- **WebHelp Classic Mobile Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\mobile\dita2webhelpImpl.xsl`
- **WebHelp Responsive Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\responsive\dita2webhelpImpl.xsl`

com.oxygenxml.webhelp.xsl.createMainFiles

You can use this extension point to override a template of the XSLT stylesheet (`createMainFilesImpl.xsl`) that produces the WebHelp main HTML page. Depending on the type of WebHelp transformation you are currently using, the path to this stylesheet is as follows:

- **WebHelp Classic Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\desktop\createMainFilesImpl.xsl`
- **WebHelp Classic Mobile Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\mobile\createMainFilesImpl.xsl`
- **WebHelp Responsive Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\responsive\createMainFilesImpl.xsl`



Attention: The customizations you made by using this extension point will affect all WebHelp transformations. If you want to have a customization that is only available for a certain plugin, please use the [Overriding the XSLT Stylesheet from a Customized ANT Build File](#) method.

XSLT-Parameter Extension Points

In case your customization stylesheet declares one or more XSLT parameters and you want to control their value from the transformation scenario, you can use one of the following XSLT parameter extension points:

com.oxygenxml.webhelp.xsl.dita2webhelp.param

Use this extension point to pass parameters to the stylesheet specified using the [com.oxygenxml.webhelp.xsl.dita2webhelp](#) extension point.

com.oxygenxml.webhelp.xml.createMainFiles.param

Use this extension point to pass parameters to the stylesheet specified using the [com.oxygenxml.webhelp.xml.createMainFiles extension point](#).

Overriding the XSLT Stylesheet from a Customized Ant Build File

This method is useful if you want to create a customization that is only available for a certain DITA-OT extension plugin. This method implies that you create a DITA-OT plugin that defines a custom `transtype`. From the Ant target associated with the plugin `transtype`, you will specify a custom XSLT stylesheet. This customized XSLT stylesheet will import the original WebHelp stylesheet and will also add some customization templates.

If you want to use this method, you need to create a DITA-OT extension plugin, as in the following sample procedure for customizing WebHelp Responsive output:

1. In the `DITA_OT_DIR\plugins\` folder, create a folder for this plugin (for example, `com.oxygenxml.webhelp.responsive.custom`).
2. Create a **plugin.xml** file (in the folder you created in step 1) that specifies a new DITA-OT `transtype` and the build file associated with the plugin. For example:

```
<plugin id="com.oxygenxml.webhelp.responsive.customization">
  <feature extension="dita.conductor.target.relative" file="integrator.xml"/>
  <transtype name="webhelp-responsive-custom" extends="webhelp-responsive"
    desc="WebHelp Responsive Customization"/>
</plugin>
```

3. Create the **integrator.xml** file that will import the actual plugin Ant build file.

```
<project basedir="." name="Webhelp Responsive Customization">
  <import file="build.xml"/>
</project>
```

4. Create the **build.xml** file that overrides the value of properties associated with the XSLT stylesheets used to produce HTML files. The following two Ant properties can be overridden to specify your customization stylesheets:

args.wh.xml

Override this property if you want to customize the XSLT stylesheet used to produce an HTML file for each topic.

args.create.main.files.xml

Override this property if you want to customize the XSLT stylesheet used to produce the main HTML file.

For customizing the WebHelp Responsive, the build file should look like:

```
<project basedir="." name="Webhelp Responsive Customization">
  <target name="dita2webhelp-custom">
    <!--
      Override this property if you want to customize the XSLT stylesheet
      used to produce an HTML file for each topic
    -->
    <property
      name="args.wh.xml"
      value="${dita.plugin.com.oxygenxml.webhelp.responsive.custom.dir}
        /xsl/dita2webhelpCustom.xml"/>
    <!--
      Override this property if you want to customize the XSLT stylesheet
      used to produce the main HTML file.
    -->
    <property
      name="args.create.main.files.xml"
      value="${dita.plugin.com.oxygenxml.webhelp.responsive.custom.dir}
        /xsl/createMainFilesCustom.xml"/>
    <!--
      Depending on which version of WebHelp you want to customize,
      you need to delegate to different build targets:
      * dita2webhelp-responsive - when you are customizing the Webhelp Responsive
      * dita2webhelp-mobile - when you are customizing the Webhelp Mobile
      * dita2webhelp - when you are customizing the Webhelp Classic
    -->
    <antcall target="dita2webhelp-responsive"/>
  </target>
</project>
```



Note: Depending on which version of WebHelp you want to customize, you need to specify one of the following build targets:

- **dita2webhelp-responsive** - To customize WebHelp Responsive (with or without feedback) output.

- **dita2webhelp-mobile** - To customize WebHelp Classic Mobile output.
- **dita2webhelp** - To customize WebHelp Classic (with or without feedback) output.

5. Create an **xsl** directory in the plugin customization directory (that you created in step 1) to store the customized XSLT stylesheets.

Customizing the XSLT Stylesheet to Produce an HTML File for Each Topic

If you want to produce an HTML file for each topic (defined with the **args.wh.xsl** property), create a **dita2webhelpCustom.xsl** stylesheet and save it in the newly created **xsl** directory. This stylesheet will import the original stylesheet and will contain the customization templates. It should look like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:math="http://www.w3.org/2005/xpath-functions/math"
  exclude-result-prefixes="xs math"
  version="2.0">
  <!--
    Import the original stylesheet used to produce an HTML file for each topic.
  -->
  <xsl:import href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/
    dita/responsive/dita2webhelp.xsl"/>
  <!--
    Please add your customization templates here.
  -->
</xsl:stylesheet>
```

Depending on which version of WebHelp you want to customize, you need to import one of the following XSLT stylesheets:

- **dita2webhelp-responsive** (WebHelp Responsive) -

```
<xsl:import
  href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/responsive/dita2webhelp.xsl"/>
```
- **dita2webhelp-mobile** (WebHelp Classic Mobile) -

```
<xsl:import
  href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/mobile/dita2webhelp.xsl"/>
```
- **dita2webhelp** (WebHelp Classic) -

```
<xsl:import
  href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/desktop/dita2webhelp.xsl"/>
```

Customizing the XSLT Stylesheet to Produce the Main HTML File

If you just want to produce a main HTML file (defined with the **args.create.main.files.xsl** property), you need to create a **createMainFilesCustom.xsl** stylesheet and save it in the newly created **xsl** directory. This stylesheet will import the original stylesheet and will contain the customization templates. It should look like:

```
<?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"
  xmlns:math="http://www.w3.org/2005/xpath-functions/math"
  exclude-result-prefixes="xs math"
  version="2.0">
  <!--
    Import the original stylesheet used to produce the main HTML file.
  -->
  <xsl:import href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/
    responsive/createMainFiles.xsl"/>
  <!--
    Please add your customization templates here.
  -->
</xsl:stylesheet>
```

Depending on which version of WebHelp you want to customize, you need to import one of the following XSLT stylesheets:

- **dita2webhelp-responsive** (WebHelp Responsive) -

```
<xsl:import
  href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/responsive/createMainFiles.xsl"/>
```
- **dita2webhelp-mobile** (WebHelp Classic Mobile) -

```
<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/mobile/createMainFiles.xsl"/>
```

- **dita2webhelp** (WebHelp Classic) -

```
<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/desktop/createMainFiles.xsl"/>
```

Related information

[\[DITA-OT 2.3\] XSLT-Import Extension Points](#)

[\[DITA-OT 2.3\] XSLT-Parameter Extension Points](#)

Changing the Topics Displayed in the WebHelp Responsive Side TOC

A specific use-case for customizing your WebHelp Responsive output by overriding the default XSLT templates is if you want to change how the topics are displayed in the side table of contents. The default transformation generates a mini table of contents for the current topic and it contains links to the children of current topic, its siblings, and all of its ancestors.

Use-Case 1: Customizing the Side TOC Using a WebHelp XSLT-Import Extension Point

Suppose you want to customize this mini table of contents to only include the current topic and its child topics (while excluding its siblings and ancestors).

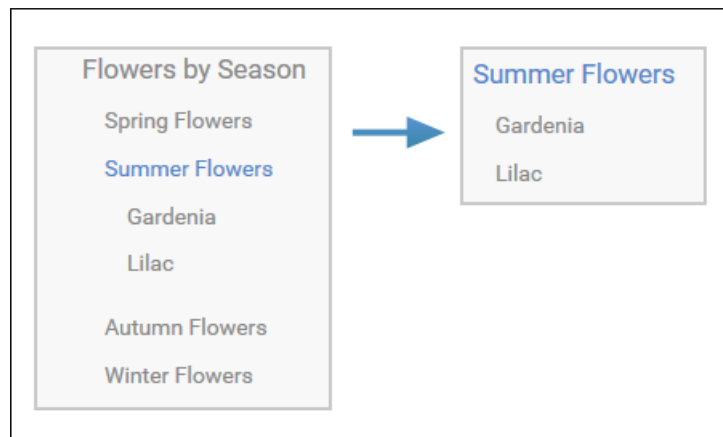


Figure 299: Example: Filtered Side Table of Contents

The default XSLT template responsible for this functionality is defined in:

`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\responsive\navigationLinks.xsl`.

```
<xsl:template
  match="
    toc:topic
    [not(@toc = 'no')]
    [not(@processing-role = 'resource-only')]"
  mode="toc-pull" priority="10">
```

This templates computes the link for the current topic along with the links for the sibling topics, and then propagates them recursively to the parent topic. For the specific use-case described above, you need to override this template so that it will produce the link for the current topic along with just the child links that the template already receives.

To add this functionality without modifying the sources of the WebHelp plugin, you can implement an external DITA-OT plugin that uses the [com.oxygenxml.webhelp.xsl.dita2webhelp extension point](#). This extension point allows you to specify a customization stylesheet that will override the template described above.

How to Use a DITA-OT Plugin to Customize the Side TOC

To add this functionality as a DITA-OT plugin, follow these steps:

1. In the `DITA_OT_DIR\plugins\` folder, create a folder for this plugin (for example, `com.oxygenxml.webhelp.custom.sidetoc`).

2. Create a **plugin.xml** file (in the folder you created in step 1) that specifies the extension point and your customization stylesheet. For example:

```
<plugin id="com.oxygenxml.webhelp.custom.sidetoc">
  <feature extension="com.oxygenxml.webhelp.xml.dita2webhelp"
    file="custom_side_TOC.xml"/>
</plugin>
```

3. Create your customization stylesheet (for example, **custom_side_TOC.xml**), and edit it to override the template that produces the side TOC:

```
<xsl:template
  match="
    toc:topic
    [not(@toc = 'no')]
    [not(@processing-role = 'resource-only')]"
  mode="toc-pull"
  priority="10">

  <xsl:param name="pathFromMapList" as="xs:string"/>
  <xsl:param name="children" select="()" as="element()*"/>
  <xsl:param name="parent" select="parent::*" as="element()?"/>

  <xsl:apply-templates select="." mode="sideTocEntry">
    <xsl:with-param name="pathFromMapList" select="$pathFromMapList"/>
    <xsl:with-param name="children" select="$children"/>
  </xsl:apply-templates>
</xsl:template>
```

4. Run the **Run DITA OT Integrator** transformation scenario.
5. Run a **DITA Map WebHelp Responsive** transformation scenario to obtain the customized side TOC.

Use-Case 2: Customizing the Side TOC Dynamically with a WebHelp XSLT-Parameter Extension Point

Another use-case might be if you want to control what is displayed in the side table of contents more dynamically, meaning that you can declare XSLT parameters in your customization stylesheet and control the parameter values from the transformation scenario. For this use-case, you can use the [com.oxygenxml.webhelp.xml.dita2webhelp.param extension point](#).

Suppose that you want to add a parameter to control whether the side TOC will only contain the current topic along with its children or if it will contain all the topics in the default form (with the current, sibling, child, and parent topics all displayed).

How to Customize the Side TOC Dynamically by Using Parameters

To add this functionality, follow these steps:

1. Create a DITA-OT plugin structure by following the first 3 steps in the [procedure above](#).
2. In the customization stylesheet that you created in step 3, declare the `side_toc_only_children` parameter and modify the template to match the topic only when the `side_toc_only_children` parameter is set to `yes`:

```
<xsl:param name="side_toc_only_children" select="'yes'"/>
...
<xsl:template
  match="
    toc:topic
    [not(@toc = 'no')]
    [not(@processing-role = 'resource-only')]
    [$side_toc_only_children = 'yes']"
  ...
```

3. Edit the **plugin.xml** file to specify the **com.oxygenxml.webhelp.xml.dita2webhelp.param** extension point and a custom parameter file by adding the following line:

```
<feature extension="com.oxygenxml.webhelp.xml.dita2webhelp.param"
  file="custom_params.xml"/>
```

4. Create a custom parameter file (for example, **custom_params.xml**). It should look like this:

```
<dummy>
  <param name="side_toc_only_children" expression="{side_toc_only_children}"
    if="side_toc_only_children"/>
</dummy>
```

5. Run the **Run DITA OT Integrator** transformation scenario.
6. Edit a **DITA Map WebHelp Responsive** transformation scenario and in the **Parameters** tab, specify the desired value (yes or no) for your custom parameter (`side_toc_only_children`).

7. Run the transformation scenario.

Search Engine Optimization for DITA WebHelp

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 fewer 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 for Oxygen XML Developer plugin to generate the `sitemap.xml` file.

- `webhelp.sitemap.change.frequency` - how frequently the WebHelp pages are likely to change (accepted values are: `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, and `never`)

- `webhelp.sitemap.priority` - the priority of each page (value ranging from 0.0 to 1.0)
3. 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.

Support for Right-to-Left (RTL) Oriented Languages for DITA WebHelp

To activate support for RTL (right-to-left) 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 Responsive Runtime Additional Parameters

A deployed WebHelp Responsive system can accept the following GET parameters:

- `contextId` - The WebHelp JavaScript engine will look for this value in the `context-help-map.xml` mapping file and load the corresponding 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`.

- `appname` - You can use this parameter in conjunction with `contextId` to search for this value in the corresponding [appname attribute value](#) in the mapping file.

```
http://localhost/webhelp/index.html?contextId=topicID&appname=myApplication
```

- `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`.

Related information

[Context-Sensitive WebHelp Responsive System](#) on page 600

Context-Sensitive WebHelp Responsive 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.

Generating Context-Sensitive Help

When WebHelp Responsive output is generated by Oxygen XML Developer plugin, the transformation process produces an XML mapping file called `context-help-map.xml` and copies it in the output folder of the transformation. This XML file maps an ID to a corresponding HTML page through an `appContext` element, as in the following example:

```
<map productID="oxy-webhelp" productVersion="1.1">
  <appContext helpID="myapp-functionid1" path="tasks/app-help1.html"/>
  <appContext helpID="myapp-functionid2" path="tasks/app-help1.html"/>
  .
  .
  .
</map>
```

The possible attributes are as follows:

helpID

A Unique ID provided by a topic from two possible sources (`resourceid` element or `id` attribute):

resourceid

The `resourceid` element is mapped into the `appContext` element and can be specified in either the `topicref` within a DITA map or in a `prolog` within a DITA topic. The `resourceid` element accepts the following attributes:

- **appname** - A name for the external application that references the topic. If this attribute is not specified, its value is considered to be empty (`""`).

- **appid** - An ID used by an application to identify the topic.
- **id** - Specifies a value that is used by a specific application to identify the topic, but this attribute is ignored if an **appid** attribute is used.



Note: Multiple **appid** values can be associated with a single **appname** value (and multiple **appname** values can be associated with a single **appid** value), but the values for both attributes work in combination to specify a specific ID for a specific application, and therefore each combination of values for the **appid** and **appname** attributes should be unique within the context of a single root map. For example, suppose that you need two different functions of an application to both open the same WebHelp page.

Example: resourceid Specified in a DITA Map

The **resourceid** element can be specified in a **topicmeta** element within a **topicref**.

```
<map title="App Help">
  <topicref href="app-help1.dita" type="task">
    <topicmeta>
      <resourceid appname="myapp" appid="functionid1"/>
      <resourceid appname="myapp" appid="functionid2"/>
    </topicmeta>
  </topicref>
</map>
```

Example: resourceid Specified in a DITA Topic

The **resourceid** element can be specified in a **prolog** element within a DITA topic.

```
<task id="app-help1">
  <title>My App Help</title>
  <prolog>
    <resourceid appname="myapp" appid="functionid1"/>
    <resourceid appname="myapp" appid="functionid2"/>
  </prolog>
  ...
</task>
```

For more information about the **resourceid** element, see [DITA Specifications: <resourceid>](#).

id

If a **resourceid** element is not declared in the DITA map or DITA topic (as described above), the **id** attribute that is set on the topic root element is mapped into the **appContext** 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 and the help system will not work properly.

path

The path to a corresponding WebHelp page. This path is relative to the location of the `context-help-map.xml` mapping file.

productID (Applicable only if you are using the WebHelp Responsive with Feedback transformation scenario)

The ID of the product for your documentation project.

productVersion (Applicable only if you are using the WebHelp Responsive with Feedback transformation scenario)

The version of the product for your documentation project.

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 the `index.html` WebHelp system file 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.

```
index.html?contextId=myDITATopic
```

Context-Sensitive Queries

You can use the URL field in your browser to search for topics in a context-sensitive WebHelp system with the assistance of the following parameters:

- **contextId** - The WebHelp JavaScript engine will look for this value in the `context-help-map.xml` mapping file and load the corresponding 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`.

- **appname** - You can use this parameter in conjunction with `contextId` to search for this value in the corresponding [appname attribute value](#) in the mapping file.

```
http://localhost/webhelp/index.html?contextId=topicID&appname=myApplication
```

Related information

[WebHelp Responsive Runtime Additional Parameters](#) on page 600

Publishing WebHelp Responsive Output on a SharePoint Site

Since WebHelp output must be published locally, on the same machine where the WebHelp process is running, to publish your files directly to a SharePoint library you need to map a network drive to connect to SharePoint and change your file extensions to `.aspx`, as described in the steps below.

To publish WebHelp Responsive output on a SharePoint site, follow this procedure:

1. Map a network drive to connect to your SharePoint library. For more information, see: <https://support.microsoft.com/en-us/kb/2616712>.
2. To allow browsers to open your published files (rather than downloading them), you need to change the file extensions from `.html` to `.aspx`. Fortunately, this can be done in the transformation scenario.
 - a. [Edit the WebHelp transformation scenario](#) and open the **Parameters** tab.
 - b. Set the `args.outext` parameter to `.aspx`.
 - c. Run the transformation scenario.

WebHelp Classic System

WebHelp is a form of online help that consists of a series of web pages (XHTML format). Its advantages include platform independence, ability to update content continuously, and it can be viewed using a regular web browser. The Oxygen XML Developer plugin WebHelp system includes several variants to suit your specific needs. The *WebHelp Classic* variant is designed for desktop systems when feedback from users is not necessary and it is available for DocBook and DITA document types.

Layout of the WebHelp Classic System Interface

The layout of the *WebHelp Classic* system is comprised of the following components:

Left Pane or Frame

This section on the left side of the help system includes the following tabs:

Content

A typical table of contents style presentation of your content. You can use the  **Expand all**/ **Collapse all** buttons to expand or collapse all the topics presented in the Table of Contents.



Note: You can enhance the appearance of items in the *Table of Contents*. See the [Customizing WebHelp Classic Systems chapter](#) for more details.

Index

Presents the index terms for your content. If your content does not contain any `indexterm` elements, this tab is not generated.

Search Results

This tab is generated when the **Search** field is used. It presents the search results in the form of links to topics where the search terms are found, along with a rating scheme for each result. For more details, see the [Search Feature section](#).

Upper Pane or Frame

The upper section of the help system includes the following features:

Search Field

Use this feature to perform searches in your content. When you enter search terms in this field, the results are displayed in the **Search Results** tab in the left section of the help system, along with a rating scheme for each result. For more details, see the [Search Feature section](#).

Frames Option

Click on this option to display the output rendered in HTML frames.

Print Option

Opens a dialog box with various printing options and a print preview.

Navigation Links

You can navigate through the content of your output using the navigation links or arrows in the upper-right part of the page. These arrows allow you to move to the [↑ Parent topic](#), [← Previous topic](#), or [→ Next topic](#). Links to the parent topics 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.

Main Pane or Frame

The content of the help pages are rendered and displayed in this main section.

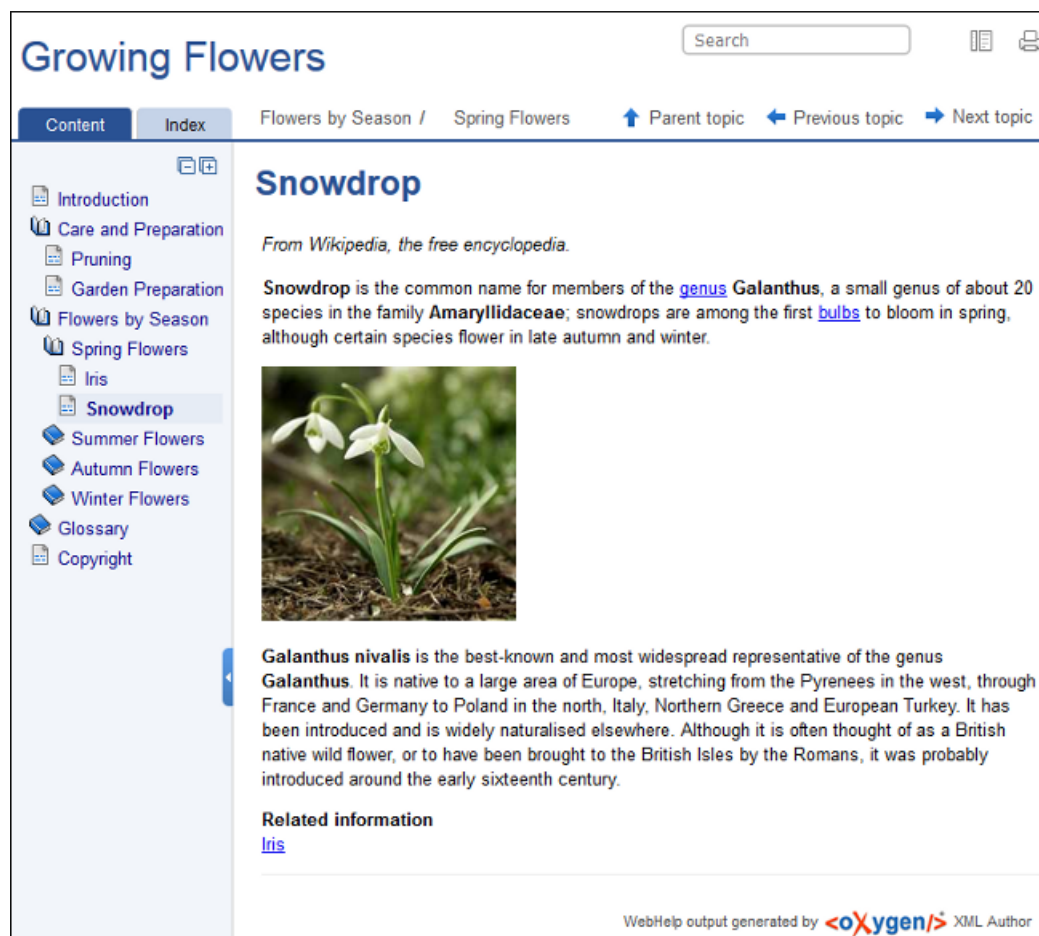


Figure 300: WebHelp Classic Output

Search Feature

When you enter search terms in the **Search** field at the top of the help system, the results are displayed in the **Search Results** tab in the left section, along with a rating scheme for each result. When you click on a result in the **Search Results** tab, the search terms are highlighted in the main pane. If you click **Enter** with the **Search** field empty, the highlights are removed.

The **Search** feature is also 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

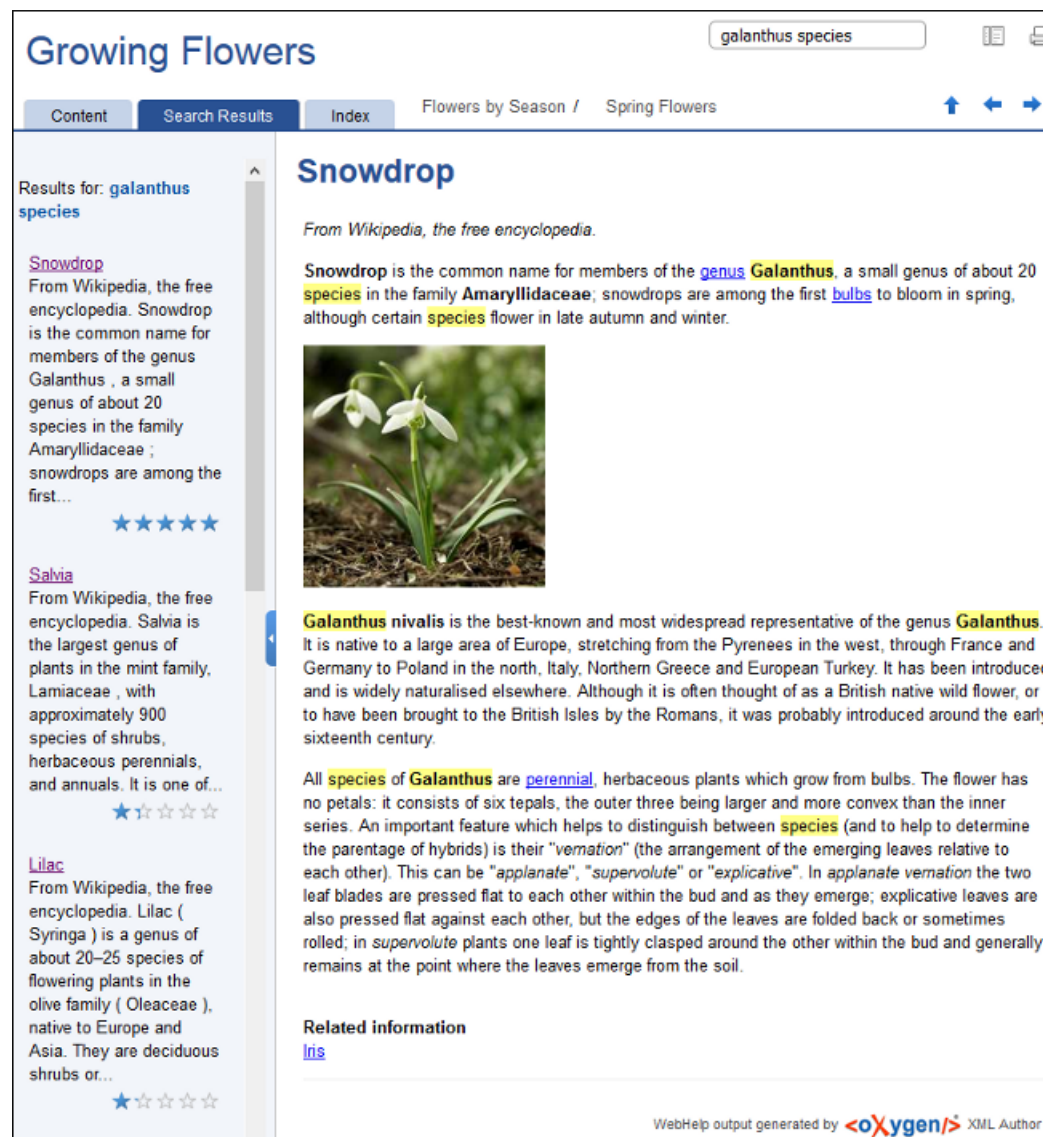


Figure 301: WebHelp Classic Search Mechanism

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 [Editing Scoring Values of Tag Elements in Search Results](#) on page 588.

This output format is compatible with the most recent versions of the following common browsers:

- Edge
- Internet Explorer (IE 8 or newer)
- Chrome
- Firefox
- Safari
- Opera

 **Important:** Due to some security restrictions in certain browsers (Google Chrome and Internet Explorer), WebHelp Classic pages loaded from the local system (through URLs of the `file:///...` format) may not work properly. We recommend that you load WebHelp Classic pages in Google Chrome or Internet Explorer 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 Classic 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 Classic pages from the local file system, the *frameset* version (`index_frames.html` start page) of the WebHelp Classic system should be used instead (`file:///...`).

WebHelp Classic with Feedback System

WebHelp is a form of online help that consists of a series of web pages (XHTML format). Its advantages include platform independence, ability to update content continuously, and a feedback mechanism that allows your authors and audience to interact with one another through comments. The Oxygen XML Developer plugin WebHelp system includes several variants to suit your specific needs. The *WebHelp Classic with Feedback* variant is designed for desktop systems, includes a feedback system that allows your users to make comments and allows you to manage and reply to them, and it is available for DocBook and DITA document types.

Layout

The layout of the *WebHelp Classic with Feedback* system is comprised of the following components:

Left Pane or Frame

This section on the left side of the help system includes the following tabs:

Content

A typical table of contents style presentation of your content. You can use the  **Expand all**/ **Collapse all** buttons to expand or collapse all the topics presented in the Table of Contents.

 **Note:** You can enhance the appearance of items in the *Table of Contents*. See the [Customizing WebHelp Classic Systems](#) chapter for more details.

Index

Presents the index terms for your content. If your content does not contain any `indexterm` elements, this tab is not generated.

Search Results

This tab is generated when the **Search** field is used. It presents the search results in the form of links to topics where the search terms are found, along with a rating scheme for each result. For more details, see the [Search Feature section](#).

Upper Pane or Frame

The upper section of the help system includes the following features:

Search Field

Use this feature to perform searches in your content. When you enter search terms in this field, the results are displayed in the **Search Results** tab in the left section of the help system, along with a rating scheme for each result. For more details, see the [Search Feature section](#).

Frames Option

Click on this option to display the output rendered in HTML frames.

Print Option

Opens a dialog box with various printing options and a print preview.

Navigation Links

You can navigate through the content of your output using the navigation links or arrows in the upper-right part of the page. These arrows allow you to move to the  **Parent topic**,  **Previous topic**, or  **Next topic**. Links to the parent topics 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.

Main Pane or Frame

The content of the help pages are rendered and displayed in this main section.

Feedback Section

The *WebHelp Classic with Feedback* system contains a **Comments** bar at the bottom of the main pane. This section is where you can interact with users through a comment system.

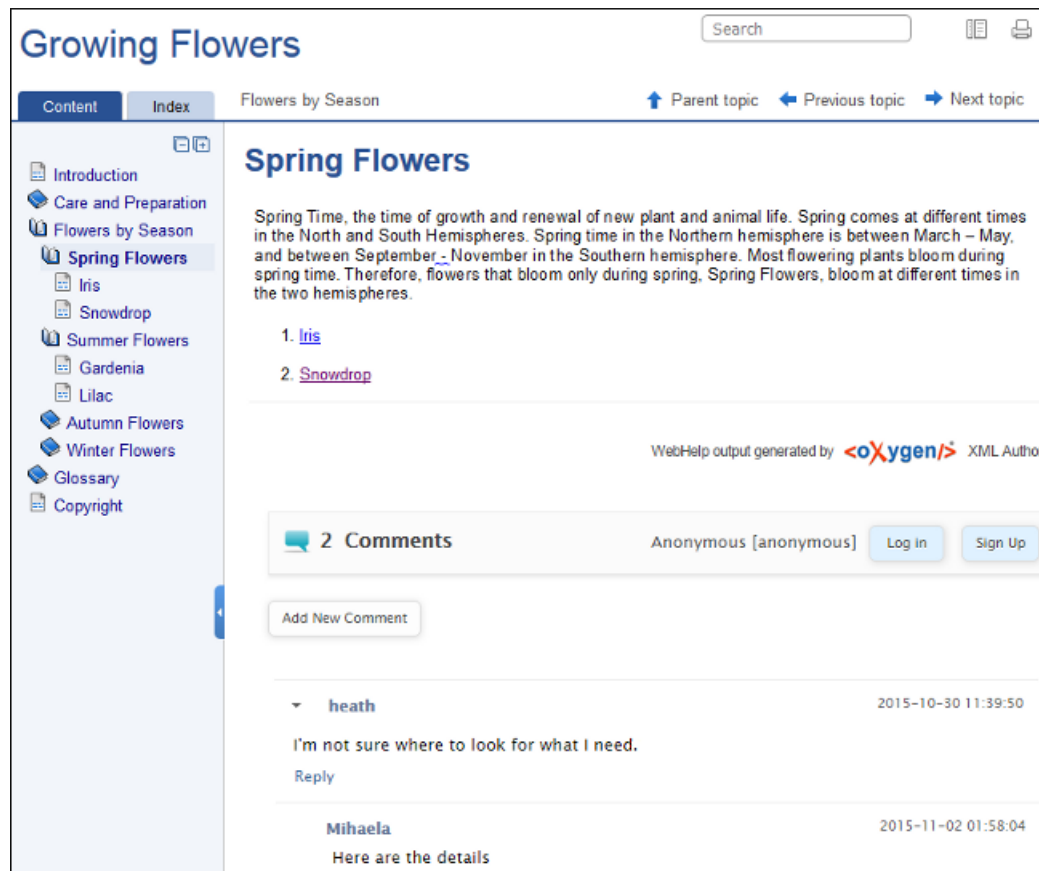


Figure 302: WebHelp Classic with Feedback System

Managing Comments

To add a new comment, click the **Add New Comment** button, or click **Reply** to add a comment to an existing thread. You can click on the **Log in** button on the right side of this bar to be authenticated as a user 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.

After you log in, your name and user name are displayed in the **Comments** bar, along with the **Log off** and **Edit** buttons. Click the **Edit** button to open the **User Profile** dialog box where 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 Classic 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.

If you are an administrator, you can manage user information and comments. For more information, see [Managing Users and Comments in a Feedback-Enabled WebHelp System](#) on page 566.

Search Feature

When you enter search terms in the **Search** field at the top of the help system, the results are displayed in the **Search Results** tab in the left section, along with a rating scheme for each result. When you click on a result in the **Search Results** tab, the search terms are highlighted in the main pane. If you click **Enter** with the **Search** field empty, the highlights are removed.

The **Search** feature is also 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

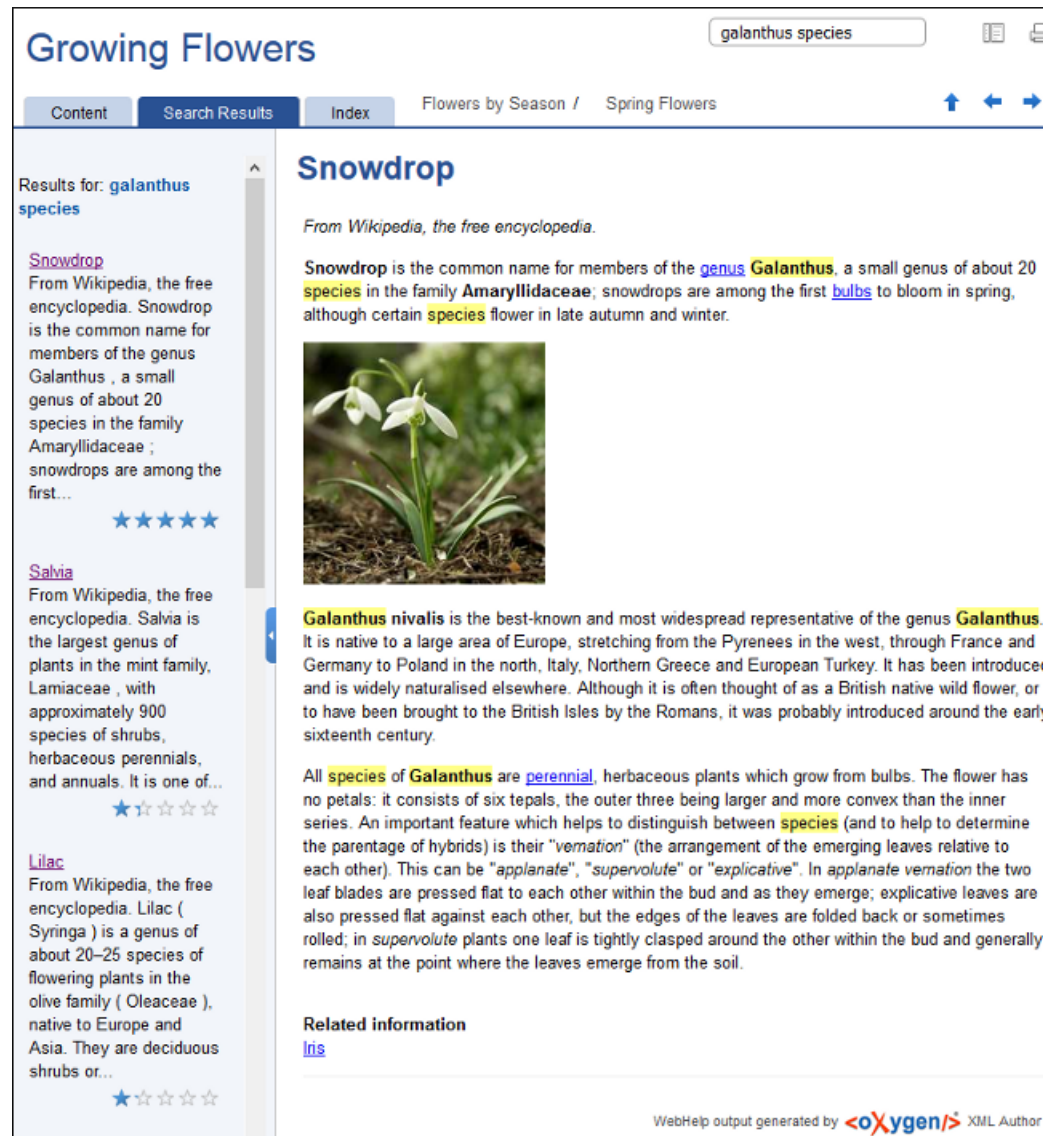


Figure 303: WebHelp Classic Search Mechanism

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 [Editing Scoring Values of Tag Elements in Search Results](#) on page 588.

This output format is compatible with the most recent versions of the following common browsers:

- Edge
- Internet Explorer (IE 8 or newer)
- Chrome
- Firefox
- Safari
- Opera

 **Important:** Due to some security restrictions in certain browsers (Google Chrome and Internet Explorer), WebHelp Classic pages loaded from the local system (through URLs of the `file:///...` format) may not work properly. We recommend that you load WebHelp Classic pages in Google Chrome or Internet Explorer only from a web server (with a URL such as `http://your.server.com/webhelp/index.html` or `http://localhost/web_pages/index.html`).

Deploying a Feedback-Enabled WebHelp System

System Requirements

The feedback-enabled WebHelp system of Oxygen XML Developer 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, Edge, Internet Explorer, Safari, Opera.

Create WebHelp with Feedback Database

The *WebHelp with Feedback* 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. Access the *phpMyAdmin* instance running on your server.
2. Click *Databases* (in the right frame) and then create a *database*. You can give it any name you want (for example *comments*).
3. Create a user with connection privileges for this database.

4. 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 with Feedback Output

If you have a web server configured with PHP and MySQL, you can deploy the *WebHelp with Feedback* 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 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_INSTALL_DIR]/feedback/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 Authentication** 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 multiple *WebHelp with Feedback* deployments. If a topic is available in multiple 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 with Feedback* 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 with Feedback* 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 three different WebHelp outputs, you have three 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 visible in other specified *products*.

Related information

[Managing Users and Comments in a Feedback-Enabled WebHelp System](#) on page 566

Refreshing the Content of a Feedback-Enabled WebHelp System

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.

Fortunately, you can refresh just the content without losing the comments or the initial system configuration. To do so, follow these steps:

1. Execute the transformation scenario that produces the *WebHelp with Feedback* output directory.
2. Go to the output directory (specified in the **Output** tab of the transformation scenario), locate the `\feedback\resources\php\config\config.php` file, and delete it.
3. Locate the `\feedback\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 Feedback-Enabled WebHelp 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 page.

oxy-webhelp 1.0 - Administrative Page						Welcome Administrator [admin] Admin Back				
						Delete Orphan Comments Delete Pending Users View All Posts Export Comments Set Version				
Search User Information										
User Name	Name	Level	Company	E-Mail	Date	WebHelp Notification	Reply Notification	Page Notification	Status	admin.user.type
admin	Administrator	admin	NA	admin@sync.ro	2015-08-04 08:56:55	yes	no	no	validated	Local User
user	user	user	noCompany	user@sync.ro	2015-08-04 08:57:46	no	yes	yes	validated	Local User

Figure 304: Administrative Page

- Use this page 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).

Customizing WebHelp Classic 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, you can style your WebHelp output through your own custom stylesheets. You can also customize your output by modifying option and parameters in the transformation scenario.

This section includes topics that explain various ways to customize your WebHelp system output, such as how to improve the appearance of the Table of Contents, add logo images in the title area, integrate with social media, add custom headers and footers, and much more.

Copying Additional Resources to WebHelp Output Directory

To copy additional resources (such as JavaScript, CSS or other resources) to the output directory of a WebHelp system, follow these steps:

1. Place all your resources in the same directory.
2. Edit the WebHelp transformation scenario.
3. Go to the **Parameters** tab.
4. Edit the value for the `webhelp.custom.resources` parameter and set it to the absolute path of the directory in step 1.
5. Click **OK** to save the changes to the transformation scenario.

All files from the new directory will be copied to the root of the WebHelp output directory.

Adding Custom HTML Content in WebHelp Classic Output

You can add custom HTML content in the WebHelp Classic output by inserting it in a well-formed XML file that will be referenced in the transformation scenario. This content may include references to additional JavaScript, CSS, and other types of resources, or such resources can be inserted inline within the HTML content that is inserted in the XML file.

To include custom HTML content in the WebHelp Classic output files, follow these steps:

1. Insert the HTML content in a well-formed XML file. There are several things to consider in regards to this XML file:
 - a. **Well-Formedness** - If the file is not a [Well-formed XML document](#) (or fragments are not well-formed), the transformation will fail.

A common use case is if you want to include several `script` or `link` elements. In this case, the XML fragment has multiple root elements and to make it well-formed, you can wrap it in an `html` element. This element tag will be filtered out and only its children will be copied to the output documents. Similarly, you can wrap your content in `head`, `body`, `html/head`, or `html/body` elements.
 - b. **Referencing Resources in the XML File** - You can include references to local resources (such as JavaScript or CSS files) by using the predefined `${oxygen-webhelp-output-dir}` macro to specify their paths relative to the output directory:

```
<html>
  <script type="text/javascript" src="${oxygen-webhelp-output-dir}/js/test.js"/>
  <link rel="stylesheet" type="text/css"
    href="${oxygen-webhelp-output-dir}/css/test.css" />
</html>
```

To copy the referenced resources to the output directory, follow the procedure in: [Copying Additional Resources to WebHelp Output Directory](#) on page 582.

- c. **Inline JavaScript or CSS Content** - If you want to include inline JavaScript or CSS content in the XML file, it is important to place this content inside an XML comment, as in the following examples:

JavaScript:

```
<script type="text/javascript">
  <!--
    /* Include JavaScript code here. */

    function myFunction() {
      return true;
    }
  -->
</script>
```

CSS:

```
<style>
  <!--
    /* Include CSS style rules here. */

    *{
      color:red
    }
  -->
</style>
```

2. Edit the WebHelp Classic transformation scenario.
3. Go to the **Parameters** tab.
4. Edit the value of the webhelp.head.script parameter and set it to the absolute path of the XML file created in step 1. Your additional content will be included at the end of the head element of your output document.



Note: If you want to include the content in the body element, use the webhelp.body.script parameter instead.

5. Click **OK** to save the changes to the transformation scenario.

Related tasks

[Copying Additional Resources to WebHelp Output Directory](#) on page 582

Adding a Button in Code Snippet Areas in DITA WebHelp Classic Output

This task will get you started with how to add an action (such as a button or link) in the code snippet areas that are displayed in WebHelp Classic output created from a DITA Map transformation. 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. It 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>
```

Adding a *Favicon* in WebHelp Systems

You can add a custom *favicon* to your WebHelp system by simply using a parameter in the transformation scenario to point to your *favicon* image. This is available for DITA and DocBook WebHelp systems using **WebHelp Responsive**, **WebHelp Responsive with Feedback**, **WebHelp Classic**, **WebHelp Classic with Feedback**, or **WebHelp Classic Mobile** transformation scenarios.

To add a *favicon*, follow these steps:

1. [Edit the WebHelp transformation scenario](#) and open the **Parameters** tab.
2. Locate the `webhelp.favicon` parameter and enter the file path that points to the image that will be use as the *favicon*.
3. Run the transformation scenario.

Adding a Logo Image in the Title Area

To add a logo in the title area of your WebHelp output, follow this procedure:

1. Edit a WebHelp transformation scenario, then open the **Parameters** tab.
2. Specify the path to your logo in the `webhelp.logo.image` parameter.
3. If you also want to add a link to your website when you click the logo image, set the URL in the `webhelp.logo.image.target.url` parameter.
4. Run the transformation scenario.

Adding Video and Audio Objects in DITA WebHelp Output

You can insert references to video and audio media resources (such as videos, audio clips, or embedded HTML frames) in your DITA topics and then publish them to WebHelp output. The media objects can be played directly in all HTML5-based outputs, including WebHelp systems.

To add media objects in the WebHelp output generated from DITA documents, follow the procedures below.

Adding Videos to DITA WebHelp Output

1. Edit the DITA topic and insert a reference to the video by manually adding an `object` element, as in one of the following examples:

```
<object outputclass="video" type="video/mp4" data="MyVideo.mp4"/>
```

or, instead of the `data` attribute, you can specify the video using a parameter like this:

```
<object outputclass="video">
  <param name="src" value="videos/MyVideo.mp4"/>
</object>
```

2. Apply a *DITA to WebHelp* transformation scenario to obtain the output.

Result: The transformation converts the `object` element to an HTML5 `video` element.

```
<video controls="controls"><source type="video/mp4" src="MyVideo.mp4"></source>
</video>
```

Adding Audio Clips to DITA WebHelp Output

1. Edit the DITA topic and insert a reference to the audio clip by manually adding an `object` element, as in one of the following examples:

```
<object outputclass="audio" type="audio/mpeg" data="MyClip.mp3"/>
```

or, instead of the `data` attribute, you can specify the audio clip using a parameter like this:

```
<object outputclass="audio">
  <param name="src" value="audio/MyClip.mp3"/>
</object>
```

2. Apply a *DITA to WebHelp* transformation scenario to obtain the output.

Result: The transformation converts the `object` element to an HTML5 audio element.

```
<audio controls="controls"><source type="audio/mpeg" src="MyClip.mp3"></source>
</audio>
```

Adding Embedded HTML Frames (such as YouTube videos) to DITA WebHelp Output

1. Edit the DITA topic and insert a reference to the embedded object by manually adding an `object` element, as in one of the following examples:

```
<object outputclass="iframe" data="https://www.youtube.com/embed/m_vv2s5Trn4"/>
```

or, instead of the `data` attribute, you can specify the object using a parameter like this:

```
<object outputclass="iframe">
  <param name="src" value="http://www.youtube.com/embed/m_vv2s5Trn4"/>
</object>
```

2. Apply a *DITA to WebHelp* transformation scenario to obtain the output.

Result: The transformation converts the `object` element to an HTML5 `iframe` element.

```
<iframe controls="controls" src="https://www.youtube.com/embed/m_vv2s5Trn4">
</iframe>
```

Adding Videos in DocBook WebHelp Classic Output

You can insert references to videos in your DocBook topics and then publish them to WebHelp Classic output. The videos can be played directly in all HTML5-based outputs, including WebHelp systems.

To add videos in the WebHelp Classic output generated from DocBook documents, follow these steps:

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.

Change Numbering Styles for Ordered Lists

Ordered lists (o l) 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.

Changing the Icons in a WebHelp Classic Table of Contents

You can change the icons that appear in a WebHelp Classic table of contents by assigning new image files in a custom CSS file. By default, the icons for the WebHelp Classic 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 other 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.

CSS Styling to Customize WebHelp Output

One way to customize WebHelp output is to use custom CSS styling. This method can be used to make small, simple styling changes or more advanced, precise changes. To implement the styling in your WebHelp output, you simply need to create the custom CSS file and reference it in your transformation scenario.

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.

Customize the Appearance of Selected Items in the Table of Contents

The appearance of selected items in the table of contents of WebHelp Classic output 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_INSTALL_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 Classic Table of Contents](#) topic).

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.

Disable Caching in WebHelp Classic Output

In cases where a set of WebHelp Classic pages need to be updated on a regular basis to deliver the latest version of the documentation, the WebHelp pages 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 Classic output, 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_INSTALL_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.

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 Developer 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_INSTALL_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.

Exclude Certain DITA Topics from WebHelp Search Results

The WebHelp **Search** engine does not index DITA topics that have the `@search` attribute set to `no`. This is useful if you have topics in your DITA map structure that you do not want to be included in search results for your WebHelp system.

To exclude DITA topics from WebHelp search results, follow these steps:

1. Edit the DITA map and for any `topicref` that you want to exclude from search results, set the `search` attribute to `no`.
For example:

```
<topicref href="../../../topics/internal-topic1.dita" search="no"/>
```

2. Save your changes to the DITA map.
3. Re-run your WebHelp system transformation scenario.

Flag DITA Content in WebHelp Output

Flagging content in WebHelp output 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 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.

Integrating Social Media and Google Tools in WebHelp Output

Oxygen XML Developer 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 Classic 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 Developer 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.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Add a Google+ Button in WebHelp Classic 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.

- 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>
```

- In Oxygen XML Developer plugin, click the  **Configure Transformation Scenario(s)** action from the toolbar.
- 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.
- Switch to the **Parameters** tab and edit the `webhelp.footer.file` parameter to reference the `google-plus-button.xml` file that you created earlier.
- Click **Ok**.
- Run the transformation scenario.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Add Tweet Button in WebHelp Classic Output

To add a Twitter™ Tweet widget to your WebHelp output, follow these steps:

- Go to the [Tweet button generator](#) page.
- Fill-in the displayed form.
The **Preview and code** area displays the code.
- 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>
```

- In Oxygen XML Developer plugin, click the  **Configure Transformation Scenario(s)** action from the toolbar.
- 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.

- Switch to the **Parameters** tab and edit the `webhelp.footer.file` parameter to reference the `tweet-button.xml` file that you created earlier.
- Click **Ok**.
- Run the transformation scenario.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Integrate Google Analytics in WebHelp Classic Output

To enable your WebHelp system to benefit from Google Analytics reports, follow these steps:

- Create a new Google Analytics account (if you do not already have one) and log on.
- Choose the Analytics solution that fits the needs of your website.
- 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-XXXXXXX-X', 'auto');
ga('send', 'pageview');
</script>
```

- Save the Tracking Code (obtained in the previous step) in a new HTML file called `googleAnalytics.html`.
- In Oxygen XML Developer plugin, click the  **Configure Transformation Scenario(s)** action from the toolbar.
- 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.
- Switch to the **Parameters** tab and edit the `webhelp.footer.file` parameter to reference the `googleAnalytics.html` file that you created earlier.
- Click **Ok**.
- Run the transformation scenario.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Integrate Google Search in WebHelp Classic Output

You can integrate Google Search into your WebHelp output.

To enable your WebHelp system to use Google Search, follow these steps:

- Go to the Google Custom Search Engine page using your Google account.
- Press the **Create a custom search engine** button.
- 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>
```

- Save the script into a well-formed HTML file called `googlecse.html`.

5. In Oxygen XML Developer 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 with the following content: `<div class="gcse-searchresults-only" data-queryParameterName="searchQuery" >` (you must use the [HTML5 version for the GCSE](#)). It is recommended that you set the `data-linkTarget` attribute value to `frm` for *frameless* versions of the WebHelp system or to `data-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. For more information about other supported attributes, see [Google Custom Search: Supported Attributes](#).
 - b) Set the value of the `webhelp.google.search.results` parameter to the file path 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.

Related information

[Integrating Social Media and Google Tools in WebHelp Output](#) on page 589

Localizing the Email Notifications of WebHelp with Feedback Systems

The feedback-enabled WebHelp systems use 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 WebHelp Classic with Feedback

1. Create the following directory:
`[OXYGEN_INSTALL_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl`
2. Copy all English template files from
`[OXYGEN_INSTALL_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_INSTALL_DIR]\frameworks\docbook\xsl\com.oxygenxml.webhelp\oxygen-webhelp\resources\php\templates\nl`
 directory and translate the content into Dutch.
4. Start Oxygen XML Developer plugin and edit the **DocBook WebHelp Classic 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 WebHelp Classic with Feedback or WebHelp Responsive 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 Developer plugin and edit the **DITA Map WebHelp Classic with Feedback** or **DITA Map WebHelp Responsive 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.

Localizing the Interface of WebHelp Output

You can localize the interface of WebHelp output for DITA or DocBook transformations.

Localizing 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. 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.
4. 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"/>`.
5. 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).
6. Run the transformation scenario to produce the WebHelp output.

Related information

[Searching Japanese Content in WebHelp Pages](#) on page 593

Localizing 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_INSTALL_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_INSTALL_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_INSTALL_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.

Related information

[Searching Japanese Content in WebHelp Pages](#) on page 593

Searching Japanese Content in WebHelp Pages

To optimize the indexing of Japanese content in WebHelp pages, the Lucene Kuromoji Japanese analyzer can be used. This analyzer is included in the Oxygen XML Developer plugin installation kit.

Activating Japanese Indexing in DITA WebHelp Systems

To activate the Japanese indexing in your WebHelp system generated from DITA content, follow these steps:

1. Set the language for your content to Japanese with one of the following two methods:
 - Edit a **DITA to WebHelp** transformation scenario and in the [Parameters tab](#), set the value of the `args.default.language` parameter to `ja-jp`.
 - Set the `xml:lang` attribute on the root of the DITA map and the referenced topics to `ja-jp`.
2. For the analyzer to work properly, search terms that are entered into the WebHelp search text field must be separated by spaces.
3. Run the **DITA to WebHelp** transformation scenario to generate the output.

Optionally a Japanese user dictionary can be set with the [webhelp.search.japanese.dictionary](#) parameter.

Activating Japanese Indexing in DocBook WebHelp Systems

To activate the Japanese indexing in your WebHelp system generated from DocBook content, follow these steps:

1. Edit a **DocBook to WebHelp** transformation scenario and in the [Parameters tab](#), set the value of the `l10n.gentext.default.language` parameter to `ja`.
2. Run the transformation scenario to generate the output.

Related tasks

[Localizing the Interface of WebHelp Output \(for DITA Map Transformations\)](#) on page 593

[Localizing the Interface of WebHelp Output \(for DocBook Transformations\)](#) on page 624

Overriding the XSLT Processing Step of a DITA WebHelp Transformation

There are several methods that you can use to customize your WebHelp output. Since WebHelp output is primarily obtained by running XSLT transformations over the DITA input files (through the **DITA Map WebHelp** transformation scenarios), one possibility would be to override the default XSLT templates that are used for WebHelp transformations by using some extension points.

XSLT-Import Extension Points

You can customize your WebHelp output by overriding the default XSLT templates that are used for each type of WebHelp transformation. This can be done by creating a DITA extension plugin that specifies one of the following extension points:

com.oxygenxml.webhelp.xml.dita2webhelp

You can use this extension point to override a template of the XSLT stylesheet (`dita2webhelpImpl.xml`) that produces an HTML file for each DITA topic. Depending on the type of WebHelp transformation you are currently using, the path to this stylesheet is as follows:

- **WebHelp Classic Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\desktop\dita2webhelpImpl.xml`
- **WebHelp Classic Mobile Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\mobile\dita2webhelpImpl.xml`
- **WebHelp Responsive Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\responsive\dita2webhelpImpl.xml`

com.oxygenxml.webhelp.xml.createMainFiles

You can use this extension point to override a template of the XSLT stylesheet (`createMainFilesImpl.xml`) that produces the WebHelp main HTML page. Depending on the type of WebHelp transformation you are currently using, the path to this stylesheet is as follows:

- **WebHelp Classic Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\desktop\createMainFilesImpl.xml`
- **WebHelp Classic Mobile Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\mobile\createMainFilesImpl.xml`
- **WebHelp Responsive Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\responsive\createMainFilesImpl.xml`



Attention: The customizations you made by using this extension point will affect all WebHelp transformations. If you want to have a customization that is only available for a certain plugin, please use the [Overriding the XSLT Stylesheet from a Customized ANT Build File](#) method.

XSLT-Parameter Extension Points

In case your customization stylesheet declares one or more XSLT parameters and you want to control their value from the transformation scenario, you can use one of the following XSLT parameter extension points:

com.oxygenxml.webhelp.xml.dita2webhelp.param

Use this extension point to pass parameters to the stylesheet specified using the [com.oxygenxml.webhelp.xml.dita2webhelp extension point](#).

com.oxygenxml.webhelp.xml.createMainFiles.param

Use this extension point to pass parameters to the stylesheet specified using the [com.oxygenxml.webhelp.xml.createMainFiles extension point](#).

Overriding the XSLT Stylesheet from a Customized Ant Build File

This method is useful if you want to create a customization that is only available for a certain DITA-OT extension plugin. This method implies that you create a DITA-OT plugin that defines a custom `transtype`. From the Ant target associated with the plugin `transtype`, you will specify a custom XSLT stylesheet. This customized XSLT stylesheet will import the original WebHelp stylesheet and will also add some customization templates.

If you want to use this method, you need to create a DITA-OT extension plugin, as in the following sample procedure for customizing WebHelp Responsive output:

1. In the `DITA_OT_DIR\plugins\` folder, create a folder for this plugin (for example, `com.oxygenxml.webhelp.responsive.custom`).
2. Create a **plugin.xml** file (in the folder you created in step 1) that specifies a new DITA-OT `transtype` and the build file associated with the plugin. For example:

```
<plugin id="com.oxygenxml.webhelp.responsive.customization">
  <feature extension="dita.conductor.target.relative" file="integrator.xml"/>
  <transtype name="webhelp-responsive-custom" extends="webhelp-responsive"
    desc="WebHelp Responsive Customization"/>
</plugin>
```

3. Create the **integrator.xml** file that will import the actual plugin Ant build file.

```
<project basedir="." name="Webhelp Responsive Customization">
  <import file="build.xml"/>
</project>
```

4. Create the **build.xml** file that overrides the value of properties associated with the XSLT stylesheets used to produce HTML files. The following two Ant properties can be overridden to specify your customization stylesheets:

args.wh.xml

Override this property if you want to customize the XSLT stylesheet used to produce an HTML file for each topic.

args.create.main.files.xml

Override this property if you want to customize the XSLT stylesheet used to produce the main HTML file.

For customizing the WebHelp Responsive, the build file should look like:

```
<project basedir="." name="Webhelp Responsive Customization">
  <target name="dita2webhelp-custom">
    <!--
      Override this property if you want to customize the XSLT stylesheet
      used to produce an HTML file for each topic
    -->
    <property
      name="args.wh.xml"
      value="${dita.plugin.com.oxygenxml.webhelp.responsive.custom.dir}
              /xsl/dita2webhelpCustom.xml"/>
    <!--
      Override this property if you want to customize the XSLT stylesheet
      used to produce the main HTML file.
    -->
    <property
      name="args.create.main.files.xml"
      value="${dita.plugin.com.oxygenxml.webhelp.responsive.custom.dir}
              /xsl/createMainFilesCustom.xml"/>
    <!--
      Depending on which version of WebHelp you want to customize,
      you need to delegate to different build targets:
      * dita2webhelp-responsive - when you are customizing the Webhelp Responsive
      * dita2webhelp-mobile - when you are customizing the Webhelp Mobile
      * dita2webhelp - when you are customizing the Webhelp Classic
    -->
    <antcall target="dita2webhelp-responsive"/>
  </target>
</project>
```



Note: Depending on which version of WebHelp you want to customize, you need to specify one of the following build targets:

- **dita2webhelp-responsive** - To customize WebHelp Responsive (with or without feedback) output.
- **dita2webhelp-mobile** - To customize WebHelp Classic Mobile output.
- **dita2webhelp** - To customize WebHelp Classic (with or without feedback) output.

5. Create an **xsl** directory in the plugin customization directory (that you created in step 1) to store the customized XSLT stylesheets.

Customizing the XSLT Stylesheet to Produce an HTML File for Each Topic

If you want to produce an HTML file for each topic (defined with the **args.wh.xml** property), create a **dita2webhelpCustom.xml** stylesheet and save it in the newly created **xsl** directory. This stylesheet will import the original stylesheet and will contain the customization templates. It should look like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:math="http://www.w3.org/2005/ xpath-functions/math"
  exclude-result-prefixes="xs math"
  version="2.0">
  <!--
    Import the original stylesheet used to produce an HTML file for each topic.
  -->
  <xsl:import href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/
              dita/responsive/dita2webhelp.xml"/>
  <!--
    Please add your customization templates here.
  -->
</xsl:stylesheet>
```

Depending on which version of WebHelp you want to customize, you need to import one of the following XSLT stylesheets:

- **dita2webhelp-responsive** (WebHelp Responsive) -
`<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/responsive/dita2webhelp.xsl"/>`
- **dita2webhelp-mobile** (WebHelp Classic Mobile) -
`<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/mobile/dita2webhelp.xsl"/>`
- **dita2webhelp** (WebHelp Classic) -
`<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/desktop/dita2webhelp.xsl"/>`

Customizing the XSLT Stylesheet to Produce the Main HTML File

If you just want to produce a main HTML file (defined with the **args.create.main.files.xsl** property), you need to create a **createMainFilesCustom.xsl** stylesheet and save it in the newly created **xsl** directory. This stylesheet will import the original stylesheet and will contain the customization templates. It should look like:

```
<?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"
  xmlns:math="http://www.w3.org/2005/xpath-functions/math"
  exclude-result-prefixes="xs math"
  version="2.0">
  <!--
    Import the original stylesheet used to produce the main HTML file.
  -->
  <xsl:import href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/
    responsive/createMainFiles.xsl"/>
  <!--
    Please add your customization templates here.
  -->
</xsl:stylesheet>
```

Depending on which version of WebHelp you want to customize, you need to import one of the following XSLT stylesheets:

- **dita2webhelp-responsive** (WebHelp Responsive) -
`<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/responsive/createMainFiles.xsl"/>`
- **dita2webhelp-mobile** (WebHelp Classic Mobile) -
`<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/mobile/createMainFiles.xsl"/>`
- **dita2webhelp** (WebHelp Classic) -
`<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/desktop/createMainFiles.xsl"/>`

Related information

[\[DITA-OT 2.3\] XSLT-Import Extension Points](#)

[\[DITA-OT 2.3\] XSLT-Parameter Extension Points](#)

Removing the *Previous/Next* Links from WebHelp Classic Output

The **Previous** and **Next** links that are created at the top area of each WebHelp Classic page can be hidden with a CSS code.

To remove these links from WebHelp Classic output, 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.

Search Engine Optimization for DITA WebHelp

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 fewer 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 for Oxygen XML Developer plugin to generate the `sitemap.xml` file.

- `webhelp.sitemap.change.frequency` - how frequently the WebHelp pages are likely to change (accepted values are: `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, and `never`)
- `webhelp.sitemap.priority` - the priority of each page (value ranging from 0.0 to 1.0)

3. 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.

Support for Right-to-Left (RTL) Oriented Languages for DITA WebHelp

To activate support for RTL (right-to-left) 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 Skin Builder

The **WebHelp Skin Builder** is a simple, easy-to-use tool, specially designed to assist users to visually customize the look and feel of the WebHelp output. It is implemented as an online tool hosted on the Oxygen XML Developer plugin website and allows you to experiment with various styles and colors over a documentation sample.

To be able to use the **Skin Builder**, you need:

- An Internet connection and unrestricted access to Oxygen XML Developer plugin website.
- A 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 <https://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**.

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 Classic Transformation Scenario

1. Start Oxygen XML Developer 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 Classic Transformation Scenario

1. Start Oxygen XML Developer 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` and `webhelp.logo.image.target.url`.
5. Run the transformation to obtain the WebHelp output.

To see our video demonstration about using the WebHelp Skin Builder, go to https://www.oxygenxml.com/demo/Skin_Builder.html.

Related information

[Skins Tab \(DITA OT Transformations\)](#)

WebHelp Classic 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 for this value in the `context-help-map.xml` mapping file and load the corresponding 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`.

- `appname` - You can use this parameter in conjunction with `contextId` to search for this value in the corresponding [appname attribute value](#) in the mapping file.

```
http://localhost/webhelp/index.html?contextId=topicID&appname=myApplication
```

- `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`.

Related information

[Context-Sensitive WebHelp Classic System](#) on page 632

Context-Sensitive WebHelp Classic 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.

Generating Context-Sensitive Help

When WebHelp Classic output is generated by Oxygen XML Developer plugin, the transformation process produces an XML mapping file called `context-help-map.xml` and copies it in the output folder of the transformation. This XML file maps an ID to a corresponding HTML page through an `appContext` element, as in the following example:

```
<map productID="oxy-webhelp" productVersion="1.1">
  <appContext helpID="myapp-functionid1" path="tasks/app-help1.html"/>
  <appContext helpID="myapp-functionid2" path="tasks/app-help1.html"/>
</map>
```

The possible attributes are as follows:

helpID

A Unique ID provided by a topic from two possible sources (`resourceid` element or `id` attribute):

resourceid

The `resourceid` element is mapped into the `appContext` element and can be specified in either the `topicref` within a DITA map or in a `prolog` within a DITA topic. The `resourceid` element accepts the following attributes:

- **appname** - A name for the external application that references the topic. If this attribute is not specified, its value is considered to be empty ("").
- **appid** - An ID used by an application to identify the topic.
- **id** - Specifies a value that is used by a specific application to identify the topic, but this attribute is ignored if an **appid** attribute is used.



Note: Multiple `appid` values can be associated with a single `appname` value (and multiple `appname` values can be associated with a single `appid` value), but the values for both attributes work in combination to specify a specific ID for a specific application, and therefore each combination of values for the `appid` and `appname` attributes should be unique within the context of a single root map. For example, suppose that you need two different functions of an application to both open the same WebHelp page.

Example: resourceid Specified in a DITA Map

The `resourceid` element can be specified in a `topicmeta` element within a `topicref`.

```
<map title="App Help">
  <topicref href="app-help1.dita" type="task">
    <topicmeta>
      <resourceid appname="myapp" appid="functionid1"/>
      <resourceid appname="myapp" appid="functionid2"/>
    </topicmeta>
  </topicref>
</map>
```

Example: resourceid Specified in a DITA Topic

The `resourceid` element can be specified in a `prolog` element within a DITA topic.

```
<task id="app-help1">
  <title>My App Help</title>
  <prolog>
    <resourceid appname="myapp" appid="functionid1"/>
    <resourceid appname="myapp" appid="functionid2"/>
  </prolog>
  ...
</task>
```

For more information about the `resourceid` element, see [DITA Specifications: <resourceid>](#).

id

If a `resourceid` element is not declared in the DITA map or DITA topic (as described above), the `id` attribute that is set on the topic root element is mapped into the `appContext` 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 and the help system will not work properly.

path

The path to a corresponding WebHelp page. This path is relative to the location of the `context-help-map.xml` mapping file.

productID (Applicable only if you are using the WebHelp Classic with Feedback transformation scenario)

The ID of the product for your documentation project.

productVersion (Applicable only if you are using the WebHelp Classic with Feedback transformation scenario)

The version of the product for your documentation project.

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
```



Tip: You can use an *anchor* in the `contextId` parameter to jump to a specific section in a document. For example, `contextId=topicID#anchor`.

Context-Sensitive Queries

You can use the URL field in your browser to search for topics in a context-sensitive WebHelp system with the assistance of the following parameters:

- `contextId` - The WebHelp JavaScript engine will look for this value in the `context-help-map.xml` mapping file and load the corresponding 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`.

- `appname` - You can use this parameter in conjunction with `contextId` to search for this value in the corresponding *appname attribute value* in the mapping file.

```
http://localhost/webhelp/index.html?contextId=topicID&appname=myApplication
```

Related information

[WebHelp Classic Runtime Additional Parameters](#) on page 631

Publishing WebHelp Classic Output on a SharePoint Site

Since WebHelp output must be published locally, on the same machine where the WebHelp process is running, to publish your files directly to a SharePoint library you need to map a network drive to connect to SharePoint and change your file extensions to `.aspx`, as described in the steps below.

To publish WebHelp Classic output on a SharePoint site, follow this procedure:

1. Map a network drive to connect to your SharePoint library. For more information, see: <https://support.microsoft.com/en-us/kb/2616712>.

2. To allow browsers to open your published files (rather than downloading them), you need to change the file extensions from `.html` to `.aspx`. Fortunately, this can be done in the transformation scenario.
 - a. [Edit the WebHelp transformation scenario](#) and open the **Parameters** tab.
 - a. For a DITA transformation, set the `args.outext` parameter to `.aspx`.
 - b. For a DocBook transformation, set the `html.ext` parameter to `.aspx`.
 - b. Run the transformation scenario.

WebHelp Classic Mobile System

WebHelp is a form of online help that consists of a series of web pages (XHTML format). Its advantages include platform independence, ability to update content continuously, and it can be viewed using a regular web browser. The Oxygen XML Developer plugin WebHelp system includes several variants to suit your specific needs. The *WebHelp Classic Mobile* variant works on multiple platforms (Android, iOS, BlackBerry, Windows Mobile) and is specially designed for mobile devices when feedback from users is not necessary. It is available for DocBook and DITA document types. The functionality of the desktop WebHelp Classic layout is preserved, it is organized in an intuitive layout, and offers table of contents, search capabilities, and index navigation.



Figure 305: WebHelp Classic Mobile

 **Important:** Due to some security restrictions in certain browsers (Google Chrome and Internet Explorer), WebHelp Classic pages loaded from the local system (through URLs of the `file:///...` format) may not work properly. We recommend that you load WebHelp Classic pages in Google Chrome or Internet Explorer only from a web server (with a URL such as `http://your.server.com/webhelp/index.html` or `http://localhost/web_pages/index.html`).

Customizing WebHelp Classic Mobile Systems

If you are familiar with CSS and coding, you can style your WebHelp output through your own custom stylesheets. You can also customize your output by modifying option and parameters in the transformation scenario. This section includes topics that explain various ways to customize your WebHelp Classic Mobile system output.

Copying Additional Resources to WebHelp Output Directory

To copy additional resources (such as JavaScript, CSS or other resources) to the output directory of a WebHelp system, follow these steps:

1. Place all your resources in the same directory.
2. Edit the WebHelp transformation scenario.
3. Go to the **Parameters** tab.

4. Edit the value for the `webhelp.custom.resources` parameter and set it to the absolute path of the directory in step 1.
5. Click **OK** to save the changes to the transformation scenario.
All files from the new directory will be copied to the root of the WebHelp output directory.

Adding Custom HTML Content in WebHelp Classic Output

You can add custom HTML content in the WebHelp Classic output by inserting it in a well-formed XML file that will be referenced in the transformation scenario. This content may include references to additional JavaScript, CSS, and other types of resources, or such resources can be inserted inline within the HTML content that is inserted in the XML file.

To include custom HTML content in the WebHelp Classic output files, follow these steps:

1. Insert the HTML content in a well-formed XML file. There are several things to consider in regards to this XML file:
 - a. **Well-Formedness** - If the file is not a [Well-formed XML document](#) (or fragments are not well-formed), the transformation will fail.

A common use case is if you want to include several `script` or `link` elements. In this case, the XML fragment has multiple root elements and to make it well-formed, you can wrap it in an `html` element. This element tag will be filtered out and only its children will be copied to the output documents. Similarly, you can wrap your content in `head`, `body`, `html/head`, or `html/body` elements.

- b. **Referencing Resources in the XML File** - You can include references to local resources (such as JavaScript or CSS files) by using the predefined `${oxygen-webhelp-output-dir}` macro to specify their paths relative to the output directory:

```
<html>
  <script type="text/javascript" src="${oxygen-webhelp-output-dir}/js/test.js"/>
  <link rel="stylesheet" type="text/css"
    href="${oxygen-webhelp-output-dir}/css/test.css" />
</html>
```

To copy the referenced resources to the output directory, follow the procedure in: [Copying Additional Resources to WebHelp Output Directory](#) on page 582.

- c. **Inline JavaScript or CSS Content** - If you want to include inline JavaScript or CSS content in the XML file, it is important to place this content inside an XML comment, as in the following examples:

JavaScript:

```
<script type="text/javascript">
  <!--
    /* Include JavaScript code here. */

    function myFunction() {
      return true;
    }
  -->
</script>
```

CSS:

```
<style>
  <!--
    /* Include CSS style rules here. */

    *{
      color:red
    }
  -->
</style>
```

2. Edit the WebHelp Classic transformation scenario.
3. Go to the **Parameters** tab.
4. Edit the value of the `webhelp.head.script` parameter and set it to the absolute path of the XML file created in step 1. Your additional content will be included at the end of the `head` element of your output document.



Note: If you want to include the content in the `body` element, use the `webhelp.body.script` parameter instead.

5. Click **OK** to save the changes to the transformation scenario.

Related tasks

[Copying Additional Resources to WebHelp Output Directory](#) on page 582

Adding a Button in Code Snippet Areas in DITA WebHelp Classic Output

This task will get you started with how to add an action (such as a button or link) in the code snippet areas that are displayed in WebHelp Classic output created from a DITA Map transformation. 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. It 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>
```

Adding a Favicon in WebHelp Systems

You can add a custom *favicon* to your WebHelp system by simply using a parameter in the transformation scenario to point to your *favicon* image. This is available for DITA and DocBook WebHelp systems using **WebHelp Responsive**, **WebHelp Responsive with Feedback**, **WebHelp Classic**, **WebHelp Classic with Feedback**, or **WebHelp Classic Mobile** transformation scenarios.

To add a *favicon*, follow these steps:

1. [Edit the WebHelp transformation scenario](#) and open the **Parameters** tab.
2. Locate the `webhelp.favicon` parameter and enter the file path that points to the image that will be use as the *favicon*.
3. Run the transformation scenario.

Adding Video and Audio Objects in DITA WebHelp Output

You can insert references to video and audio media resources (such as videos, audio clips, or embedded HTML frames) in your DITA topics and then publish them to WebHelp output. The media objects can be played directly in all HTML5-based outputs, including WebHelp systems.

To add media objects in the WebHelp output generated from DITA documents, follow the procedures below.

Adding Videos to DITA WebHelp Output

1. Edit the DITA topic and insert a reference to the video by manually adding an `object` element, as in one of the following examples:

```
<object outputclass="video" type="video/mp4" data="MyVideo.mp4"/>
```

or, instead of the `data` attribute, you can specify the video using a parameter like this:

```
<object outputclass="video">
  <param name="src" value="videos/MyVideo.mp4"/>
</object>
```

2. Apply a *DITA to WebHelp* transformation scenario to obtain the output.

Result: The transformation converts the `object` element to an HTML5 `video` element.

```
<video controls="controls"><source type="video/mp4" src="MyVideo.mp4"></source>
</video>
```

Adding Audio Clips to DITA WebHelp Output

1. Edit the DITA topic and insert a reference to the audio clip by manually adding an `object` element, as in one of the following examples:

```
<object outputclass="audio" type="audio/mpeg" data="MyClip.mp3"/>
```

or, instead of the `data` attribute, you can specify the audio clip using a parameter like this:

```
<object outputclass="audio">
  <param name="src" value="audio/MyClip.mp3"/>
</object>
```

2. Apply a *DITA to WebHelp* transformation scenario to obtain the output.

Result: The transformation converts the `object` element to an HTML5 `audio` element.

```
<audio controls="controls"><source type="audio/mpeg" src="MyClip.mp3"></source>
</audio>
```

Adding Embedded HTML Frames (such as YouTube videos) to DITA WebHelp Output

1. Edit the DITA topic and insert a reference to the embedded object by manually adding an `object` element, as in one of the following examples:

```
<object outputclass="iframe" data="https://www.youtube.com/embed/m_vv2s5Trn4"/>
```

or, instead of the `data` attribute, you can specify the object using a parameter like this:

```
<object outputclass="iframe">
  <param name="src" value="http://www.youtube.com/embed/m_vv2s5Trn4"/>
</object>
```

2. Apply a *DITA to WebHelp* transformation scenario to obtain the output.

Result: The transformation converts the `object` element to an HTML5 `iframe` element.

```
<iframe controls="controls" src="https://www.youtube.com/embed/m_vv2s5Trn4">
</iframe>
```

Adding Videos in DocBook WebHelp Classic Output

You can insert references to videos in your DocBook topics and then publish them to WebHelp Classic output. The videos can be played directly in all HTML5-based outputs, including WebHelp systems.

To add videos in the WebHelp Classic output generated from DocBook documents, follow these steps:

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 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.

Change Numbering Styles for Ordered Lists

Ordered lists (O L) 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.

Changing the Icons in a WebHelp Classic Table of Contents

You can change the icons that appear in a WebHelp Classic table of contents by assigning new image files in a custom CSS file. By default, the icons for the WebHelp Classic 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 other 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.

CSS Styling to Customize WebHelp Output

One way to customize WebHelp output is to use custom CSS styling. This method can be used to make small, simple styling changes or more advanced, precise changes. To implement the styling in your WebHelp output, you simply need to create the custom CSS file and reference it in your transformation scenario.

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.

Customize the Appearance of Selected Items in the Table of Contents

The appearance of selected items in the table of contents of WebHelp Classic output 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_INSTALL_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 Classic Table of Contents](#) topic).

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.

Disable Caching in WebHelp Classic Output

In cases where a set of WebHelp Classic pages need to be updated on a regular basis to deliver the latest version of the documentation, the WebHelp pages 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 Classic output, 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_INSTALL_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.

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 Developer 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_INSTALL_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.

Exclude Certain DITA Topics from WebHelp Search Results

The WebHelp **Search** engine does not index DITA topics that have the `@search` attribute set to `no`. This is useful if you have topics in your DITA map structure that you do not want to be included in search results for your WebHelp system.

To exclude DITA topics from WebHelp search results, follow these steps:

1. Edit the DITA map and for any `topicref` that you want to exclude from search results, set the `search` attribute to `no`. For example:

```
<topicref href="../../../topics/internal-topic1.dita" search="no"/>
```

2. Save your changes to the DITA map.
3. Re-run your WebHelp system transformation scenario.

Flag DITA Content in WebHelp Output

Flagging content in WebHelp output 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 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.

Integrating Social Media and Google Tools in WebHelp Output

Oxygen XML Developer 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 Classic 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 Developer 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.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Add a Google+ Button in WebHelp Classic 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 Developer 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.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Add Tweet Button in WebHelp Classic 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 Developer 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.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Integrate Google Analytics in WebHelp Classic 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-XXXXXXX-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 Developer 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.

Related information

[DITA Map to WebHelp Output](#) on page 450

How to Integrate Google Search in WebHelp Classic 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 Developer 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 with the following content: `<div class="gcse-searchresults-only" data-queryParameterName="searchQuery" >` (you must use the [HTML5 version for the GCSE](#)). It is recommended that you set the `data-linkTarget` attribute value to `frm` for *frameless* versions of the WebHelp system or to `data-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. For more information about other supported attributes, see [Google Custom Search: Supported Attributes](#).

- b) Set the value of the `webhelp.google.search.results` parameter to the file path 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.

Related information

[Integrating Social Media and Google Tools in WebHelp Output](#) on page 589

Localizing the Interface of WebHelp Output

You can localize the interface of WebHelp output for DITA or DocBook transformations.

Localizing 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. 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.
4. 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"/>`.
5. 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).
6. Run the transformation scenario to produce the WebHelp output.

Related information

[Searching Japanese Content in WebHelp Pages](#) on page 593

Localizing 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_INSTALL_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_INSTALL_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_INSTALL_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.

Related information

[Searching Japanese Content in WebHelp Pages](#) on page 593

Searching Japanese Content in WebHelp Pages

To optimize the indexing of Japanese content in WebHelp pages, the Lucene Kuromoji Japanese analyzer can be used. This analyzer is included in the Oxygen XML Developer plugin installation kit.

Activating Japanese Indexing in DITA WebHelp Systems

To activate the Japanese indexing in your WebHelp system generated from DITA content, follow these steps:

1. Set the language for your content to Japanese with one of the following two methods:
 - Edit a **DITA to WebHelp** transformation scenario and in the [Parameters tab](#), set the value of the `args.default.language` parameter to `ja-jp`.
 - Set the `xml:lang` attribute on the root of the DITA map and the referenced topics to `ja-jp`.
2. For the analyzer to work properly, search terms that are entered into the WebHelp search text field must be separated by spaces.
3. Run the **DITA to WebHelp** transformation scenario to generate the output.

Optionally a Japanese user dictionary can be set with the [webhelp.search.japanese.dictionary parameter](#).

Activating Japanese Indexing in DocBook WebHelp Systems

To activate the Japanese indexing in your WebHelp system generated from DocBook content, follow these steps:

1. Edit a **DocBook to WebHelp** transformation scenario and in the [Parameters tab](#), set the value of the `l10n.gentext.default.language` parameter to `ja`.
2. Run the transformation scenario to generate the output.

Related tasks

[Localizing the Interface of WebHelp Output \(for DITA Map Transformations\)](#) on page 593

[Localizing the Interface of WebHelp Output \(for DocBook Transformations\)](#) on page 624

Overriding the XSLT Processing Step of a DITA WebHelp Transformation

There are several methods that you can use to customize your WebHelp output. Since WebHelp output is primarily obtained by running XSLT transformations over the DITA input files (through the **DITA Map WebHelp** transformation scenarios), one possibility would be to override the default XSLT templates that are used for WebHelp transformations by using some extension points.

XSLT-Import Extension Points

You can customize your WebHelp output by overriding the default XSLT templates that are used for each type of WebHelp transformation. This can be done by creating a DITA extension plugin that specifies one of the following extension points:

com.oxygenxml.webhelp.xsl.dita2webhelp

You can use this extension point to override a template of the XSLT stylesheet (`dita2webhelpImpl.xsl`) that produces an HTML file for each DITA topic. Depending on the type of WebHelp transformation you are currently using, the path to this stylesheet is as follows:

- **WebHelp Classic Transformations** -
`DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\desktop\dita2webhelpImpl.xsl`

- **WebHelp Classic Mobile Transformations** -
DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\mobile\dita2webhelpImpl.xsl
- **WebHelp Responsive Transformations** -
DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\responsive\dita2webhelpImpl.xsl

com.oxygenxml.webhelp.xsl.createMainFiles

You can use this extension point to override a template of the XSLT stylesheet (createMainFilesImpl.xsl) that produces the WebHelp main HTML page. Depending on the type of WebHelp transformation you are currently using, the path to this stylesheet is as follows:

- **WebHelp Classic Transformations** -
DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\desktop\createMainFilesImpl.xsl
- **WebHelp Classic Mobile Transformations** -
DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\mobile\createMainFilesImpl.xsl
- **WebHelp Responsive Transformations** -
DITA_OT_DIR\plugins\com.oxygenxml.webhelp\xsl\dita\responsive\createMainFilesImpl.xsl



Attention: The customizations you made by using this extension point will affect all WebHelp transformations. If you want to have a customization that is only available for a certain plugin, please use the [Overriding the XSLT Stylesheet from a Customized ANT Build File](#) method.

XSLT-Parameter Extension Points

In case your customization stylesheet declares one or more XSLT parameters and you want to control their value from the transformation scenario, you can use one of the following XSLT parameter extension points:

com.oxygenxml.webhelp.xsl.dita2webhelp.param

Use this extension point to pass parameters to the stylesheet specified using the [com.oxygenxml.webhelp.xsl.dita2webhelp extension point](#).

com.oxygenxml.webhelp.xsl.createMainFiles.param

Use this extension point to pass parameters to the stylesheet specified using the [com.oxygenxml.webhelp.xsl.createMainFiles extension point](#).

Overriding the XSLT Stylesheet from a Customized Ant Build File

This method is useful if you want to create a customization that is only available for a certain DITA-OT extension plugin. This method implies that you create a DITA-OT plugin that defines a custom `transtype`. From the Ant target associated with the plugin `transtype`, you will specify a custom XSLT stylesheet. This customized XSLT stylesheet will import the original WebHelp stylesheet and will also add some customization templates.

If you want to use this method, you need to create a DITA-OT extension plugin, as in the following sample procedure for customizing WebHelp Responsive output:

1. In the `DITA_OT_DIR\plugins\` folder, create a folder for this plugin (for example, `com.oxygenxml.webhelp.responsive.custom`).
2. Create a **plugin.xml** file (in the folder you created in step 1) that specifies a new DITA-OT `transtype` and the build file associated with the plugin. For example:

```
<plugin id="com.oxygenxml.webhelp.responsive.customization">
  <feature extension="dita.conductor.target.relative" file="integrator.xml"/>
  <transtype name="webhelp-responsive-custom" extends="webhelp-responsive"
    desc="WebHelp Responsive Customization"/>
</plugin>
```

3. Create the **integrator.xml** file that will import the actual plugin Ant build file.

```
<project basedir="." name="Webhelp Responsive Customization">
  <import file="build.xml"/>
</project>
```

4. Create the **build.xml** file that overrides the value of properties associated with the XSLT stylesheets used to produce HTML files. The following two Ant properties can be overridden to specify your customization stylesheets:

args.wh.xml

Override this property if you want to customize the XSLT stylesheet used to produce an HTML file for each topic.

args.create.main.files.xml

Override this property if you want to customize the XSLT stylesheet used to produce the main HTML file.

For customizing the WebHelp Responsive, the build file should look like:

```
<project basedir="." name="Webhelp Responsive Customization">
  <target name="dita2webhelp-custom">
    <!--
      Override this property if you want to customize the XSLT stylesheet
      used to produce an HTML file for each topic
    -->
    <property
      name="args.wh.xml"
      value="${dita.plugin.com.oxygenxml.webhelp.responsive.custom.dir}
              /xsl/dita2webhelpCustom.xml"/>

    <!--
      Override this property if you want to customize the XSLT stylesheet
      used to produce the main HTML file.
    -->
    <property
      name="args.create.main.files.xml"
      value="${dita.plugin.com.oxygenxml.webhelp.responsive.custom.dir}
              /xsl/createMainFilesCustom.xml"/>

    <!--
      Depending on which version of WebHelp you want to customize,
      you need to delegate to different build targets:
      * dita2webhelp-responsive - when you are customizing the Webhelp Responsive
      * dita2webhelp-mobile - when you are customizing the Webhelp Mobile
      * dita2webhelp - when you are customizing the Webhelp Classic
    -->
    <antcall target="dita2webhelp-responsive"/>
  </target>
</project>
```



Note: Depending on which version of WebHelp you want to customize, you need to specify one of the following build targets:

- **dita2webhelp-responsive** - To customize WebHelp Responsive (with or without feedback) output.
- **dita2webhelp-mobile** - To customize WebHelp Classic Mobile output.
- **dita2webhelp** - To customize WebHelp Classic (with or without feedback) output.

5. Create an **xsl** directory in the plugin customization directory (that you created in step 1) to store the customized XSLT stylesheets.

Customizing the XSLT Stylesheet to Produce an HTML File for Each Topic

If you want to produce an HTML file for each topic (defined with the **args.wh.xml** property), create a **dita2webhelpCustom.xml** stylesheet and save it in the newly created **xsl** directory. This stylesheet will import the original stylesheet and will contain the customization templates. It should look like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:math="http://www.w3.org/2005/xpath-functions/math"
  exclude-result-prefixes="xs math"
  version="2.0">
  <!--
    Import the original stylesheet used to produce an HTML file for each topic.
  -->
  <xsl:import href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/
                  dita/responsive/dita2webhelp.xml"/>

  <!--
    Please add your customization templates here.
  -->
</xsl:stylesheet>
```

Depending on which version of WebHelp you want to customize, you need to import one of the following XSLT stylesheets:

- **dita2webhelp-responsive** (WebHelp Responsive) -

```
<xsl:import
  href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/responsive/dita2webhelp.xml"/>
```
- **dita2webhelp-mobile** (WebHelp Classic Mobile) -


```
<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/mobile/dita2webhelp.xsl"/>
```

- **dita2webhelp** (WebHelp Classic) -

```
<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/desktop/dita2webhelp.xsl"/>
```

Customizing the XSLT Stylesheet to Produce the Main HTML File

If you just want to produce a main HTML file (defined with the **args.create.main.files.xsl** property), you need to create a **createMainFilesCustom.xsl** stylesheet and save it in the newly created **xsl** directory. This stylesheet will import the original stylesheet and will contain the customization templates. It should look like:

```
<?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"
  xmlns:math="http://www.w3.org/2005/xpath-functions/math"
  exclude-result-prefixes="xs math"
  version="2.0">
  <!--
    Import the original stylesheet used to produce the main HTML file.
  -->
  <xsl:import href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/
    responsive/createMainFiles.xsl"/>
  <!--
    Please add your customization templates here.
  -->
</xsl:stylesheet>
```

Depending on which version of WebHelp you want to customize, you need to import one of the following XSLT stylesheets:

- **dita2webhelp-responsive** (WebHelp Responsive) -

```
<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/responsive/createMainFiles.xsl"/>
```

- **dita2webhelp-mobile** (WebHelp Classic Mobile) -

```
<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/mobile/createMainFiles.xsl"/>
```

- **dita2webhelp** (WebHelp Classic) -

```
<xsl:import
href="plugin:com.oxygenxml.dita-ot.plugin.webhelp:xsl/dita/desktop/createMainFiles.xsl"/>
```

Related information

[\[DITA-OT 2.3\] XSLT-Import Extension Points](#)

[\[DITA-OT 2.3\] XSLT-Parameter Extension Points](#)

Removing the *Previous/Next* Links from WebHelp Classic Output

The **Previous** and **Next** links that are created at the top area of each WebHelp Classic page can be hidden with a CSS code.

To remove these links from WebHelp Classic output, 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.

Search Engine Optimization for DITA WebHelp

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 fewer 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 for Oxygen XML Developer plugin to generate the `sitemap.xml` file.

- `webhelp.sitemap.change.frequency` - how frequently the WebHelp pages are likely to change (accepted values are: `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, and `never`)
- `webhelp.sitemap.priority` - the priority of each page (value ranging from 0.0 to 1.0)

3. 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.

Support for Right-to-Left (RTL) Oriented Languages for DITA WebHelp

To activate support for RTL (right-to-left) 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 Classic 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 for this value in the `context-help-map.xml` mapping file and load the corresponding 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`.

- `appname` - You can use this parameter in conjunction with `contextId` to search for this value in the corresponding [appname attribute value](#) in the mapping file.

```
http://localhost/webhelp/index.html?contextId=topicID&appname=myApplication
```

- `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`.

Related information

[Context-Sensitive WebHelp Classic System](#) on page 632

Context-Sensitive WebHelp Classic 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.

Generating Context-Sensitive Help

When WebHelp Classic output is generated by Oxygen XML Developer plugin, the transformation process produces an XML mapping file called `context-help-map.xml` and copies it in the output folder of the transformation. This XML file maps an ID to a corresponding HTML page through an `appContext` element, as in the following example:

```
<map productID="oxy-webhelp" productVersion="1.1">
  <appContext helpID="myapp-functionid1" path="tasks/app-help1.html"/>
  <appContext helpID="myapp-functionid2" path="tasks/app-help1.html"/>
</map>
```

The possible attributes are as follows:

helpID

A Unique ID provided by a topic from two possible sources (`resourceid` element or `id` attribute):

resourceid

The `resourceid` element is mapped into the `appContext` element and can be specified in either the `topicref` within a DITA map or in a `prolog` within a DITA topic. The `resourceid` element accepts the following attributes:

- **appname** - A name for the external application that references the topic. If this attribute is not specified, its value is considered to be empty ("").
- **appid** - An ID used by an application to identify the topic.
- **id** - Specifies a value that is used by a specific application to identify the topic, but this attribute is ignored if an **appid** attribute is used.



Note: Multiple **appid** values can be associated with a single **appname** value (and multiple **appname** values can be associated with a single **appid** value), but the values for both attributes work in combination to specify a specific ID for a specific application, and therefore each combination of values for the **appid** and **appname** attributes should be unique within the context of a single root map. For example, suppose that you need two different functions of an application to both open the same WebHelp page.

Example: resourceid Specified in a DITA Map

The **resourceid** element can be specified in a **topicmeta** element within a **topicref**.

```
<map title="App Help">
  <topicref href="app-help1.dita" type="task">
    <topicmeta>
      <resourceid appname="myapp" appid="functionid1"/>
      <resourceid appname="myapp" appid="functionid2"/>
    </topicmeta>
  </topicref>
</map>
```

Example: resourceid Specified in a DITA Topic

The **resourceid** element can be specified in a **prolog** element within a DITA topic.

```
<task id="app-help1">
  <title>My App Help</title>
  <prolog>
    <resourceid appname="myapp" appid="functionid1"/>
    <resourceid appname="myapp" appid="functionid2"/>
  </prolog>
  ...
</task>
```

For more information about the **resourceid** element, see [DITA Specifications: <resourceid>](#).

id

If a **resourceid** element is not declared in the DITA map or DITA topic (as described above), the **id** attribute that is set on the topic root element is mapped into the **appContext** 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 and the help system will not work properly.

path

The path to a corresponding WebHelp page. This path is relative to the location of the **context-help-map.xml** mapping file.

productID (Applicable only if you are using the WebHelp Classic with Feedback transformation scenario)

The ID of the product for your documentation project.

productVersion (Applicable only if you are using the WebHelp Classic with Feedback transformation scenario)

The version of the product for your documentation project.

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
```



Tip: You can use an *anchor* in the `contextId` parameter to jump to a specific section in a document. For example, `contextId=topicID#anchor`.

Context-Sensitive Queries

You can use the URL field in your browser to search for topics in a context-sensitive WebHelp system with the assistance of the following parameters:

- `contextId` - The WebHelp JavaScript engine will look for this value in the `context-help-map.xml` mapping file and load the corresponding 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`.

- `appname` - You can use this parameter in conjunction with `contextId` to search for this value in the corresponding [appname attribute value](#) in the mapping file.

```
http://localhost/webhelp/index.html?contextId=topicID&appname=myApplication
```

Related information

[WebHelp Classic Runtime Additional Parameters](#) on page 631

Publishing WebHelp Classic Output on a SharePoint Site

Since WebHelp output must be published locally, on the same machine where the WebHelp process is running, to publish your files directly to a SharePoint library you need to map a network drive to connect to SharePoint and change your file extensions to `.aspx`, as described in the steps below.

To publish WebHelp Classic output on a SharePoint site, follow this procedure:

1. Map a network drive to connect to your SharePoint library. For more information, see: <https://support.microsoft.com/en-us/kb/2616712>.
2. To allow browsers to open your published files (rather than downloading them), you need to change the file extensions from `.html` to `.aspx`. Fortunately, this can be done in the transformation scenario.
 - a. [Edit the WebHelp transformation scenario](#) and open the **Parameters** tab.
 - a. For a DITA transformation, set the `args.outext` parameter to `.aspx`.
 - b. For a DocBook transformation, set the `html.ext` parameter to `.aspx`.
 - b. Run the transformation scenario.

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 Developer plugin license.

The WebHelp output files created with the Oxygen XML WebHelp plugin are the same as the output files produced when you run DITA or DocBook to WebHelp transformation scenarios from within Oxygen XML Developer plugin.

When an automated process is required due to the amount of output needed, do the following:

1. Install the Oxygen XML WebHelp plugin.
2. Acquire a Oxygen XML WebHelp license from https://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-responsive** - The transformation that produces *WebHelp Responsive* and *WebHelp Responsive with Feedback* output for desktop and mobile devices.
- **webhelp** - The transformation that produces *WebHelp Classic* output for desktop.
- **webhelp-feedback** - The transformation that produces feedback-enabled *WebHelp Classic with Feedback* for desktop.
- **webhelp-mobile** - The transformations that produces *WebHelp Classic Mobile* output for mobile devices.

Integrating the Oxygen XML WebHelp Plugin with the DITA Open Toolkit

The requirements of the Oxygen XML WebHelp plugin for the DITA Open Toolkit are as follows:

- Java Virtual Machine 1.6 or newer if you want to use DITA-OT 1.8.5 or Java Virtual Machine 1.8 or newer if you want to use DITA-OT 2.3.3.
- DITA Open Toolkit 1.8.5 or 2.3.3 (includes Saxon 9.x libraries).



Note: The Oxygen XML WebHelp Plugin has been tested with these two specific versions (**DITA-OT 1.8.5** or **DITA-OT 2.3.3**) and therefore they are the recommended versions.

To integrate the Oxygen XML WebHelp plugin with the DITA Open Toolkit, follow these steps:

1. Download and install a [Java Virtual Machine](#) version compatible with the DITA-OT version.
2. Download and unpack the [DITA Open Toolkit](#) version 1.8.5 or 2.3.3.
3. Go to [Oxygen XML WebHelp website](#), download the latest installation kit, and unzip it.
4. Copy all plugin directories from the unpacked archive to the `plugins` directory of the DITA OT distribution. This is necessary to enable certain functionality. For example, the `com.oxygenxml.highlight` directory adds syntax highlight capabilities to your WebHelp output for codeblock sections that contain source code snippets (XML, Java, JavaScript).
5. If you have not already done so, [copy your license key into a licensekey.txt file](#) and place it in the `DITA_OT_DIR/plugins/com.oxygenxml.webhelp` directory.
6. In the home directory of the DITA Open Toolkit, run `ant -f integrator.xml`.

Related tasks

[Licensing the Oxygen XML WebHelp Plugin for DITA OT](#) on page 654

[Upgrading the Oxygen XML WebHelp Plugin for DITA OT](#) on page 654

Licensing the Oxygen XML WebHelp Plugin for DITA OT

To register the license for the Oxygen XML WebHelp plugin for the DITA Open Toolkit, follow these steps:

1. Open the `DITA_OT_DIR/plugins/com.oxygenxml.webhelp` directory and create a file called `licensekey.txt`.
2. In this file, copy your license key that you purchased for your Oxygen XML WebHelp plugin.

The WebHelp transformation process reads the Oxygen XML WebHelp license key from this file. In case the file does not exist, or it contains an invalid license, an error message will be displayed.

Related tasks

[Integrating the Oxygen XML WebHelp Plugin with the DITA Open Toolkit](#) on page 654

[Upgrading the Oxygen XML WebHelp Plugin for DITA OT](#) on page 654

Upgrading the Oxygen XML WebHelp Plugin for DITA OT

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.xml`, `oxygen_custom_html.xml`) and directories (`com.oxygenxml.highlight`, `com.oxygenxml.media`, `com.oxygenxml.webhelp`).
2. Go to [Oxygen XML WebHelp website](#), download the latest installation kit, and unzip it.

3. Copy all plugin directories from the unpacked archive to the `plugins` directory of the DITA OT distribution. This is necessary to enable certain functionality. For example, the `com.oxygenxml.highlight` directory adds syntax highlight capabilities to your WebHelp output for `codeblock` sections that contain source code snippets (XML, Java, JavaScript).
4. In the home directory of the DITA Open Toolkit, run `ant -f integrator.xml`.

Related tasks

[Integrating the Oxygen XML WebHelp Plugin with the DITA Open Toolkit](#) on page 654

[Licensing the Oxygen XML WebHelp Plugin for DITA OT](#) on page 654

Running an External DITA Transformation Using the Oxygen XML WebHelp Plugin

There are two main ways to run a DITA to WebHelp ([webhelp-responsive](#), [webhelp](#), [webhelp-feedback](#), [webhelp-mobile](#)) transformation using the Oxygen XML WebHelp plugin:

Startup Script from WebHelp Plugin Method

The first method involves running the `ditawebhelp.bat` or `ditawebhelp.sh` startup script that comes bundled in the WebHelp plugin folder:

- `WEBHELP_PLUGIN_DIR\ditawebhelp.bat` script file (Windows based systems)
- `WEBHELP_PLUGIN_DIR\ditawebhelp.sh` script file (Unix/Linux based systems)

Before using the script to generate output, you must customize it to specify the paths to the Java VM, DITA Open Toolkit, 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 scripts reference the `dost-patches-DITA-1.8.jar` JAR file containing DITA OT 1.8.5-specific patches. If you use DITA OT 1.7, please update that reference to `dost-patches-DITA-1.7.jar`. If you use DITA OT 2.3.3, no patches are needed, so just remove the reference.

- `TRANSTYPE` - Specifies the type of the transformation you want to apply. You can set it to `webhelp-responsive`, `webhelp`, `webhelp-feedback` or `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.

Startup Script from DITA OT Distribution Method

The second method involves using the startup script that comes bundled with each DITA OT distribution.

DITA OT 1.8.5

For DITA Open Toolkit 1.8.5, the general instructions for running the startup script can be found here: <http://www.dita-ot.org/1.8/readme/building-output-using-cl-tool.html>.

DITA OT 2.3.3

For DITA Open Toolkit 2.3.3, the general instructions for running the startup script can be found here: <http://www.dita-ot.org/2.3/parameters/dita-command-arguments.html>.

DITA OT Startup Script Examples:

- `DITA_OT_DIR\bin\dita.bat -i path_to_input.ditamap -f webhelp` (Windows based systems)
- `DITA_OT_DIR\bin\dita -i path_to_input.ditamap -f webhelp` (Unix/Linux based systems)



Note: For the `-f` format (`-f`) argument, you use the following transformation types: `webhelp-responsive`, `webhelp`, `webhelp-feedback` or `webhelp-mobile`.

The downside of this approach is the fact that you will lose certain small fixes and patches that Oxygen XML Developer plugin adds to the automated DITA OT processing. For example, you may lose the ability to process DITA topics that contain entity references inside them.

Related tasks

[Integrating the Oxygen XML WebHelp Plugin with the DITA Open Toolkit](#) on page 654

Additional Oxygen XML WebHelp Plugin Parameters for DITA

You can append the following parameters to the command line that runs the transformation:

-Dwebhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

-Dwebhelp.custom.resources

The file path to a directory that contains resources files. All files from this directory will be copied to the root of the WebHelp output.

-Dwebhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

-Dwebhelp.footer.file (not available for WebHelp Responsive systems)

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 excerpt 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 (not available for WebHelp Responsive systems)

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 Developer plugin footer is inserted in each WebHelp page.

-Dwebhelp.google.search.results

A file path 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.google.search.script

A file path that specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google.

-Dwebhelp.logo.image.target.url (not available for WebHelp Classic Mobile systems)

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.logo.image (not available for WebHelp Classic Mobile systems)

Specifies a path to an image displayed as a logo in the left side of the output header.

-Dwebhelp.product.id (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: < > / \ ' () { } = ; * % + &.

-Dwebhelp.product.version (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: < > / \ ' () { } = ; * % + &.

-Dwebhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine that Oxygen XML Developer plugin uses for indexing Japanese content in the WebHelp pages.

-Dwebhelp.search.ranking

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.show.changes.and.comments

When set to `yes`, user comments, replies to comments, and tracked changes are published in the WebHelp output. The default value is `no`.

-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). For example, 0.3, 0.5, or 0.8. 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.skin.css (available only for WebHelp Classic and WebHelp Classic with Feedback systems)

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.

Parameters Specific to WebHelp Responsive Output (available only for WebHelp Responsive and WebHelp Responsive with Feedback systems)

-Dwebhelp.fragment.after.body

In the generated output it displays a given XHTML fragment after the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.after.logo_and_title

In the generated output it displays a given XHTML fragment after the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.after.main.page.search

In the generated output it displays a given XHTML fragment after the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.after.toc_or_tiles

In the generated output it displays a given XHTML fragment after the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.after.top_menu

In the generated output it displays a given XHTML fragment after the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.before.body

In the generated output it displays a given XHTML fragment before the page body. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.before.logo_and_title

In the generated output it displays a given XHTML fragment before the logo and title. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.before.main.page.search

In the generated output it displays a given XHTML fragment before the search field. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.before.toc_or_tiles

In the generated output it displays a given XHTML fragment before the table of contents or tiles in the main page. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.before.top_menu

In the generated output it displays a given XHTML fragment before the top menu. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.footer

In the generated output it displays a given XHTML fragment as the page footer. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.head

In the generated output it displays a given XHTML fragment as a page header. The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.fragment.welcome

In the generated output it displays a given XHTML fragment as a welcome message (or title). The value of the parameter can be either an XHTML fragment or a path to a file that contains a well-formed XHTML fragment.

-Dwebhelp.merge.nested.topics.related.links

Specifies if the related links from nested topics will be merged with the links in the parent topic. Thus the links will be moved from the topic content to the related links component and all of the links from the same group (for example, *Related Tasks*, *Related References*, *Related Information*) are merged into a single group. The default value is **yes**.

-Dwebhelp.show.breadcrumb

Specifies if the breadcrumb component will be presented in the output. The default value is **yes**.

-Dwebhelp.show.child.links

Specifies if child links will be generated in the output for all topics that have subtopics. The default value is **no**.

-Dwebhelp.show.indexterms.link

Specifies if an icon that links to the index will be presented in the output. The default value is **yes**.

-Dwebhelp.show.main.page.tiles

Specifies if the tiles component will be presented in the main page of the output. For a *tree* style layout, this parameter should be set to **no**.

-Dwebhelp.show.main.page.toc

Specifies if the table of contents will be presented in the main page of the output. The default value is **yes**.

-Dwebhelp.show.navigation.links

Specifies if navigation links will be presented in the output. The default value is **yes**.

-Dwebhelp.show.print.link

Specifies if a print link or icon will be presented within each topic in the output. The default value is **yes**.

-Dwebhelp.show.side.toc

Specifies if a side table of contents will be presented on the right side of each topic in the output. The default value is **yes**.

-Dwebhelp.show.top.menu

Specifies if a menu will be presented at the topic of the main page in the output. The default value is **yes**.

-Dwebhelp.top.menu.depth

Specifies the maximum depth level of the topics that will be included in the top menu. The default value is **2**.



Note: Note that the `fix.external.refs.com.oxygenxml` parameter is not supported in Oxygen XML WebHelp plugin. This parameter is normally used to specify whether or not the application tries to fix such references in a temporary files folder before the DITA Open Toolkit is invoked on the fixed references.

Further Customization

If you need to further customize the transformation process, you can 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.csspath` parameter, use:

```
-Dargs.csspath=[CSS_FILE_PATH]
```

where `[CSS_FILE_PATH]` is the location of the directory that contains the CSS file.

Related information

[DITA OT Parameter Reference](#)

Configuring the Comment Database for DITA WebHelp Systems with Feedback

If you run the [webhelp-responsive](#) or [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.

Building Oxygen XML WebHelp Output on Jenkins

This procedure assumes that you have already [integrated, configured, and registered the WebHelp plugin with the DITA Open Toolkit](#).

To integrate WebHelp output with the Jenkins continuous integration tool, follow these steps:

1. Create a Maven project to incorporate the DITA-OT that already integrates Oxygen XML WebHelp.
2. Go to the root of your Maven project and edit the `pom.xml` file to include the following fragment:

```
<properties>
  <oxygen-webhelp>${basedir}/tools/DITA-OT1.8.5/
    plugins/com.oxygenxml.webhelp</oxygen-webhelp>
</properties>

<plugin>
  <artifactId>exec-maven-plugin</artifactId>
  <groupId>org.codehaus.mojo</groupId>
  <executions>
    <execution><!-- Run our version calculation script -->
      <id>Version Calculation</id>
      <phase>generate-sources</phase>
      <goals>
        <goal>exec</goal>
      </goals>
      <configuration>
        <executable>${oxygen-webhelp}/ditawebhelp.bat</executable>
      </configuration>
    </execution>
  </executions>
</plugin>
```



Note: In the fragment above it is assumed that you are using DITA-OT version 1.8.5. If you are using another version, please adjust the path accordingly.

3. Go to the Jenkins top page and [create a new Jenkins job](#). Configure this job to suit your particular requirements, such as the build frequency and location of the Maven project.

Building Oxygen XML WebHelp Output on Travis CI

This topic assumes you have a DITA project hosted on a GitHub public or private repository.

The goal of this tutorial is to help you setup a Travis continuous integration job that automatically publishes your DITA project to [GitHub pages](#) after every commit. The published website will contain a feedback link on each page that would allow a contributor to easily suggest changes to the documentation by creating a pull request on GitHub with just a few clicks.

Enable the Travis CI Build

1. [Sign in to Travis CI](#) with your GitHub account, accepting the GitHub [access permissions confirmation](#).
2. Once you are signed in, and you have synchronized your GitHub repositories, go to your [profile page](#) and enable Travis CI for the repository you want to build.

Configure the Travis CI Build in your GitHub Project

1. Checkout your GitHub project locally.
2. Copy the `.travis` folder from [here](#) to the root directory of your project.
3. In the root of your GitHub project, add a file called `.travis.yml` with the following content:

```
language: dita
install:
  - echo "Installed"
script:
  - sh .travis/publish.sh
after_success:
  - sh .travis/deploy.sh
env:
  global:
    - DITAMAP=/path/to/your/ditamap/file
    - DITAVAL=/path/to/your/ditaval/file
    - ANT_OPTS=-Xmx1024M
```



Note: Replace `/path/to/your/ditamap/file` and `/path/to/your/ditaval/file` with the appropriate paths to your DITA Map and ditaval files.

4. Create a GitHub personal access token by following [this procedure](#).
5. [Define an environment variable in the repository settings](#) that has the name `GH_TOKEN` and the value equal with the GitHub personal access token created earlier.

Register Your License Key

1. Edit your `.gitignore` file (or create it if it does not already exist) and add the following line:

```
licenseKey.txt
```

2. Copy your WebHelp license to the root of your GitHub project in a file called `licenseKey.txt`.



Important: The `licenseKey.txt` file should not be committed to GitHub as it contains a license key that is issued only to you.

3. [Encrypt the license key file](#) and **add** it to the `.travis.yml` configuration file. This way only the Travis CI server will be able to decrypt it during the build process.

Commit to GitHub

1. Commit the following files and folders and push the commit to GitHub:

```
git add .gitignore licenseKey.txt.enc .travis.yml .travis/
git commit -m "Set up the Travis CI publishing system"
git push
```

2. Create a `gh-pages` branch in your GitHub project where the WebHelp output will be published. You can follow the procedure [here](#).

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 Classic* output for desktop.
- **webhelp-feedback** - The transformation that produces feedback-enabled *WebHelp Classic with Feedback* for desktop.
- **webhelp-mobile** - The transformations that produces *WebHelp Classic Mobile* output for mobile devices.
- **webhelp-responsive** - The transformation that produces *WebHelp Responsive* and *WebHelp Responsive with Feedback* output for desktop and mobile devices.

Integrating the Oxygen XML WebHelp Plugin with the DocBook XSL Distribution

The WebHelp plugin transformations run as an Ant build script. The requirements are:

- Ant 1.8 or later
- Java Virtual Machine 1.6 or later
- DocBook XSL 1.78.1 or 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 the DocBook XSL distribution on your computer.
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

To register the license for the Oxygen XML WebHelp plugin for the DocBook XSL distribution, follow these steps:

1. Create a `licenseKey.txt` file in the `com.oxygenxml.webhelp` subdirectory of the DocBook XSL directory.
2. In this file, copy the license key that you purchased for your Oxygen XML WebHelp plugin.

The WebHelp transformation process reads the Oxygen XML Developer plugin license key from this file. If the file does not exist, or it contains an invalid license, an error message is displayed.

Upgrading the Oxygen XML WebHelp Plugin for DocBook

To upgrade your Oxygen XML WebHelp plugin for DocBook, follow these steps:

1. Navigate to the `[OXYGEN_INSTALL_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_INSTALL_DIR]/frameworks/docbook/xsl`.

Running an External DocBook Transformation Using the WebHelp Plugin

To run a DocBook to WebHelp ([webhelp](#), [webhelp-feedback](#), [webhelp-mobile](#)) transformation using the Oxygen XML WebHelp plugin, use:

- The `docbook.bat` script file for Windows based systems.
- The `docbook.sh` script file for Unix/Linux based systems.



Note: You can call these files in an automated process or from the command line.

The `docbook.bat` and the `docbook.sh` files are located in the home directory of the Oxygen XML WebHelp Plugin. Before using them to generate an WebHelp system, customize them to match the paths to the JVM, DocBook XSL distribution and Saxon engine, and also to set the transformation type. To do this, open a script file and edit the following variables:

- `JVM_INSTALL_DIR` - Specifies the path to the Java Virtual Machine installation directory on your disk.
- `ANT_INSTALL_DIR` - Specifies the path to the installation directory of Ant.
- `SAXON_6_DIR` - Specifies the path to the installation directory of Saxon 6.5.5.
- `SAXON_9_DIR` - Specifies the path to the installation directory of Saxon 9.1.0.8.
- `DOCBOOK_XSL_DIR` - Specifies the path to the installation directory of the DocBook XSL distribution.
- `TRANSTYPE` - Specifies the type of the transformation you want to 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 can append the following parameters to the command line that runs the transformation:

-Dwebhelp.copyright

Adds a small copyright text that appears at the end of the *Table of Contents* pane.

-Dwebhelp.favicon

The file path that points to an image to be used as a *favicon* in the WebHelp output.

-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 excerpt 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 Developer plugin footer is inserted in each WebHelp page.

-Dwebhelp.product.id (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: `<` `>` `/` `\` `'` `(` `)` `{` `}` `=` `;` `*` `%` `+` `&`.

-Dwebhelp.product.version (available only for Feedback-enabled systems)

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.



Restriction: The following characters are not allowed in the value of this parameter: `< > / \ ' ( ) { } = ; * % + &`.

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
```

Configuring the Comment Database for DocBook WebHelp Classic with Feedback

If you run the [webhelp-feedback](#) transformation using the WebHelp plugin, you need to configure the database that holds the user comments. The instructions for configuring the database are presented in the `installation.html` file, located at: `[OUTPUT_DIR]/oxygen-webhelp/resources/installation.html`. The `installation.html` file is created by the transformation process.

Chapter 10

Querying Documents

Topics:

- [Working with XPath Expressions](#)
- [Working with XQuery](#)

This chapter includes information about running XPath expressions and working with the XQuery language. Each mechanism has some useful features and tools to help you use them, and Oxygen XML Developer plugin has some unique methods of handling each mechanism.

Working with XPath Expressions

XPath is a language for addressing specific parts of an XML document. XPath, such as 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 various 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>.

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 can use the `doc()` function to specify the source file for which the expressions are executed. When you connect to a database, the expressions are executed over that database. If you are using the **XPath/XQuery Builder** view and the current file is an XSLT document, Oxygen XML Developer plugin executes the expressions over the XML document in the associated scenario.

If the view is not displayed, it can be opened from the **Window > Show View** menu.

The upper part of the view contains the following actions:

XPath version chooser drop-down menu

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 Developer 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 Developer plugin returns a compilation error.



Execute XPath button

Use 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](#).



Favorites button

Allows you to save certain expressions that you can later reuse. To add an expression as a favorite, press the star button and enter a name for it. 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 Developer 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 cursor move** - When enabled and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed.
-  **Evaluate as you type** - When you select this option, the XPath expression you are composing is evaluated in real time.



Note: The  **Evaluate as you type** option and the automatic validation are disabled when the scope is other than **Current file**.

-  **Options** - Opens the Preferences page of the currently selected processing engine.

XPath scope menu

Oxygen XML Developer plugin allows you to define a scope for which the XPath operation will be executed. You can choose where the XPath expression will be executed:

-  **Current file** - Current selected file only.
-  **Enclosing project** - All the files of the project that encloses the current edited file.
-  **Workspace selected files** - The files selected in the workspace. The files are collected from the last selected resource provider view (**Navigator**, **Project Explorer** or **Package Explorer**).
-  **All opened files** - All files opened in the application.
-  **Opened archive** - Files open in the [Archive Browser](#) view.
-  **Working sets** - The selected working sets.

At the bottom of the scope menu the following scope configuration actions are available:

-  **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 for 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.

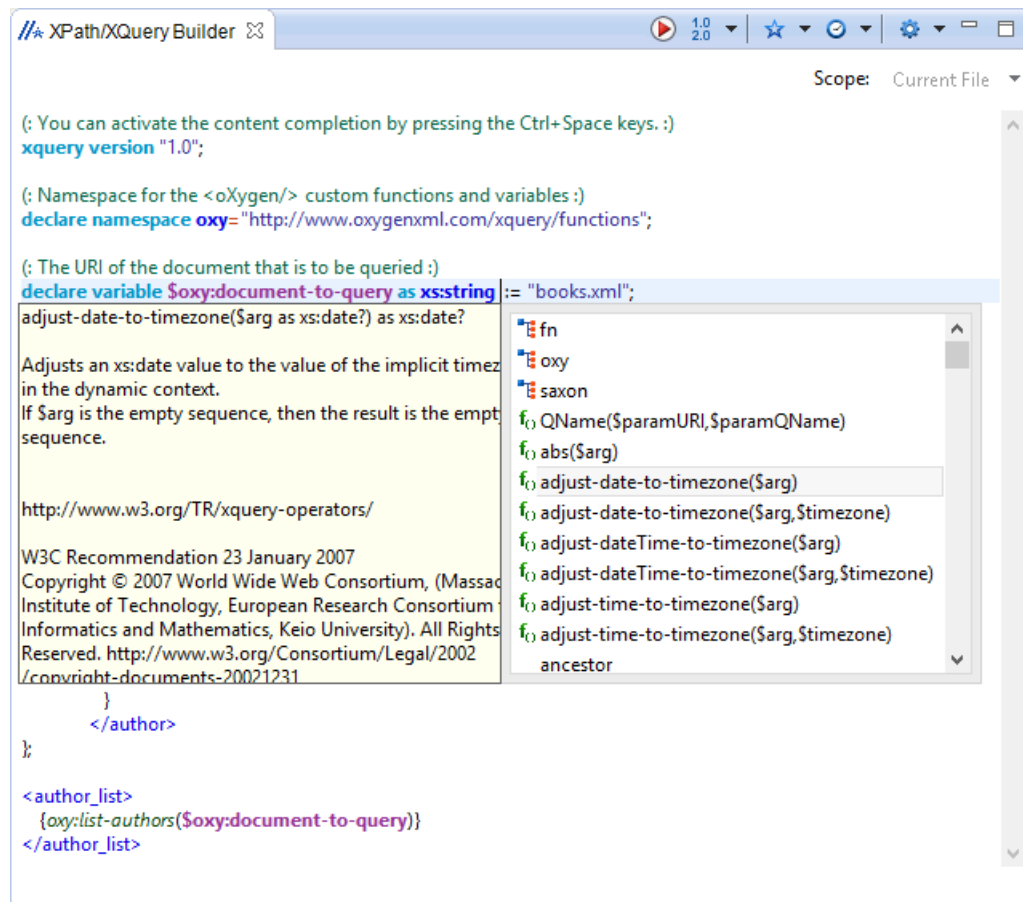


Figure 306: XPath/XQuery Builder View

When you hover your cursor over the XPath/XQuery version icon , a tooltip is displayed to let you know what engine Oxygen XML Developer plugin currently uses.

While you edit an XPath or XQuery expression, Oxygen XML Developer plugin assists you with the following features:

- **Content Completion Assistant** - It offers context-dependent proposals and takes into account the cursor position in the document you are editing. The set of functions proposed by the **Content Completion Assistant** also depends on the engine version. Select the engine version from the drop-down menu available in the toolbar.
- **Syntax Highlight** - Allows you to identify the components of an expression. To customize the colors of the components of the expression, [open the Preferences dialog box](#) and go to **Editor > 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.

Related information

[XPath Expression Results View](#) on page 667

XPath Expression Results View

When you run an XPath expression, Oxygen XML Developer plugin displays the results of its execution in the **XPath Results** view.

This view contains the following columns:

- **Description** - The result that Oxygen XML Developer plugin displays when you run an XPath expression.
- **XPath location** - The path to the matched node.
- **Resource** - The name of the document that you run the XPath expression on.

- **System ID** - The path to the document itself.
- **Location** - 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 Developer plugin displays **Not available** in the **Location** column. Oxygen XML Developer plugin displays the results in a valid XPath expression format.

```
- /node[value]/node[value]/node[value] -
```

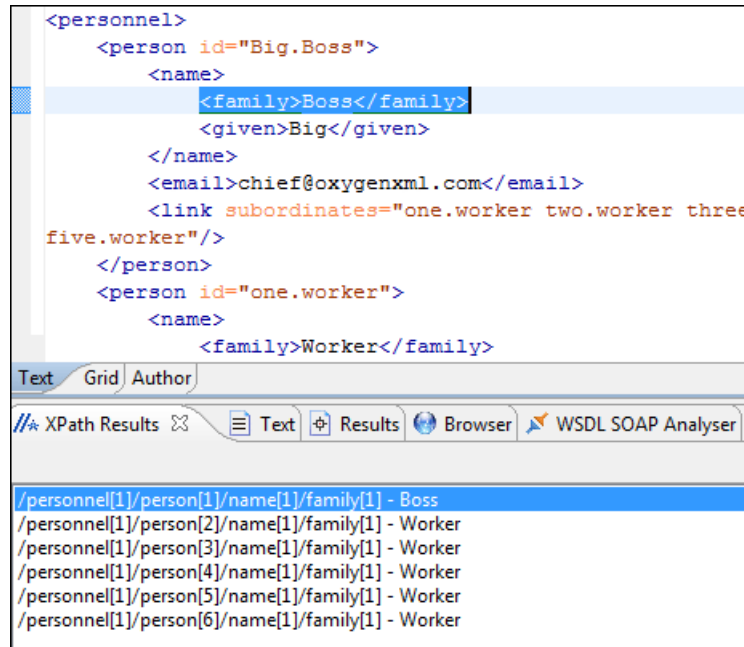


Figure 307: XPath Results Highlighted in Editor Panel with Character Precision

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 Developer plugin adds a result in the **Results View** for each `example` node found in any `sect2` node.

For example, if the result of the above XPath expression is:

```
- /chapter[1]/sect1[3]/sect2[7]/example[1]
```

it means that in the edited file, the `example` node is located in the first chapter, third section level one, seventh section level 2.

XPath and XML 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/mydocsdire?recurse=yes;select=*.xml;
  parser=ro.sync.xml.parser.CatalogEnabledXMLReader"))
```

XPath Prefix Mapping

To define default mappings between prefixes and namespace URIs go to the [XPath preferences page](#) 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 Developer plugin binds this namespace to the first free prefix from the list: `default`, `default1`, `default2`, and so on. For example, if you define the default namespace `xmlns="something"` and the prefix `default` is not associated with another namespace, you can match tags without prefix in an XPath expression typed in the XPath toolbar by using the prefix `default`. To find all the `level` elements when you define a default namespace in the root element, use this expression: `//default:level` in the XPath toolbar.

Working with 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 are 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.

XQuery Syntax Highlights 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 Developer plugin provides syntax highlight for keywords and all known XQuery functions and operators. A **Content Completion Assistant** is also available and can be activated with the **(Ctrl (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 such as 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 that 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 included 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. Otherwise, the default prefix for the Saxon namespace (`saxon`) is used.

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. Otherwise, only the function name being used.

The content completion pop-up window presents all the variables and functions from both the edited XQuery file and its imports.

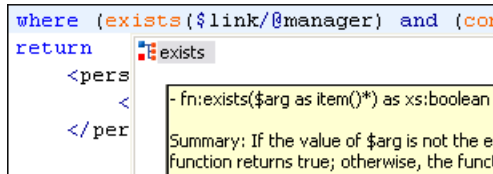


Figure 308: XQuery Content Completion

XQuery Outline View

The XQuery document structure is presented in the **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 and it allows quick access to components. By default, it is displayed on the left side of the editor. If the view is not displayed, it can be opened from the **Window > Show View** menu.

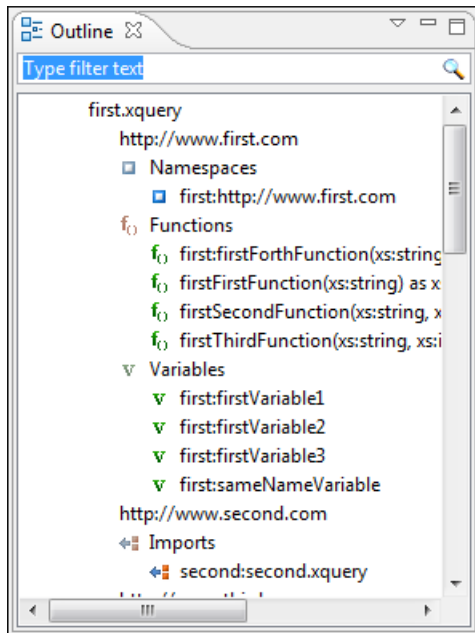


Figure 309: XQuery Outline View

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.

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.

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 can use the `doc()` function to specify the source file for which the expressions are executed. When you connect to a database, the expressions are executed over that database. If you are using the **XPath/XQuery Builder** view and the current file is an XSLT document, Oxygen XML Developer plugin executes the expressions over the XML document in the associated scenario.

If the view is not displayed, it can be opened from the **Window > Show View** menu.

The upper part of the view contains the following actions:

XPath version chooser drop-down menu

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 Developer 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 Developer plugin returns a compilation error.



Execute XPath button

Use 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](#).



Favorites button

Allows you to save certain expressions that you can later reuse. To add an expression as a favorite, press the star button and enter a name for it. 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 Developer 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 cursor move** - When enabled and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed.

-  **Evaluate as you type** - When you select this option, the XPath expression you are composing is evaluated in real time.



Note: The  **Evaluate as you type** option and the automatic validation are disabled when the scope is other than **Current file**.

-  **Options** - Opens the Preferences page of the currently selected processing engine.

XPath scope menu

Oxygen XML Developer plugin allows you to define a scope for 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 (**Navigators**, **Project Explorer** or **Package Explorer**).
-  **All opened files** - All files opened in the application.
-  **Opened archive** - Files open in the [Archive Browser](#) view.
-  **Working sets** - The selected working sets.

At the bottom of the scope menu the following scope configuration actions are available:

-  **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 for 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.

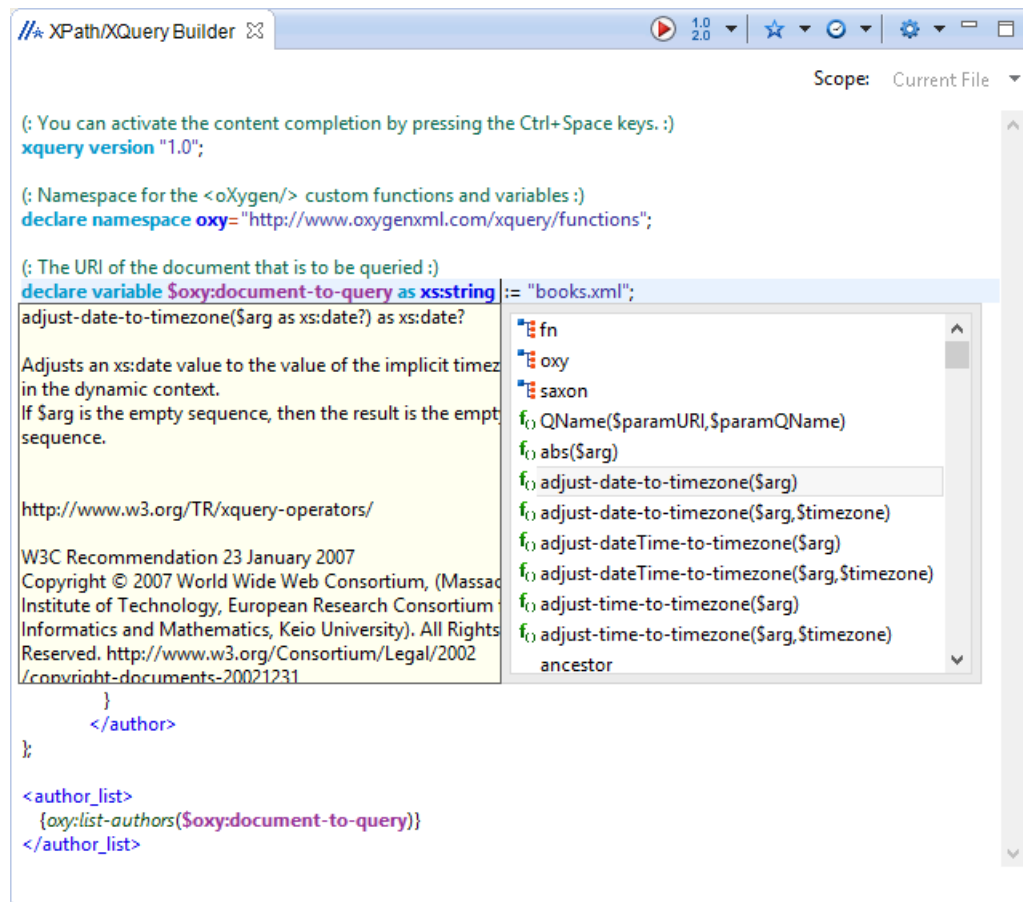


Figure 310: XPath/XQuery Builder View

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 Developer plugin currently uses.

While you edit an XPath or XQuery expression, Oxygen XML Developer plugin assists you with the following features:

- **Content Completion Assistant** - It offers context-dependent proposals and takes into account the cursor position in the document you are editing. The set of functions proposed by the **Content Completion Assistant** also depends on the engine version. Select the engine version from the drop-down menu available in the toolbar.
- **Syntax Highlight** - Allows you to identify the components of an expression. To customize the colors of the components of the expression, [open the Preferences dialog box](#) and go to **Editor > 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.

Related information

[XPath Expression Results View](#) on page 667

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 the **XSLT/XQuery Input** view. If the view is not displayed, it can be opened from the **Window > Show View** menu. The tree nodes represent the elements of the documents.

XSLT

If you click a node in the **XSLT/XQuery Input** view, 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.

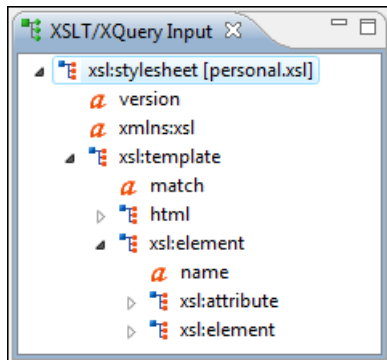


Figure 311: XSLT Input View

XSLT 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:

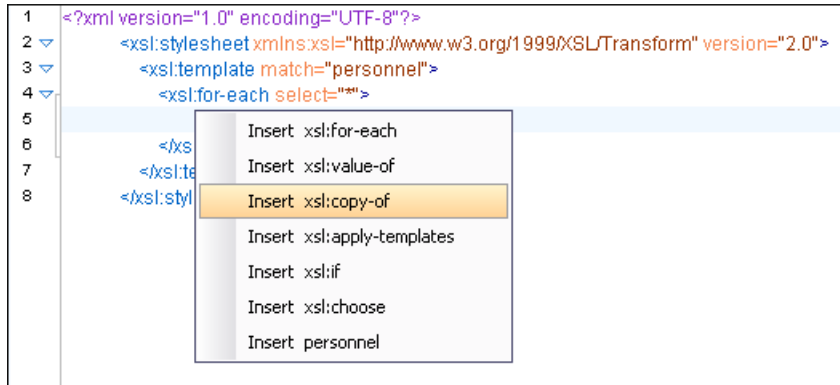


Figure 312: XSLT Input Drag and Drop Pop-up Menu

If you select **Insert xsl:copy-of** (for example), the resulting document will look like this:

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

Figure 313: XSLT Input Drag and Drop Result

XQuery

You can also use the **XSLT/XQuery Input** view to drag and drop a node into the editing area to quickly insert XQuery expressions.

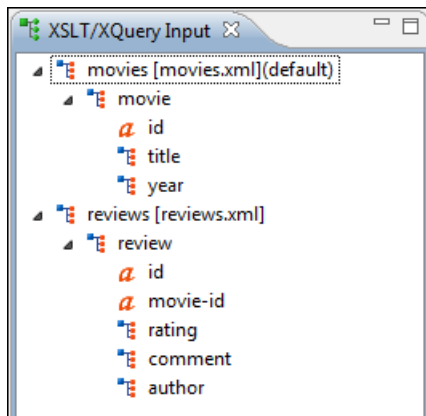


Figure 314: XQuery Input View

XQuery Example:

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>
```

```
<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, the following pop-up menu is displayed:

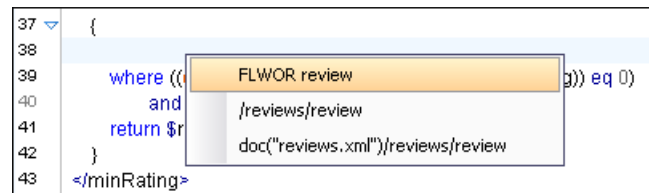


Figure 315: XQuery Input Drag and Drop Pop-up Menu

Select **FLWOR review**, the resulting document will look like this:

```

37 {
38   for $review in doc("reviews.xml")/reviews/review
39   return
40   where ({compare($rev/rating/text(), string($minRating)) eq 0}
41         and ($rev/@movie-id = $movie/@id))
42   return $rev/author
43 }

```

Figure 316: XQuery Input Drag and Drop Result

XQuery Validation

With Oxygen XML Developer plugin, you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.6.0.7 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.

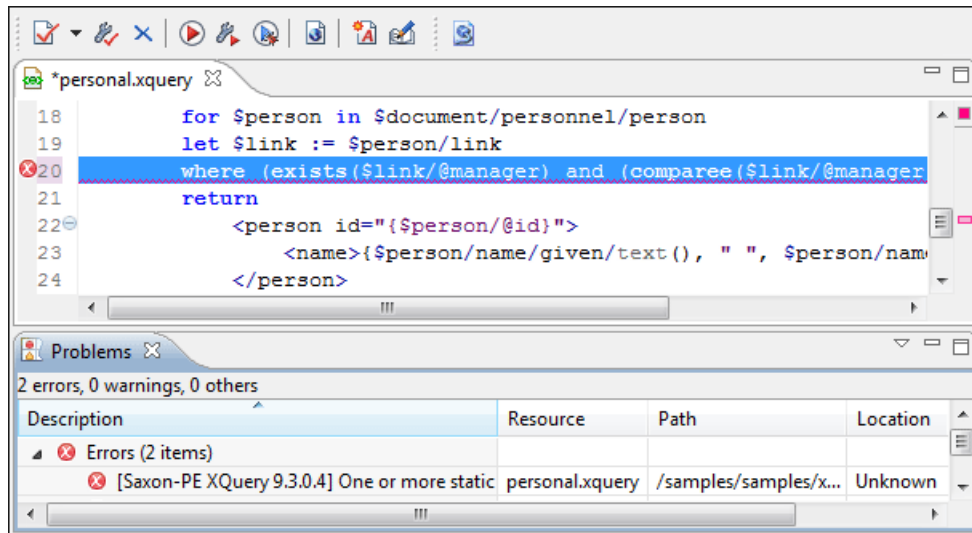


Figure 317: XQuery Validation

 **Note:** If you choose a processor that does not support XQuery validation, Oxygen XML Developer plugin displays a warning when trying to validate.

When you open an XQuery document from a connection that supports validation (for example, MarkLogic, or eXist), by default Oxygen XML Developer plugin uses this connection for validation. If you open an XQuery file using a MarkLogic connection, the validation better resolves imports.

Transforming XML Documents Using XQuery

XQuery is similar to 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 also supports XQuery 3.0 transformations.

XQuery 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 Developer plugin after configuring the environment variables.
2. *Open the Preferences dialog box* and go to **Data Sources**.
3. Click the **New** button in the **Data Sources** panel.
4. Enter a unique name for the data source.
5. Select **XQuery API for Java(XQJ)** in the **Type** combo box.

6. Press the **Add** button to add XQJ API-specific files.

You can manage the driver files using the **Add**, **Remove**, **Detect**, and **Stop** buttons.

Oxygen XML Developer plugin detects any implementation of `javax.xml.xquery.XQDataSource` and presents it in **Driver class** field.

7. Select the most suited driver in the **Driver class** combo box.
8. Click the **OK** button to finish the data source configuration.

How to Configure an XQJ Connection

The steps for configuring an XQJ connection are the following:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for this connection.
4. Select one of the previously configured [XQJ data sources](#) in the **Data Source** combo box.
5. Fill-in the connection details.

The properties presented in the connection details table are automatically detected depending on the selected data source.

6. Click the **OK** button.

Display XQuery 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 [Size limit of Sequence view option](#). The  **XQuery options** button from the **More results available** label provides a quick access to this option by opening the [XQuery preferences page](#) where the option can be modified.

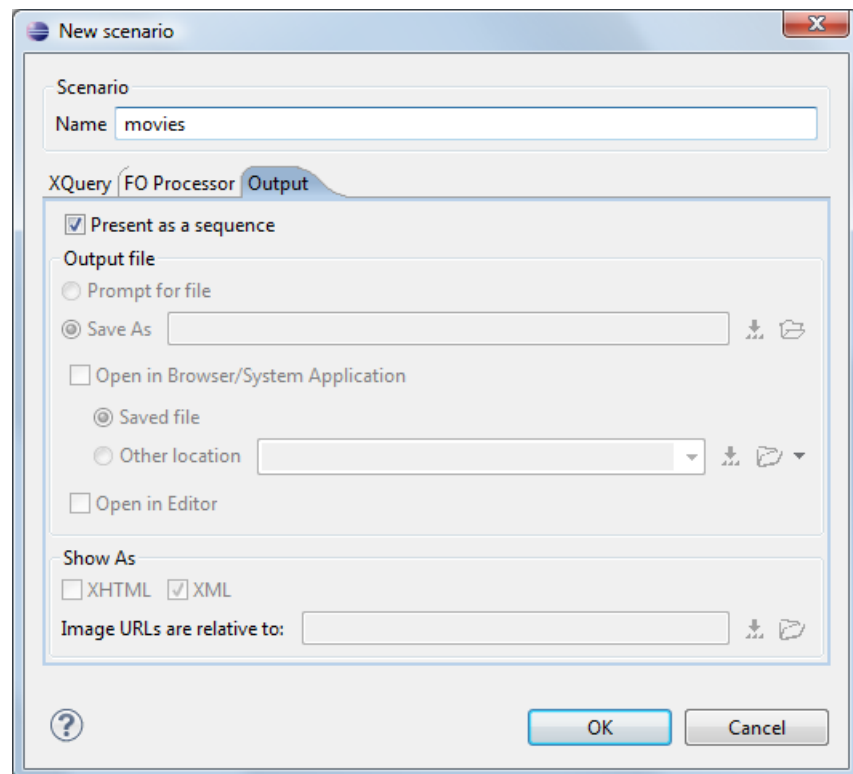


Figure 318: XQuery transformation result displayed in Sequence view

A chunk of the XQuery transformation result is displayed in the **Sequence** view.

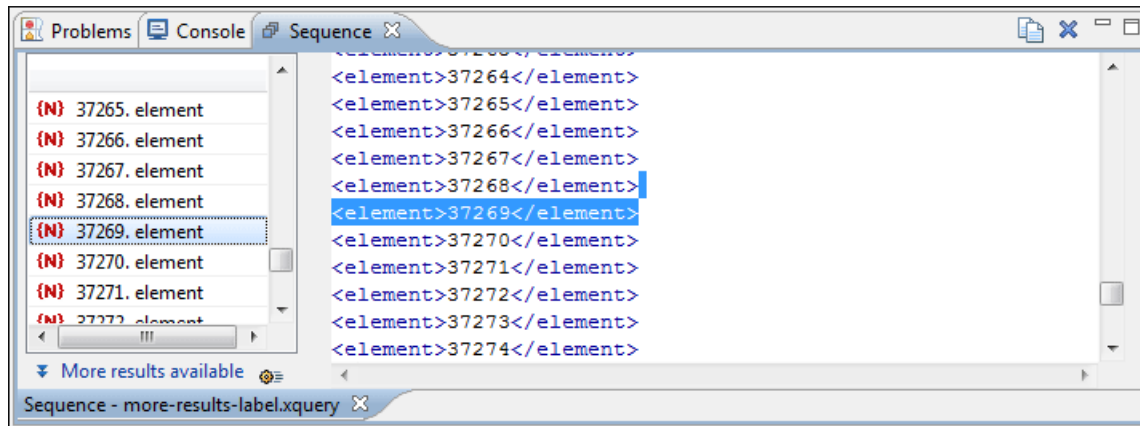


Figure 319: XQuery transformation result displayed in Sequence view

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 [those in the Saxon HE/PE/EE preferences page](#) but they are configured as a specific set of transformation options for each transformation scenario, while the values set in the preferences page apply as global options. The advanced options configured in a transformation scenario override the global options defined in the preferences page.

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")

Allows you to choose how the *strip whitespaces* operation is handled. You can choose 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")

Allows you to set the optimization level. It is the value is an integer in the range of 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 enabled (default value), Saxon runs the XQuery transformation with the XQuery 3.0 support.

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.

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 9.6.0.7 configuration file that is used for XQuery transformation and validation scenarios.

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 `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 result 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 for non-fatal errors are written (wherever 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>.

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 an 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 Developer plugin offers support for XQuery Update 1.0. The XQuery Update Facility provides expressions that can be used to make persistent changes to instances of the XQuery 1.0 and XPath 2.0 Data Model. Thus, besides querying XML documents, you can modify them using the various insert/delete/modify/create methods available in the [XQuery Update 1.0](#) standard.

Choose Saxon 9.6.0.7 EE as a transformer in the scenario associated with the XQuery files containing update statements and Oxygen XML Developer plugin will notify you if the update was successful.

Using XQuery Update to modify a tag name in an XML file

```
rename node doc("books.xml")//publisher[1]//book[1] as "firstBook"
```

Chapter

11

Working with Archives

Topics:

- [Browsing Archives](#)
- [Working with Archive Files](#)

Oxygen XML Developer plugin includes a useful **Archive Browser** view that offers the means to work with files directly from various types of archives (for example, opening and saving files directly in archives, or browsing and modifying archive structures). The archive support is available for all ZIP-type archives, including:

- ZIP archives
- EPUB books
- JAR archives
- Office Open XML (OOXML) files
- Open Document Format (ODF) files
- IDML files

You can transform, validate, and perform many other operations on files directly from an archive. For instance, you can transform, or validate files directly from OOXML or ODF packages, and the structure and content of the ZIP archives can be opened, edited, and saved, similar to any other ZIP archive browsing tool. Also, when browsing for a URL in various dialog boxes, you can use the  **Browse for archived file** action to browse and select files from a particular archive.

Browsing Archives

To view the contents and structure of an archive, use one of the following methods:

- Open an archive from the **Navigator view**. This will open the archive in the main editing pane. The archive will be unmounted when the editor is closed.
- Use the Eclipse File System (EFS) by right-clicking the archive in the **Navigator** view and choosing **Expand Zip Archive**. This will expand the archive in the **Navigator** view. All the standard **Navigator** actions are available on the mounted archive. To close the archive, you can use the **Collapse ZIP Archive** action and any file opened from the expanded archive will be closed when the archive is unmounted.



Tip: If a file is not recognized by Oxygen XML Developer plugin as a supported archive type, you can add it from the [Archive preferences page](#).

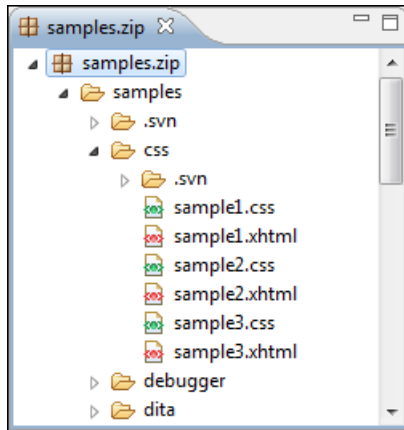


Figure 320: Browsing an Archive

Toolbar Actions

The following actions are available on the **Archive Browser** toolbar:



Validate (available for EPUB archives only)

Checks the EPUB archive to see if its content and structure is valid.



New folder

Creates a folder as child of the selected folder in the browsed archive.



New file

Creates a file as child of the selected folder in the browsed archive.



Add files

Adds existing files as children of the selected folder in the browsed archive.



Note: You can also add files in the archive by dragging them from the file browser or the **Navigator** view and dropping them in the archive editor.



Delete

Deletes the selected resource in the browsed archive.



Open in System Application

Opens the selected resource in the default system application that is associated with that type of file.



Archive Options

Opens the [Archive preferences page](#).

Contextual Menu Actions

The following additional actions are available from the contextual menu for resources in the **Archive Browser** view:

Open

Opens a resource from the archive in the editor.

Extract

Extracts a resource from the archive in a specified folder.

New folder

Creates a folder as child of the selected folder in the browsed archive.

New file

Creates a file as child of the selected folder in the browsed archive.

Add files

Adds existing files as children of the selected folder in the browsed archive.

 **Note:** On OS X, the **Add file** action is also available and it allows you to add one file at a time.

Rename

Renames a resource in the archive.

Find/Replace in Files

Allows you to search for and replace specific pieces of text inside the archive.

Cut

Cuts the selected archive resource.

Copy

Copies the selected archive resource.

Paste

Pastes a file or folder into the archive.

Delete

Removes a file or folder from archive.

Copy location

Copies the URL location of the selected resource.

Refresh

Refreshes the selected resource.

Properties

Shows the properties of the selected resource.

Working with Archive Files

Oxygen XML Developer plugin includes support for working with various types of archives. These include EPUB, OOXML, and ODF files.

When these types of files are opened in the **Navigators** view or the main editing pane, their internal components are expanded:

- Document content (XHTML and image files).
- Packaging files.
- Container files.

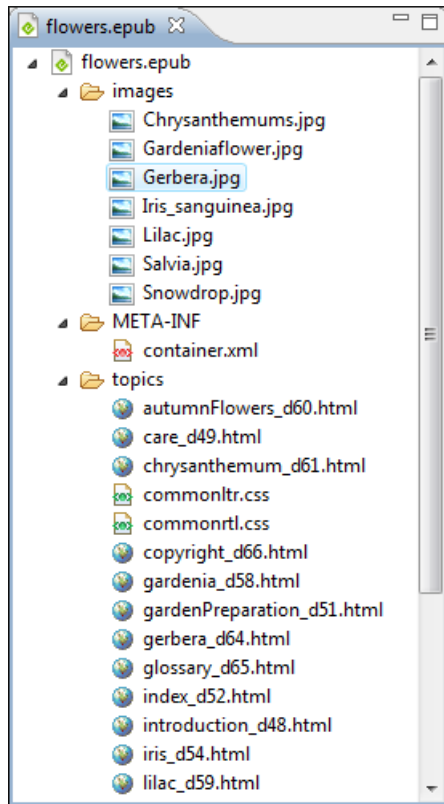


Figure 321: EPUB File Displayed in Eclipse

When an archive is expanded, you can add or delete files that compose the archive structure. All changes made to the structure of an archive are saved immediately. You can open files from within the archive to *edit them in the main editing pane and save changes* back to the archive. You can also use the  **Open in System Application** action to open the archive in the default system application that is associated with that type of file.

EPUB-Specific Validation

When working with EPUB archives, Eclipse includes a  **Validate** action on the toolbar that checks the EPUB archive to make sure the structure and content are valid. Oxygen XML Developer plugin uses the open-source *EpubCheck* validator to perform the validation. This validator detects many types of errors, including OCF container structure, OPF and OPS mark-up, as well as internal reference consistency.

To watch our video demonstration about EPUB support in Oxygen XML Developer plugin, go to <https://www.oxygenxml.com/demo/Epub.html>.

Related information

[The Archive Browser View](#) on page 682

Creating an Archive

To create an archive from scratch, follow these steps:

1. Go to **File > New > New from Templates**.
2. Choose your particular type of archive template. For example, select one of the **ODF**, **OOXML**, or **EPUB** templates.
3. Click **Next** and choose the name and location of the file.
4. Click **Finish**.

A skeleton archive is saved on disk and opened in Eclipse.



Tip: Use toolbar and contextual menu actions to edit, add, and remove resources from the archive. For EPUB archives, you can use the  **Validate** action to verify the integrity of the EPUB archive.

Editing and Saving Files Inside an Archive

You can open files directly from an archive and then edit them in the main editor pane. To open a file, simply double-click it or select  **Open** from the contextual menu.

When saving the file back to the 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](#).

Migrating Archives to DITA or TEI

Certain types of archives can be converted to DITA or TEI. For example, OOXML (Office Open XML) archive files with the DOCX file extension can be migrated to DITA or TEI.

To migrate DOCX files to DITA or TEI, follow these steps:

1. Open and expand the archive in Eclipse.
2. Open the **document.xml** file contained in the archive.
3. Run one of the following predefined transformation scenarios:
 - a) **DOCX DITA** to migrate to DITA.
 - b) **DOCX TEI P5** to migrate to TEI.
4. You may need to do some manual reconfiguring to map DOCX styles to DITA or TEI content.



Tip: Oxygen XML Developer plugin also includes a predefined transformation scenario called **ODT TEI P5** for converting ODF archive files with the ODT file extension to TEI and a similar process can be used to migrate ODT files to TEI.

Chapter 12

Databases and CMS Integration

Topics:

- [Working with Databases](#)
- [Content Management System \(CMS\) Integration](#)

Oxygen XML Developer plugin provides support for connecting and integrating with various databases and content management systems. This section includes information about the database-related features in Oxygen XML Developer plugin. It explains how to connect with the supported databases, presents the actions that are available for each type, and includes information about SharePoint and Documentum (CMS) integration.

Working with Databases

XML is a storage and interchange format for structured data and is supported by all major database systems. Oxygen XML Developer plugin offers the means for managing the interaction with some of the most commonly used databases (both *Relational* and *Native XML* databases). Through this interaction, Oxygen XML Developer plugin helps users with browsing, content editing, importing from databases, using XQuery with databases, SQL execution, and generating XML Schema from a database structure.

Related information

[Integration with Microsoft SharePoint](#) on page 743

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 (click the  connection) . The database structure can be expanded to resource level, or even all the way to column level for tables inside relational databases. Oxygen XML Developer plugin supports multiple simultaneous database connections and the connection tree in the **Data Source Explorer** view provides an easy method for browsing them.

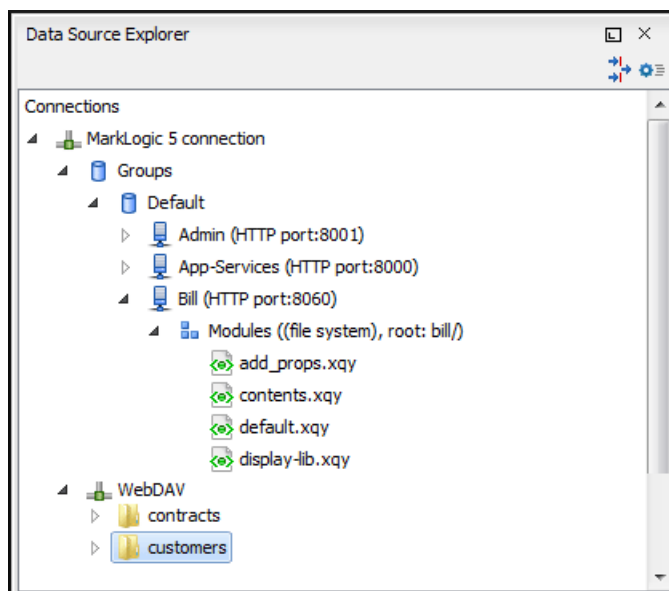


Figure 322: Data Source Explorer View

The objects (nodes) that are displayed in the **Data Source Explorer** view depend on the connection type and structure of the database. Various contextual menu actions are available for each hierarchical level and for some connections you can add or move resources in a container by simply dragging them from the **Navigator** view, a file browsing application, or another database.

Toolbar Actions

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.

Database-Specific Contextual Menu Actions

Each specific type of database will also include its own specific contextual menu actions in the **Data Source Explorer** view. The actions depend on the type of database, the type of node, or the hierarchical level of the node in which the contextual menu is invoked.

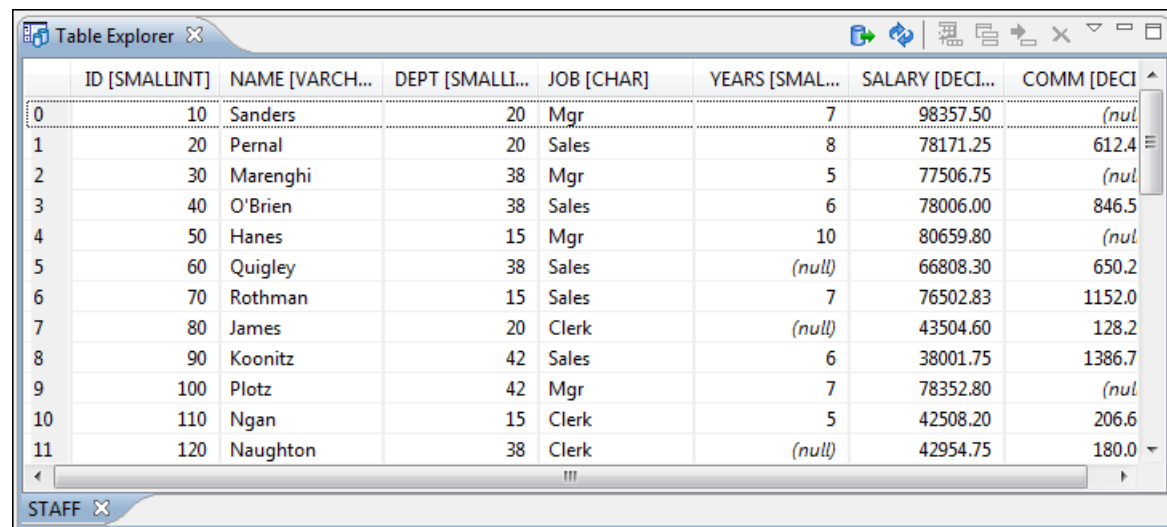
For more information on the specific actions that are available, see the topics in this section for each specific type of database.

Related information

[Data Sources Preferences](#) on page 30

Table Explorer View

Relational databases tables in the **Data Source Explorer** view can be displayed and edited in the **Table Explorer** view by selecting the **Edit** action from the contextual menu of a **Table** node or by double-clicking one of its fields. To modify the content of a cell, double-click it and start typing. When editing is complete, Oxygen XML Developer plugin attempts to update the database with the new cell content.



	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 323: 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 option in 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:

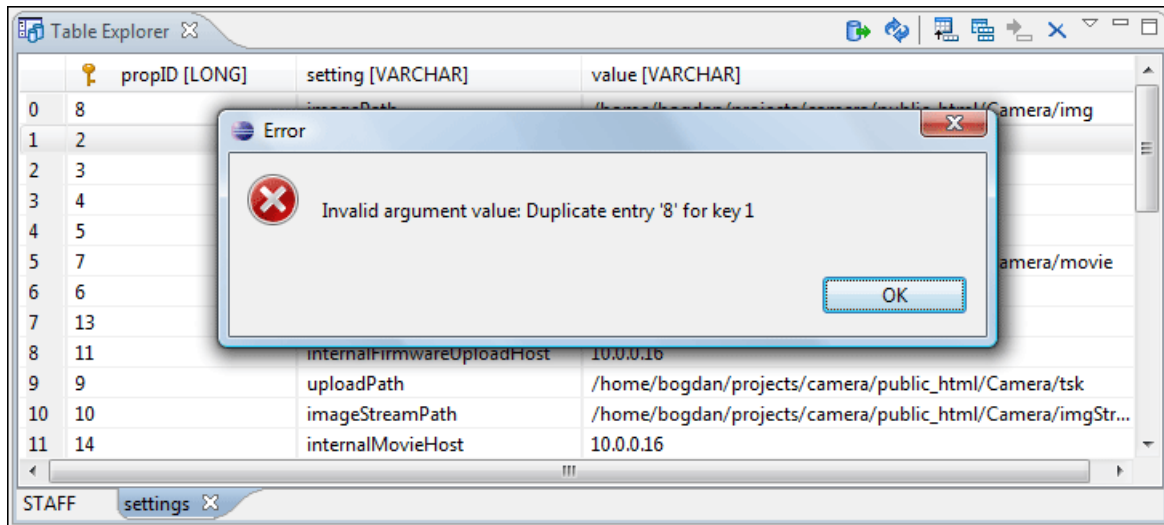


Figure 324: Duplicate Entry for Primary Key

Table Explorer Contextual Menu Actions

Common editing 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 in the **Table Explorer** view, also includes 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.

Table Explorer Toolbar Actions

The toolbar of the **Table Explorer** view 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.

Related information

[Data Source Explorer View](#) on page 687

Database Connection Support

Oxygen XML Developer plugin offers support for a variety of *Relational* and *Native XML* database connections. The database drivers and connections for various types of database are configured in the [Data Sources preferences page](#) and once configured, the database connections can be viewed and managed in the [Data Source Explorer view](#). Oxygen XML Developer plugin also includes a [Database perspective](#) that helps you to manage databases.

The database support in Oxygen XML Developer plugin offers a variety of capabilities, including:

- Browsing the structure of databases in the [Data Source Explorer view](#).
- Viewing relational tables in the [Table Explorer view](#).
- Executing SQL queries against databases.
- Calling stored procedures with input and output parameters.
- XQuery execution with databases.
- Exporting data from databases to XML.

Relational Database Support

Relational databases use a relational model and are based on tables linked by a common key. Oxygen XML Developer plugin offers support for the most commonly used relational databases, including:

- IBM DB2
- Oracle 11g
- Microsoft SQL Server
- PostgreSQL
- MySQL

Oxygen XML Developer plugin also offers generic support (table browsing and execution of SQL queries) for any JDBC-compliant database (for example, *MariaDB*).

To watch our video demonstration about the integration between the relational databases and Oxygen XML Developer plugin, go to https://www.oxygenxml.com/demo/Author_Database_Integration.html.

Native XML Database Support

Native XML databases have an XML-based internal model and their fundamental unit of storage is XML. They use XML as an interface to specify documents as tree structured data that may contain unstructured text, but on disk the data is stored as optimized binary files. This makes query and retrieval processes faster. Oxygen XML Developer plugin offers support for the most commonly used native XML databases, including:

- Berkeley DB XML
- eXist
- MarkLogic
- Documentum xDB (X-Hive/DB) 10
- Oracle XML DB
- Base X

To watch our video demonstration about the integration between the native XML databases and Oxygen XML Developer plugin, go to https://www.oxygenxml.com/demo/Author_Database_XML_Native.html.

Related information

[WebDAV Connections](#) on page 729

[Integration with Microsoft SharePoint](#) on page 743

IBM DB2 Database Connections

Oxygen XML Developer plugin includes support for IBM DB2 database connections. Oxygen XML Developer plugin allows you to browse the structure of an IBM DB2 database in the [Data Source Explorer view](#), open tables in the [Table Explorer view](#), and perform various operations on the resources in the repository.

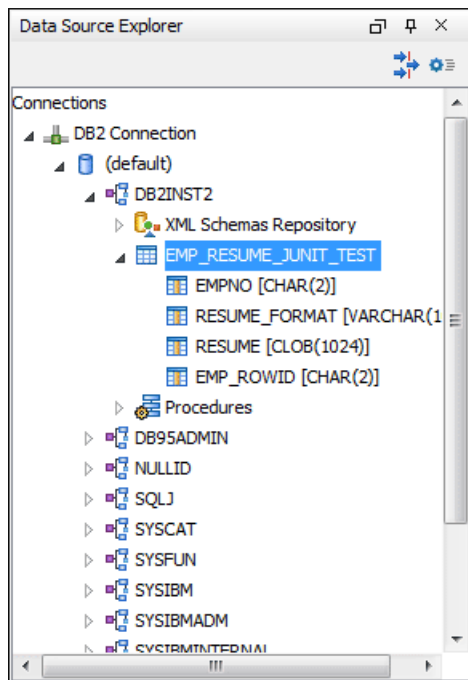


Figure 325: IBM DB2 Database Connection

Configuring an IBM DB2 Database Connection

To configure the support for the IBM DB2 database, follow this procedure:

1. Go to the [IBM website](#) and in the *DB2 Clients and Development Tools* category select the *DB2 Driver for JDBC and SQLJ* download link. Fill out the download form and download the zip file. Unzip the zip file and use the `db2jcc.jar` and `db2jcc_license_cu.jar` files in Oxygen XML Developer plugin for [configuring a DB2 data source](#).
2. [Configure IBM DB2 Data Source drivers](#).
3. [Configure an IBM DB2 Server Connection](#).
4. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure IBM DB2 Data Source Drivers

Available in the Enterprise edition only.

To configure a data source for connecting to an IBM DB2 server, follow these steps:

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.
2. Unzip the downloaded archive.
3. [Open the Preferences dialog box](#) and go to **Data Sources**.
4. Click the **+ New** button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.

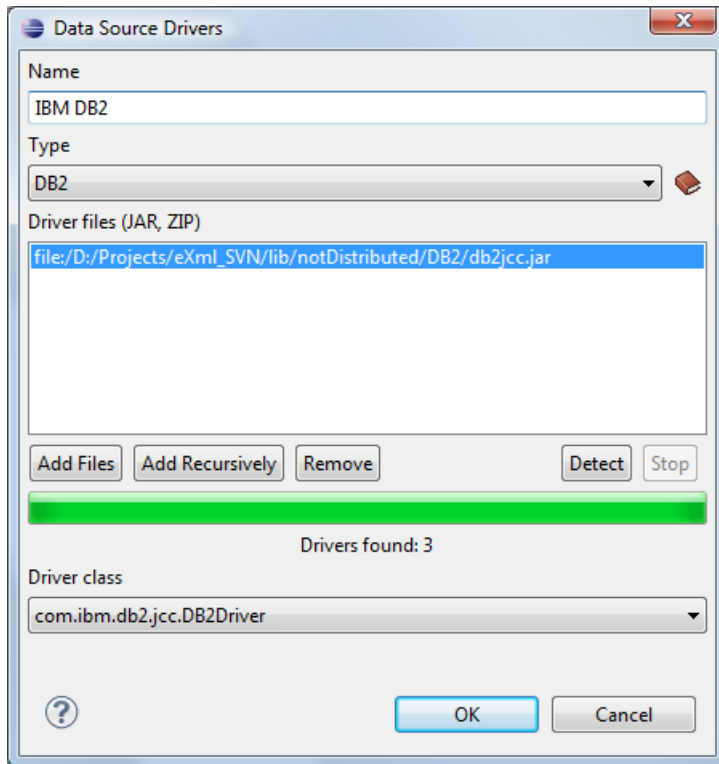


Figure 326: Data Source Drivers Configuration Dialog Box

5. Enter a unique name for the data source.
6. Select **DB2** in the driver **Type** drop-down menu.
7. Click the **Add Files** button and select the IBM DB2 driver files from the archive that you downloaded and unzipped.

The IBM DB2 driver files are:

- db2jcc.jar
- db2jcc_license_cisuz.jar
- db2jcc_license_cu.jar

8. Select the most appropriate **Driver class**.
9. Click the **OK** button to finish the data source configuration.
10. Continue on to [configure your IBM DB2 connection](#).

To watch our video demonstration about running XQuery against an IBM DB2 Pure XML database, go to <https://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.

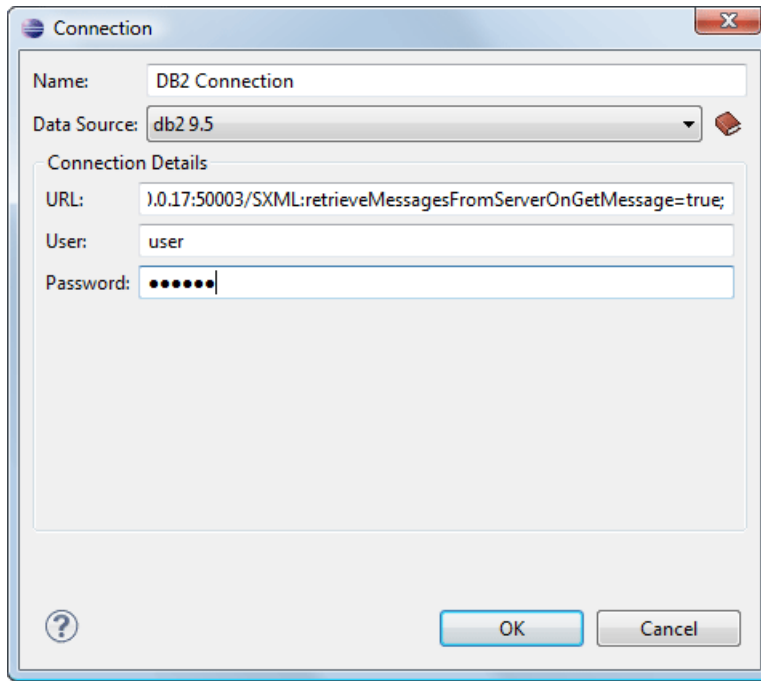


Figure 327: Connection Configuration Dialog Box

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 <https://www.oxygenxml.com/demo/DB2.html>.

IBM DB2 Contextual Menu Actions

General Contextual Menu Actions

For relational databases, the following general actions are available in the contextual menu of the **Data Source Explorer** view, depending on the node in which it is invoked:



Refresh

Performs a refresh on the selected node.

Disconnect (available on Connection nodes)

Closes the current database connection. If a table is already open, you are warned to close it before proceeding.



Configure Database Sources (available on Connection nodes)

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Edit (available on Table nodes)

Opens the selected table in the **Table Explorer** view.



Export to XML (available on Table nodes)

Opens the **Export Criteria** dialog box (a thorough description of this dialog box can be found in the [Import from Database](#) chapter).

Database-Specific Contextual Menu Actions

In addition to the general contextual menu actions in the **Data Source Explorer** view, the various nodes in IBM DB2 connections include the following additional contextual menu actions:

XML Schema Repository Level Nodes

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. Schema dependencies management can be done by using the **Add** and **Remove** buttons.

Schema Level Nodes

Unregister

Removes the selected schema from the XML Schema Repository.

View

Opens the selected schema in Oxygen XML Developer plugin.

Microsoft SQL Server Database Connections

Oxygen XML Developer plugin includes support for Microsoft SQL Server database connections. Oxygen XML Developer plugin allows you to browse the structure of a SQL Server database in the **Data Source Explorer view**, open tables in the **Table Explorer view**, and perform various operations on the resources in the repository.

Configuring a Microsoft SQL Server Connection

To configure the support for a 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 MS SQL Server Data Source drivers](#).
3. [Configure a MS SQL Server Connection](#).
4. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure Microsoft SQL Server Data Source Drivers

Available in the Enterprise edition only.

To configure a data source for connecting to a Microsoft SQL server, follow these steps:

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. [Open the Preferences dialog box](#) and go to **Data Sources**.
3. Click the  **New** button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.

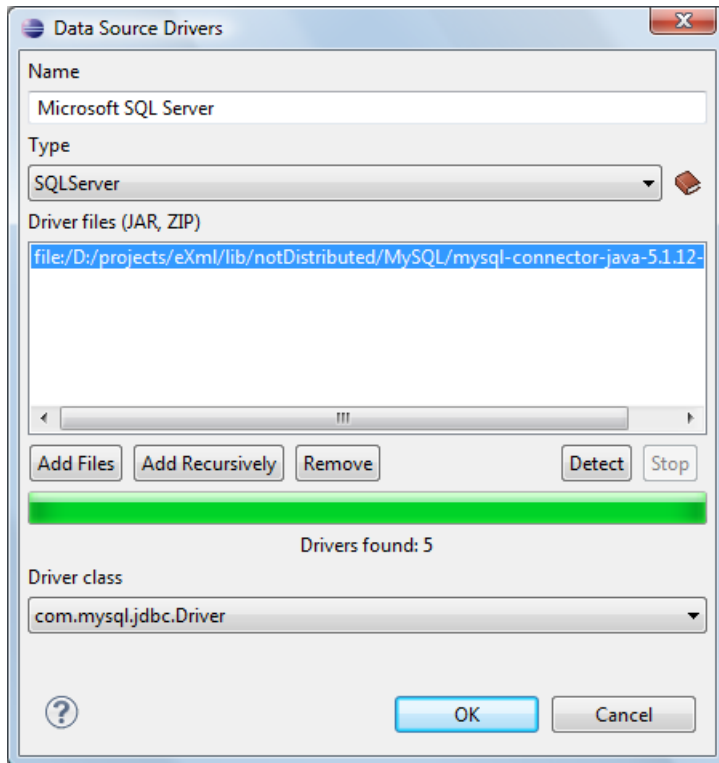


Figure 328: Data Source Drivers Configuration Dialog Box

4. Enter a unique name for the data source.
5. Select **SQLServer** in the driver **Type** drop-down menu.
6. Click the **Add Files** button and select the Microsoft SQL Server driver file that you downloaded. The SQL Server driver file is called `sqljdbc.jar`.
7. Select the most appropriate **Driver class**.
8. Click the **OK** button to finish the data source configuration.
9. Continue on to [configure your Microsoft SQL Server connection](#).

How to Configure a Microsoft SQL Server Connection

The support to configure a Microsoft SQL Server connection is available in the Enterprise edition only.

To configure a connection to a Microsoft SQL Server, follow these steps:

1. [Open the Preferences dialog 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.

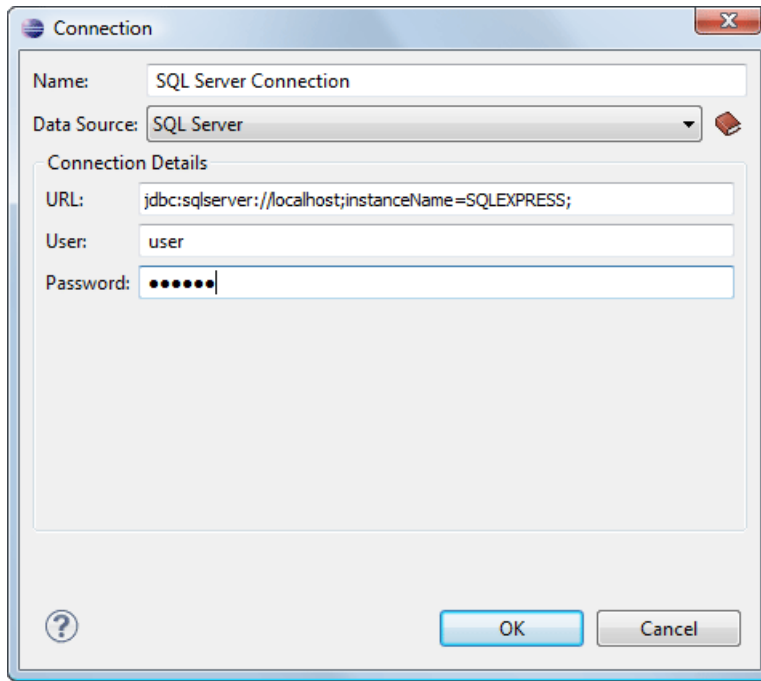


Figure 329: Connection Configuration Dialog Box

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.

Microsoft SQL Server Contextual Menu Actions

General Contextual Menu Actions

For relational databases, the following general actions are available in the contextual menu of the **Data Source Explorer** view, depending on the node in which it is invoked:



Refresh

Performs a refresh on the selected node.

Disconnect (available on Connection nodes)

Closes the current database connection. If a table is already open, you are warned to close it before proceeding.



Configure Database Sources (available on Connection nodes)

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Edit (available on Table nodes)

Opens the selected table in the **Table Explorer** view.

Export to XML (available on **Table nodes**)

Opens the **Export Criteria** dialog box (a thorough description of this dialog box can be found in the [Import from Database](#) chapter).

Database-Specific Contextual Menu Actions

In addition to the general contextual menu actions in the **Data Source Explorer** view, the resource level nodes in Microsoft SQL Server connections include the following additional contextual menu action:

XML Schema Repository Level Nodes

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. Schema dependencies management can be done by using the **Add** and **Remove** buttons.

Schema Level Nodes

Add

Adds a new schema to the XML Schema files.

Unregister

Removes the selected schema from the XML Schema Repository.

View

Opens the selected schema in Oxygen XML Developer plugin.

Oracle Database Connections

The Oracle database is a common relational type of database system. Oxygen XML Developer plugin comes with built-in support for the 11g version of the database system. The Oracle database also includes a Oracle XML DB component that adds native XML support. Oxygen XML Developer plugin allows you to browse Oracle repositories in the [Data Source Explorer view](#), open tables in the [Table Explorer view](#), and perform various operations on the resources in the repository.

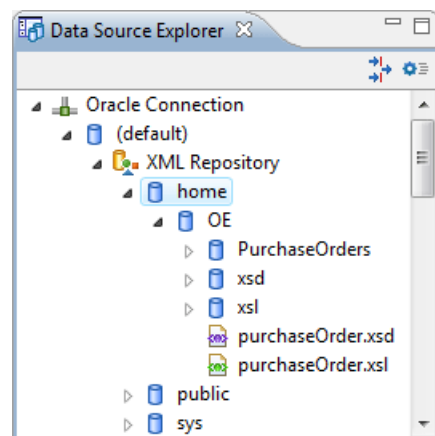


Figure 330: Oracle Database Connection

Related information

[Using XQuery with Oracle XML DB](#)

Configuring an Oracle 11g Database Connection

To configure the support for a Oracle 11g database, follow this procedure:

1. Go to <http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html> and download the Oracle 11g JDBC driver called `ojdbc6.jar`.
2. [Configure Oracle 11g Data Source drivers](#).
3. [Configure an Oracle 11g Connection](#).

4. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure Oracle 11g Data Source Drivers

Available in the Enterprise edition only.

To configure a data source for connecting to an Oracle 11g server, follow these steps:

1. Go to <http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html> and download the Oracle 11g JDBC driver called `ojdbc6.jar`.
2. [Open the Preferences dialog box](#) and go to **Data Sources**.
3. Click the **+ New** button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.

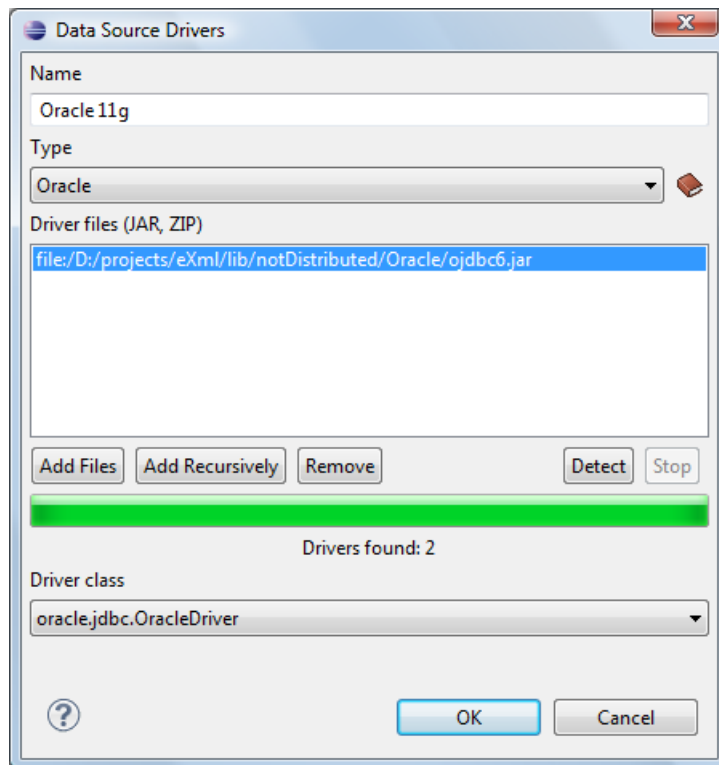


Figure 331: Data Source Drivers Configuration Dialog Box

4. Enter a unique name for the data source.
5. Select *Oracle* in the driver **Type** drop-down menu.
6. Click the **Add Files** button and select the Oracle driver file that you downloaded. The Oracle driver file is called `ojdbc5.jar`.
7. Select the most appropriate **Driver class**.
8. Click the **OK** button to finish the data source configuration.
9. Continue on to [configure your Oracle connection](#).

How to Configure an Oracle 11g Connection

Available in the Enterprise edition only.

To configure a connection to an Oracle 11g 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.

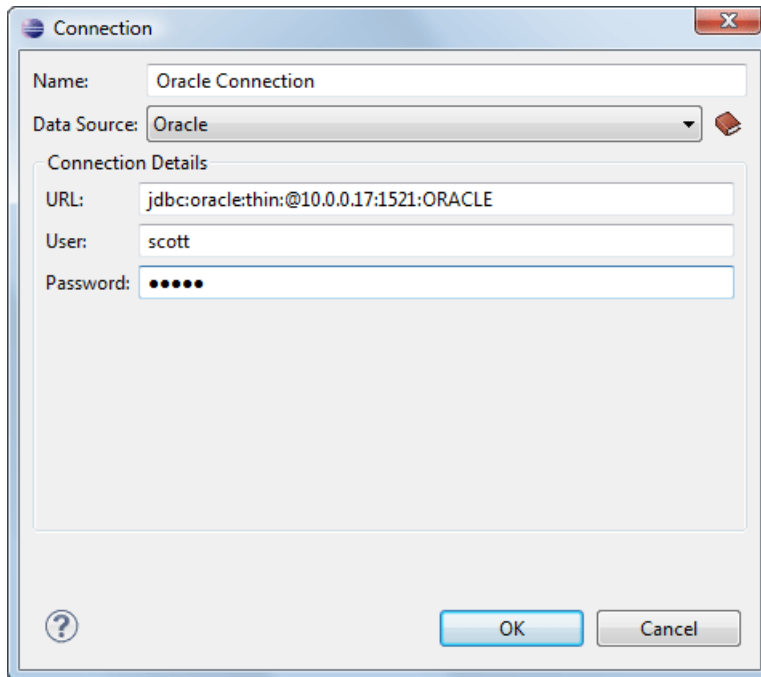


Figure 332: Connection Configuration Dialog Box

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.

Oracle Database Contextual Menu Actions

General Contextual Menu Actions

For relational databases, the following general actions are available in the contextual menu of the **Data Source Explorer** view, depending on the node in which it is invoked:

Refresh

Performs a refresh on the selected node.

Disconnect (available on **Connection nodes**)

Closes the current database connection. If a table is already open, you are warned to close it before proceeding.

Configure Database Sources (available on **Connection nodes**)

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Edit (available on **Table nodes**)

Opens the selected table in the **Table Explorer** view.

Export to XML (available on **Table nodes**)

Opens the **Export Criteria** dialog box (a thorough description of this dialog box can be found in the [Import from Database](#) chapter).

Database-Specific Contextual Menu Actions

In addition to the general contextual menu actions in the **Data Source Explorer** view, the various nodes in Oracle database connections include the following additional contextual menu actions:

XML Schema Repository Level Nodes

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.

XML Repository Level Nodes

Add container

Adds a new child container to the current one.



Add resource

Adds a new resource to the folder.

Container Level Nodes

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.

Resource Level Nodes

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 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.



Properties

Shows various properties of the current container.

Compare

Compares two selected resources.

PostgreSQL Database Connections

Oxygen XML Developer plugin includes support for PostgreSQL database connections. Oxygen XML Developer plugin allows you to browse the structure of a PostgreSQL database in the [Data Source Explorer view](#), open tables in the [Table Explorer view](#), and perform various operations on the resources in the repository.

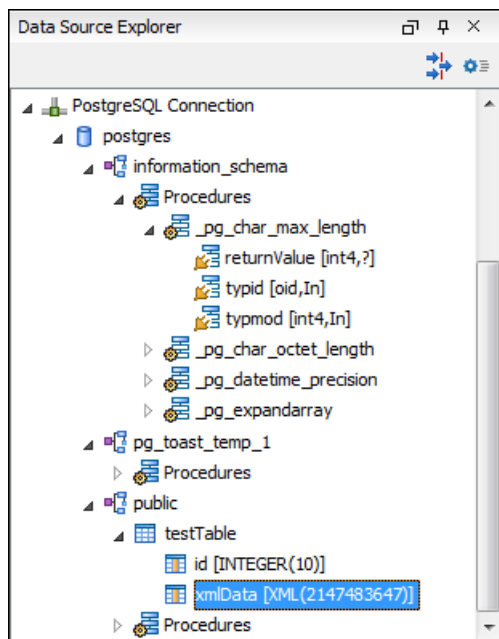


Figure 333: PostgreSQL Database Connection

Configuring a PostgreSQL Database Connection

To configure the support for a PostgreSQL database, follow this procedure:

1. Go to <http://jdbc.postgresql.org/download.html> and download the PostgreSQL 8.3 JDBC3 driver.
2. [Configure PostgreSQL Data Source drivers](#).
3. [Configure a PostgreSQL Connection](#).
4. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure PostgreSQL 8.3 Data Source Drivers

To configure a data source for connecting to a PostgreSQL server, follow these steps:

1. Go to <http://jdbc.postgresql.org/download.html> and download the PostgreSQL 8.3 JDBC3 driver.
2. Open the **Preferences dialog box** and go to **Data Sources**.
3. Click the **+ New** button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.

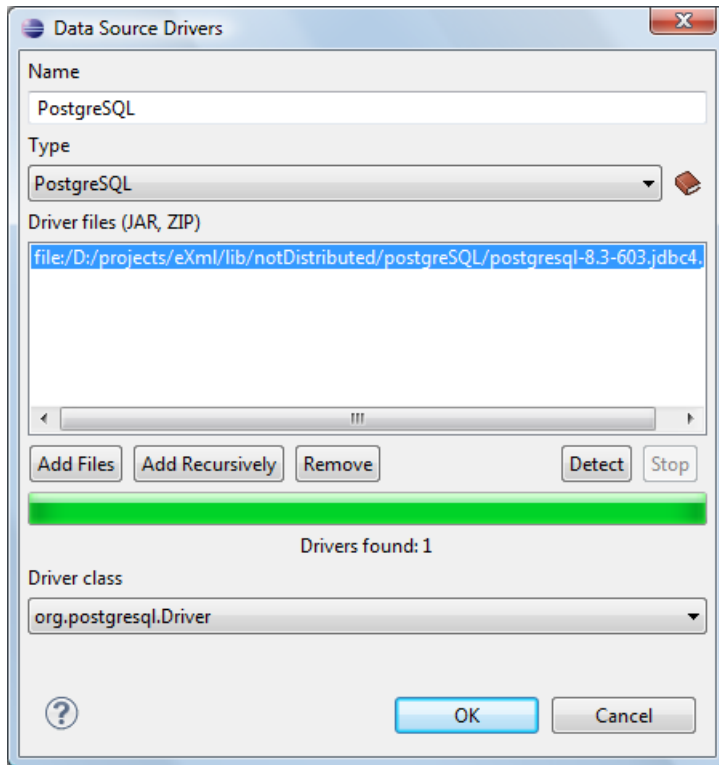


Figure 334: Data Source Drivers Configuration Dialog Box

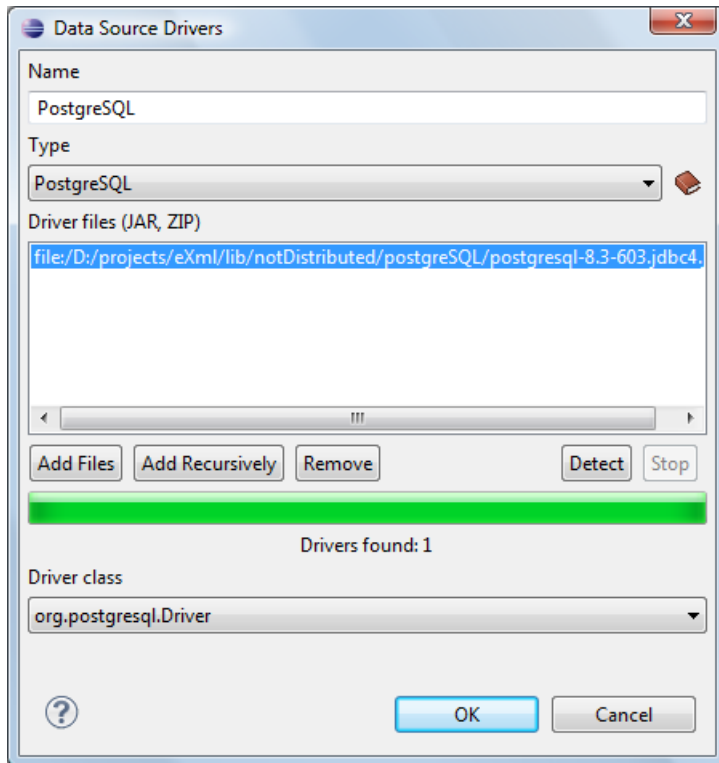


Figure 335: Data Source Drivers Configuration Dialog Box

4. Enter a unique name for the data source.
5. Select *PostgreSQL* in the driver **Type** drop-down list.
6. Click the **Add Files** button and select the PostgreSQL driver file that you downloaded.
The PostgreSQL driver file is called `postgresql-8.3-603.jdbc3.jar`.
7. Select the most appropriate **Driver class**.
8. Click the **OK** button to finish the data source configuration.
9. Continue on to [configure your PostgreSQL connection](#).

How to Configure a PostgreSQL 8.3 Connection

To configure a connection to a PostgreSQL 8.3 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.

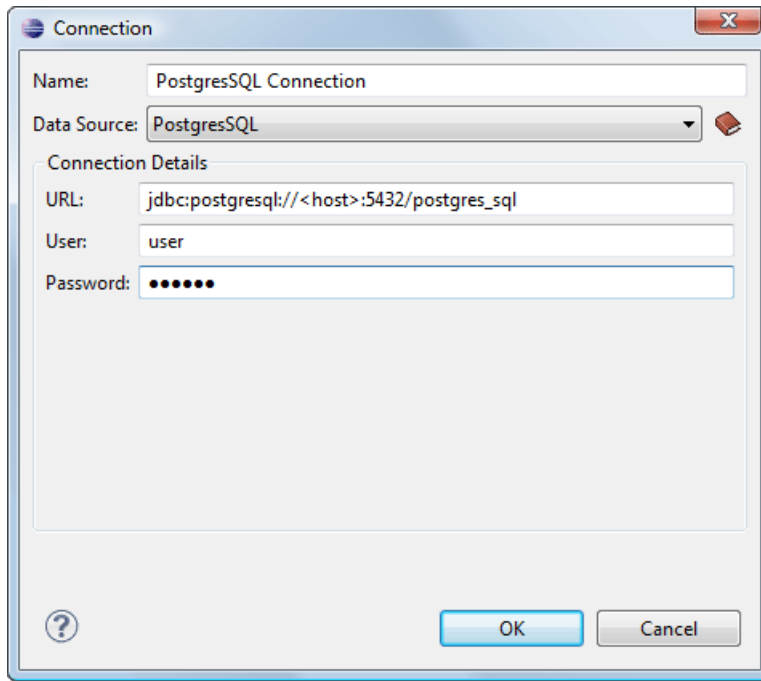


Figure 336: Connection Configuration Dialog Box

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.

PostgreSQL Contextual Menu Actions

General Contextual Menu Actions

For relational databases, the following general actions are available in the contextual menu of the **Data Source Explorer** view, depending on the node in which it is invoked:



Refresh

Performs a refresh on the selected node.

Disconnect (available on Connection nodes)

Closes the current database connection. If a table is already open, you are warned to close it before proceeding.



Configure Database Sources (available on Connection nodes)

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Edit (available on Table nodes)

Opens the selected table in the **Table Explorer** view.



Export to XML (available on Table nodes)

Opens the **Export Criteria** dialog box (a thorough description of this dialog box can be found in the [Import from Database](#) chapter).

Database-Specific Contextual Menu Actions

In addition to the general contextual menu actions in the **Data Source Explorer** view, the resource level nodes in PostgreSQL connections include the following additional contextual menu action:

Resource Level Nodes

Compare

Compares two selected resources.

Berkeley DB XML Database Connections

Oxygen XML Developer plugin includes support for Berkeley DB XML database connections. Oxygen XML Developer plugin allows you to browse the structure of a Berkeley DB XML database in the [Data Source Explorer view](#) and perform various operations on the resources in the repository.

Oracle Berkeley DB XML is an open source, embeddable XML database with XQuery-based access to documents stored in containers and indexed based on their content. It is built on top of the Oracle Berkeley DB and inherits its features and attributes, along with native XML support. A detailed description can be found at: <http://www.oracle.com/us/products/database/berkeley-db/xml/overview/index.html>.

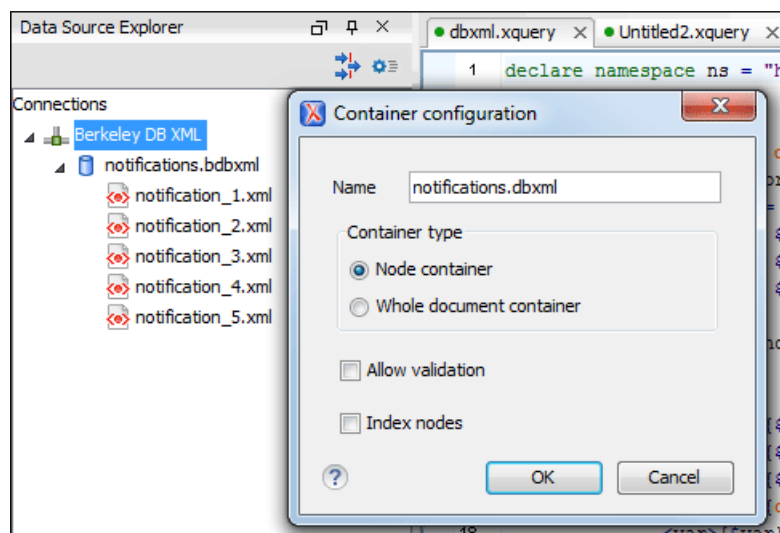


Figure 337: Berkeley DB XML Connection

Configuring a Berkeley DB XML Database Connection

Follow this procedure to configure the support for a Berkeley DB XML database:

1. [Configure Berkeley DB XML Data Source drivers](#).
2. [Configure a Berkeley DB XML Connection](#).
3. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure Berkeley DB XML Data Source Drivers

For this procedure, you need to already have a Berkeley DB XML database installed on your system.

Oxygen XML Developer plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. To configure a data source for a Berkeley DB XML database, follow these steps:

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 Files** button to add the Berkeley DB driver files.

The driver files for the Berkeley DB database (and their locations) are as follows:

- `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.
7. Continue on to [configure your Berkeley DB XML connection](#).

How to Configure a Berkeley DB XML Connection

Oxygen XML Developer plugin supports Berkeley DB XML versions 2.3.10, 2.4.13, 2.4.16 & 2.5.16. To configure a connection to a Berkeley DB XML database, follow these steps:

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 **Join existing environment** checkbox.

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.

Berkeley DB XML Contextual Menu Actions

While browsing Berkeley DB XML connections in the **Data Source Explorer** view, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Disconnect

Stops the connection.

New Collection

Opens a **Container configuration** dialog box that allows you to add a new container in the repository.

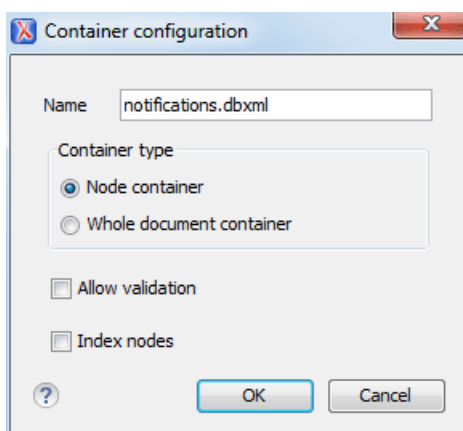


Figure 338: Container Configuration Dialog Box

This dialog box allows you to configure the following:

- **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. You can select one of the following types:
 - **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 selection and is the 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, documents will be validated when they are loaded into the container. The default behavior is to not validate documents.
- **Index nodes** - If checked, indices for the container will 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**.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Container Level Nodes



Import Files

Allows you to add a new file on the connection, in the current folder.

Export

Allows you to export the folder on the remote connection to a local folder.



Cut

Removes the current selection and places it in the clipboard.



Paste

Pastes the copied selection.

Rename

Renames the current resource



Delete

Deletes the current container.

Edit indices

Opens a **Container Indices** dialog box that allows you to configure indices properties for the selected Berkeley container.

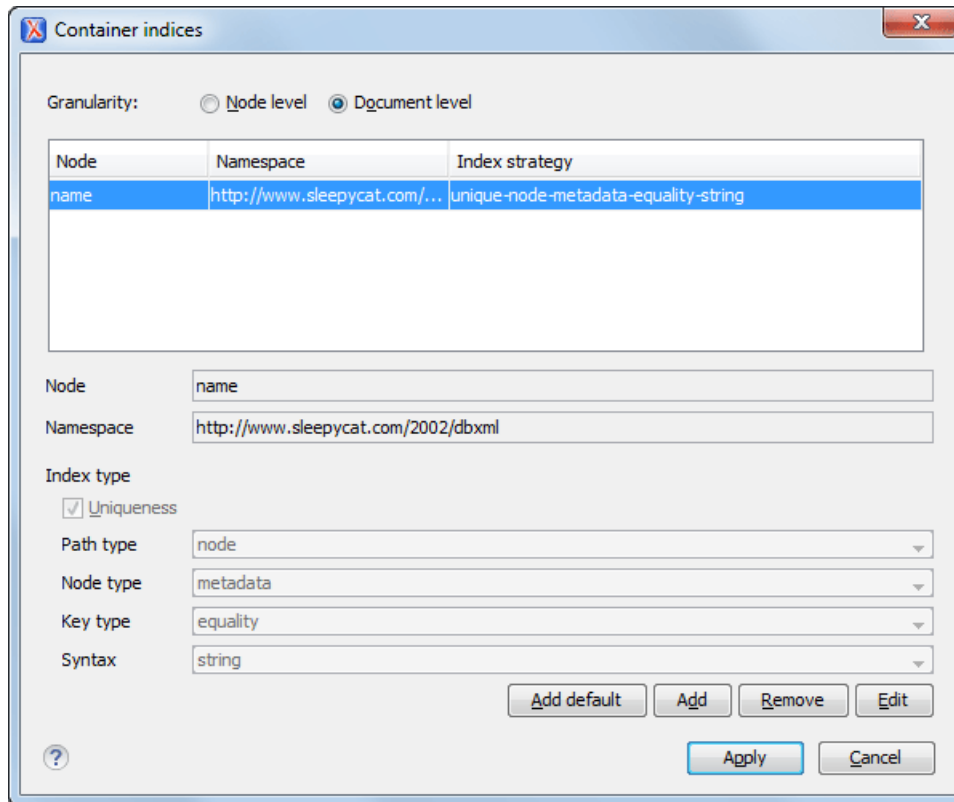


Figure 339: Container Indices Dialog Box

This dialog box allows you to configure the following properties:

- **Granularity** - A measure of the level of details of your data in the database. You can select one of the following:
 - **Document level** - Good option for retrieving large documents.
 - **Node level** - Good option for retrieving nodes from within documents.
- **Node** - The name of the node.
- **Namespace** - The index namespace.
- **Index type:**
 - **Uniqueness** - Indicates whether or not the indexed value must be unique within the container.
 - **Path type** - Drop-down menu that allows you to select from the following:
 - **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** - Drop-down menu that allows you to select from the following:
 - **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** - Drop-down menu that allows you to select from the following:
 - **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** - The syntax describes the type of data the index contains and is mostly used to determine how indexed values are compared. The default value is `string`.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Resource Level Nodes

Open

Opens the selected resource in the editor.



Cut

Removes the current selection and places it in the clipboard.

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.

Rename

Renames the current resource



Delete

Deletes the current container.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.

Compare

Compares two selected resources.

Debugging with Berkeley DB XML

The Berkeley DB XML database added a debugging interface starting with version 2.5. The current version is supported in the Oxygen XML Developer plugin XQuery Debugger. *The same restrictions and peculiarities* apply for the Berkeley debugger as for the MarkLogic debugger.

eXist Database Connections

Oxygen XML Developer plugin includes support for eXist database connections. Oxygen XML Developer plugin allows you to browse the structure of a eXist database in the [Data Source Explorer view](#) and perform various operations on the resources in the repository.

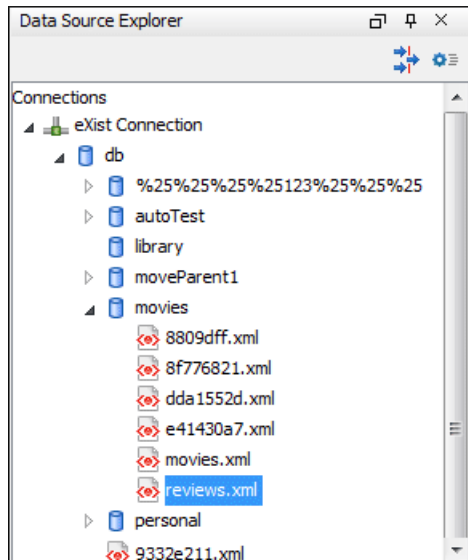


Figure 340: eXist Database Connection

Configuring an eXist Database Connection

There are two ways to configure the support for an eXist database:

1. Use the dedicated **Create eXist-db XML connection** connection wizard.
 - a. [Open the Preferences dialog box](#), go to **Data Sources** and click the **Create eXist-db XML connection** link.
 - b. Enter your connection details in the connection wizard and click **OK**.

 **Important:** To create an eXist connection using this wizard, Oxygen XML Developer plugin expects the `exist/webstart/exist.jnlp` path to be accessible at the provided **Host** and **Port**.
 - c. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).
2. Use the **Data Sources** preferences page to configure your connection.
 - a. [Configure eXist Data Source drivers](#).
 - b. [Configure an eXist Connection](#).
 - c. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure eXist Data Source Drivers

For this procedure, you need to already have an eXist database server installed.

Oxygen XML Developer plugin supports eXist database server versions up to and including version 2.2. To configure a data source for an eXist database, follow these steps:

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 Files** button to add the eXist driver files.

The following driver files should be added and 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 Developer plugin.

- `exist.jar`

- lib/core/xmlldb.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 data source configuration.
7. Continue on to [configure your eXist connection](#).

To watch our video demonstration about running XQuery against an eXist XML database, go to https://www.oxygenxml.com/demo/eXist_Database.html.

How to Configure an eXist Connection

To configure a connection to an eXist database, follow these steps:

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 https://www.oxygenxml.com/demo/eXist_Database.html.

eXist Contextual Menu Actions

While browsing eXist database connections in the **Data Source Explorer** view, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Disconnect

Stops the connection.

Refresh

Performs a refresh on the selected node.

Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.

Container Level Nodes

New File

Creates a new file on the connection, in the current folder.

New Collection

Creates a new collection on the connection.

Import Folders

Imports folders on the server.

Import Files

Allows you to add a new file on the connection, in the current folder.

Export

Allows you to export the folder on the remote connection to a local folder.



Cut

Removes the current selection and places it in the clipboard.



Paste

Pastes the copied selection.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Resource Level Nodes

Open

Opens the selected resource in the editor.

Save As

Allows you to save the selected resource as a file on disk.



Cut

Removes the current selection and places it in the clipboard.



Copy

Copies the current selection into the clipboard.

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.

Rename

Renames the current resource



Delete

Deletes the current container.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.

Compare

Compares two selected resources.

MarkLogic Database Connections

Oxygen XML Developer plugin Enterprise edition includes support for MarkLogic database connections. 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 MarkLogic server. You can expand each application server and view all of its configured modules, and 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 the following query:

```
(xdmp:directory-create('/code/modules/'),  
xdmp:directory-create('/code/modules/imports/'))
```

For more information about directory properties, go to: <http://blakeley.com/blogfile/2012/03/19/directory-assistance/>.

MarkLogic and XQuery

MarkLogic connections can be used in conjunction with XQuery scripts to debug and solve problems with XQuery transformations. XQuery modules can also be validated using a MarkLogic server to allow you to spot possible issues without the need of actually executing the XQuery script.

When *debugging XQuery files with MarkLogic*, you can use the **Data Source Explorer** view to open the files from the application server that is involved in the debugging process. By using the **Data Source Explorer** view, any imported modules are better identified by the MarkLogic server. You can also *use step actions and breakpoints* in the modules to help identify problems.

Modules Container

For each Application server (for example: *Bill (HTTP port:8060)*), you have access to the XQuery modules that are visible to that server. When editing, executing, or debugging XQuery it is recommended to open the XQuery files from this  **Modules** container.

 **Note:** You can also manage resources for a MarkLogic database through a WebDAV connection, although it is not recommended if you work with XQuery files since imported modules may not be resolved correctly.

Requests Container

Each MarkLogic application server includes a  **Requests** container. In this container, Oxygen XML Developer plugin displays both queries that are stopped for debugging purposes and 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 requests action*.

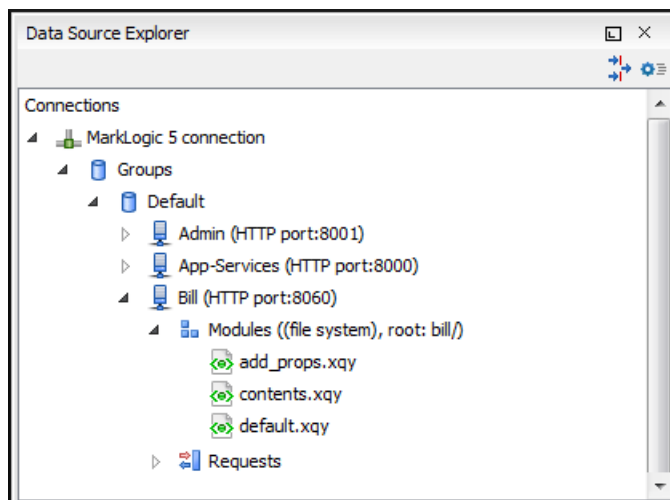


Figure 341: MarkLogic Connection in Data Source Explorer

Configuring a MarkLogic Database Connection

Note that this feature is available in Oxygen XML Developer plugin Enterprise edition only.

Follow this procedure to configure the support for a MarkLogic database connection:

1. Download the MarkLogic driver from [MarkLogic Community site](#).
2. [Configure MarkLogic Data Source drivers](#).
3. [Configure a MarkLogic Connection](#).
4. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

Related information

[MarkLogic Development in Oxygen XML Developer plugin](#) on page 715

How to Configure MarkLogic Data Source Drivers

Available in the Enterprise edition only.



Note: Oxygen XML Developer plugin supports MarkLogic version 4.0 or later.

To configure a data source for MarkLogic, follow this procedure:

1. Download the **XCC Java distribution** zip file from: <http://developer.marklogic.com/products/xcc>.
2. Unzip the downloaded archive.
3. [Open the Preferences dialog box](#) and go to **Data Sources**.
4. Click the **+ New** button in the **Data Sources** panel.
5. Enter a unique name for the data source.
6. Select *MarkLogic* from the driver **Type** drop-down list.
7. Click the **Add Files** button and select the MarkLogic driver file from the `lib` folder of the archive that you downloaded and unzipped. The driver file name is `marklogic-xcc-{server_version}.jar`, where `{server_version}` is the MarkLogic server version.
8. Click the **OK** button to finish the data source configuration.
9. Continue on to [configure your MarkLogic Connection](#).

How to Configure a MarkLogic Connection

Available in the Enterprise edition only.



Note: Oxygen XML Developer plugin supports MarkLogic version 4.0 or later.

To configure a connection to a MarkLogic database, follow these steps:

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 Developer plugin uses XCC connector to interact with MarkLogic XDBC server and requires the basic authentication schema to be set. Starting with version MarkLogic 4.0 the default authentication method when you create an 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 for 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.

MarkLogic Development in Oxygen XML Developer plugin

The Oxygen XML Developer plugin support for MarkLogic includes features designed for developers, such as debugging XQuery transformations, remote and collaborative debugging, XQuery editing and validation, and an XQuery builder that helps to improve productivity.

Working with XQuery Files

MarkLogic supports working with XQuery files to create queries over stored XML content. You can open an XQuery file, configure a transformation scenario to match your MarkLogic connection, write the XQuery, and then execute it.

When editing XQuery modules stored on the MarkLogic server, the [Outline view](#) collects and displays all the functions from all imported modules. The **Content Completion Assistant** also presents all of these functions along with the latest built-in XQuery functions in accordance with the server version.

When developing queries for MarkLogic, it is best to open the resources from the [Data Source Explorer view](#). When you execute or debug XQuery files opened from this view, imported modules can be resolved better by the MarkLogic server. Another advantage is that validation is automatically performed on the MarkLogic server, including any imported modules.

XQuery Debugging

Oxygen XML Developer plugin allows you to use MarkLogic connections to debug real applications that use XQuery (for example, web applications that trigger XQuery executions). By setting the server in debug mode, you can intercept all the XQuery scripts that run on that server. Oxygen XML Developer plugin connects to the MarkLogic server, shows you the running XQuery scripts, and allows you to debug them. The remote debugging support also allows you to debug collaboratively. Multiple users can participate in the same debugging session. You can start a debugging session and another user can continue it, and vice versa.

Working with Modules

MarkLogic has a concept of two types of XQuery modules, *library* and *main* modules. A *library* module is used to define functions. Library modules cannot be evaluated directly. They are imported, either from other library modules or from main modules. A *main* module is used as an entry point that can be executed as an XQuery program. For more information on these types of modules, see [XQuery Library Modules and Main Modules](#).

When working with *library* modules, you need to create a validation scenario and associate it with the module. In the validation scenario you need to specify a main module as the entry point for validation. The modules need to be deployed on a MarkLogic server because Oxygen XML Developer plugin will request the server to validate the modules.

To validate *library* modules stored on a MarkLogic server, follow these steps:

1. [Configure a MarkLogic database connection](#).
2. Expand the MarkLogic connection in the [Data Source Explorer view](#) and open the *library* modules. The *main* module must also be opened from the **Data Source Explorer** view.
3. [Configure a validation scenario](#) for each *library* module. Specify the *main* module in the **URL of the file to validate** field.

Result: Validation is done on the server that contains the *main* module. The *main* module and all other *library* modules involved in the validation must be saved. Otherwise, the server will validate what was saved on the server, without the uncommitted changes. Also, the **Content Completion Assistant** and the **Outline** view should now present the functions from all the modules.

Related information

[Debugging with MarkLogic](#) on page 715

[Configuring a MarkLogic Database Connection](#) on page 713

Debugging with MarkLogic

Oxygen XML Developer plugin includes support for debugging XQuery transformations that are executed against a MarkLogic database.

To use a debugging session against the MarkLogic engine, follow these steps:

1. Configure a [MarkLogic data source](#) and a [MarkLogic connection](#).
2. Make sure that the debugging support is enabled in the MarkLogic server that Oxygen XML Developer plugin accesses. On the server side, debugging must be activated in the XDBC server and in the *Task Server* section of the server control console (the switch *debug allow*). If the debugging is not activated, the MarkLogic server reports a DBG - TASKDEBUGALLOW error.



Note: An XDBC application server must be running to connect to the MarkLogic server and this XDBC server will be used to process XQuery expressions against the server. You can change the XDBC application server that Oxygen XML Developer plugin uses to process XQuery expressions by selecting the [Use it to execute queries action](#) from the contextual menu in the **Data Source Explorer** view.

3. Open the XQuery file and start the debugging process.
 - If you want to debug an XQuery file stored on the MarkLogic server, we recommend you to use the [Data Source Explorer view](#) and open the file from the application server that is involved in the debugging process. This improves the resolving of any imported modules.
 - The MarkLogic XQuery debugger integrates seamlessly into the [XQuery Debugger perspective](#). If you have a MarkLogic validation scenario configured for the XQuery file, you can choose to [debug the scenario](#) directly.
 - Otherwise, 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.



Note: Before starting a debugging session, it is 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** view and run debugging sessions from them.

In a MarkLogic debugging session, you can use step actions and breakpoints to help identify problems. When you add a breakpoint on a line where the debugger never stops, Oxygen XML Developer plugin displays a warning message. These warnings are displayed for breakpoints you add either in the main XQuery (which you can open locally or from the server) or for breakpoints you add in any XQuery that is opened from the connection that participates in the debugging session. For more information, see [Using Breakpoints for Debugging Queries that Import Modules with MarkLogic](#) on page 717.

Remote Debugging with MarkLogic

Oxygen XML Developer plugin allows you to debug remote applications that use XQuery (for example, web applications that trigger XQuery executions). Oxygen XML Developer plugin connects to a MarkLogic server, shows you the running XQuery scripts and allows you to debug them. You can even pause the scripts so that you can start the debugging queries in the exact context of the application. You can also switch a server to debug mode to intercept all XQuery scripts.

Oxygen XML Developer plugin also 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.



Important: When using the remote debugging feature, the HTTP and the XDBC servers involved in the debugging session must have the same module configuration.

To watch our video demonstration about the XQuery debugger for MarkLogic, go to <https://www.oxygenxml.com/demo/XQueryDebuggerforMarkLogic.html>.

Related information

[MarkLogic Development in Oxygen XML Developer plugin](#) on page 715

[Configuring a MarkLogic Database Connection](#) on page 713

Using Breakpoints for Debugging Queries that Import Modules with MarkLogic

When debugging queries that imports modules stored in the database, it is recommended to place *breakpoints* in the modules. 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 all the modules will be considered. Also, make sure that there are no other opened modules that are not involved in the current debugging session.

To place *breakpoints* in the modules, use the following procedure:

1. In the **Data Source Explorer** view, open all the modules from the  **Modules** container of the *XDBC application server* that performs the debugging.
2. *Set breakpoints* in the module as needed.
3. *Continue debugging* the query.

If you get a warning that the breakpoints failed to initialize, try the following solutions:

- Check the *Breakpoints view* and make sure there are no older breakpoints (set on resources that are not part of the current debugging context).
- Make sure you open the modules from the context of the application server that does the debugging and place breakpoints there.

Related information

MarkLogic Database Connections on page 712

MarkLogic Development in Oxygen XML Developer plugin on page 715

Peculiarities and Limitations of the MarkLogic Debugger

MarkLogic debugger has the following peculiarities and limitations:

- Debugging support is only available 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'.
- 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 only 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** view.
 - 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.

MarkLogic Contextual Menu Actions

While browsing MarkLogic connections in the **Data Source Explorer** view, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the *Data Sources preferences page* where you can configure both data sources and connections.

Disconnect

Stops the connection.

Link to Project

Links the connection to a project. This is helpful for *MarkLogic debugging sessions*.

Refresh

Performs a refresh on the selected node.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Container Level Nodes

Enable Debug Mode

Switches the server to a debugging mode. For more information, see [MarkLogic debugging sessions](#).

Use it to Execute Queries

The server will be used to process XQuery expressions against it.



Refresh

Performs a refresh on the selected node.



Module or Folder Level Nodes

Export

Allows you to export the folder on the remote connection to a local folder.



Refresh

Performs a refresh on the selected node.



Requests Level Nodes



Refresh

Performs a refresh on the selected node.

Cancel all requests

Cancels all queries that are either running or stopped on the application server. You can use this action to clean up the entire **Requests** container at the end of your sessions.



Resource Level Nodes

Open

Opens the selected resource in the editor.

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.



Refresh

Performs a refresh on the selected node.

Compare

Compares two selected resources.

Related information

[Configuring a MarkLogic Database Connection](#) on page 713

[MarkLogic Development in Oxygen XML Developer plugin](#) on page 715

[Debugging with MarkLogic](#) on page 715

Documentum xDB (X-Hive/DB) 10 Database Connections

Oxygen XML Developer plugin includes support for Documentum xDB (X-Hive/DB) 10 database connections. Oxygen XML Developer plugin allows you to browse the structure of a Documentum xDB (X-Hive/DB) 10 database in the [Data Source Explorer view](#) and perform various operations on the resources in the repository.

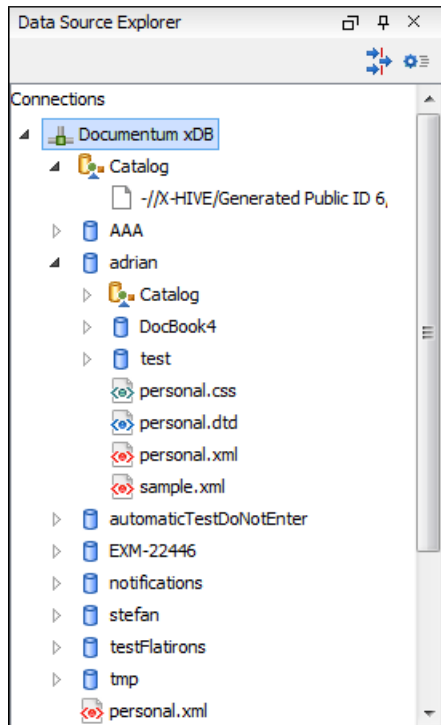


Figure 342: Documentum xDB (X-Hive/DB) 10 Connection

Configuring a Documentum xDB (X-Hive/DB) 10 Database Connection

Follow this procedure to configure the support for a Documentum xDB (X-Hive/DB) 10 database:

1. [Configure Documentum xDB Data Source drivers.](#)
2. [Configure a Documentum xDB Connection.](#)
3. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure Documentum xDB (X-Hive/DB) 10 Data Source Drivers

Available in the Enterprise edition only. For this procedure, you need to already have a Documentum xDB server installed.

To configure a data source for Documentum xDB (X-Hive/DB) 10, follow these steps:

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 *Documentum xDB* from the driver **Type** drop-down menu.
5. Click the **Add** button to add the 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.
7. Continue on to [configure your Documentum xDB \(X-Hive/DB\) 10 connection.](#)

How to Configure an Documentum xDB (X-Hive/DB) 10 Connection

To configure a connection to a Documentum xDB (X-Hive/DB) 10 database, follow these steps:



Note: The bootstrap type of X-Hive/DB connections is not supported in Oxygen XML Developer plugin. The following procedure explains the `xhive://` protocol connection type.

1. Open the [Preferences dialog box](#) and go to **Data Sources**.
2. Click the **+ New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select one of the previously configured data sources from the **Data Source** drop-down 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.

Documentum xDB (X-Hive/DB) 10 Contextual Menu Actions

While browsing Documentum xDB (X-Hive/DB) 10 connections in the **Data Source Explorer** view, the various nodes include the following contextual menu actions:



Connection Level Nodes



Configure Database Sources

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Disconnect

Stops the connection.

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.



Refresh

Performs a refresh on the selected node.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Catalog Level Nodes

Add AS Models

Allows you to add a new abstract schema model to the selected catalog.

Export

Allows you to export the folder on the remote connection to a local folder.

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.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Container (Library) Level Nodes

New File

Creates a new file on the connection, in the current folder.

Add Library

Allows you to add a new library.

Import Folders

Imports folders on the server.

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 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.

Export

Allows you to export the folder on the remote connection to a local 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

Renames the current resource



Delete

Deletes the current container.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Resource Level Nodes

Open

Opens the selected resource in the editor.

Save As

Allows you to save the selected resource as a file on disk.

**Cut**

Removes the current selection and places it in the clipboard.

**Copy**

Copies the current selection into the clipboard.

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.

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.

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.

Rename

Renames the current resource

**Delete**

Deletes the current container.

**Refresh**

Performs a refresh on the selected node.

**Find/Replace in Files**

Allows you to find and replace text in multiple files from the connection.

Compare

Compares two selected resources.

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`

Question:

I am able to access my XML Database in the **Data Source Explorer** and open files for reading but when I try to save changes to a file back into the database, I receive the following error: "Cannot save the file. DTD factory class `org.apache.xerces.impl.dv.dtd.DTDDVFactoryImpl` does not extend from `DTDDVFactory`." How can I fix this?

Answer:

`xhive.jar` contains a `MANIFEST.MF` with a classpath:

```
Class-Path: core/antlr-runtime.jar core/aspectjrt.jar core/fastutil-shrunked.jar
core/google-collect.jar core/icu4j.jar core/lucene-regex.jar core/lucene.jar
core/serializer.jar core/xalan.jar core/xercesImpl.jar
```

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` - `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.

MySQL Database Connections

Oxygen XML Developer plugin includes support for MySQL database connections. Oxygen XML Developer plugin allows you to browse the structure of a SQL Server database in the [Data Source Explorer view](#), open tables in the [Table Explorer view](#), and perform various operations on the resources in the repository.

Configuring a MySQL Database Connection

To configure the support for a MySQL database, follow this procedure:

1. [Configure MySQL Data Source drivers.](#)
2. [Configure a MySQL Connection.](#)
3. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure MySQL Data Source Drivers

To connect to a MySQL server, you need to create a generic JDBC type data source based on [the MySQL JDBC driver available on the MySQL website](#).

To configure this data source, follow these steps:

1. Go to https://www.oxygenxml.com/database_drivers.html and download the appropriate MySQL driver.
2. [Open the Preferences dialog box](#) and go to **Data Sources**.
3. Click the  **New** button in the **Data Sources** panel.

The dialog box for configuring a data source is opened.

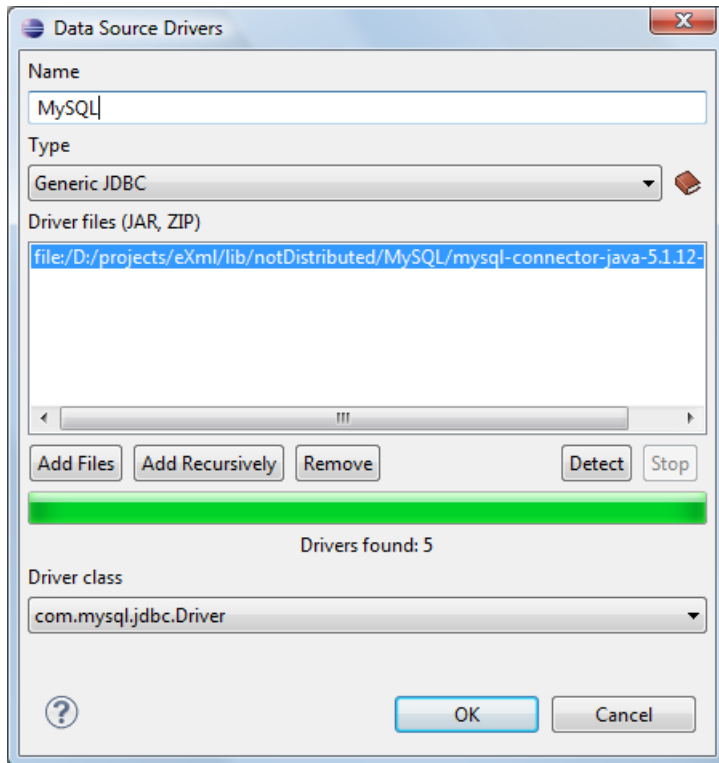


Figure 343: Data Source Drivers Configuration Dialog Box

4. Enter a unique name for the data source.
5. Select *Generic JDBC* in the driver **Type** drop-down list.
6. Click the **Add Files** button and select the MySQL driver file that you downloaded.
The driver file for the MySQL server is called `mysql-com.jar`.
7. Select the most appropriate **Driver class**.
8. Click the **OK** button to finish the data source configuration.
9. Continue on to [configure your MySQL connection](#).

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.

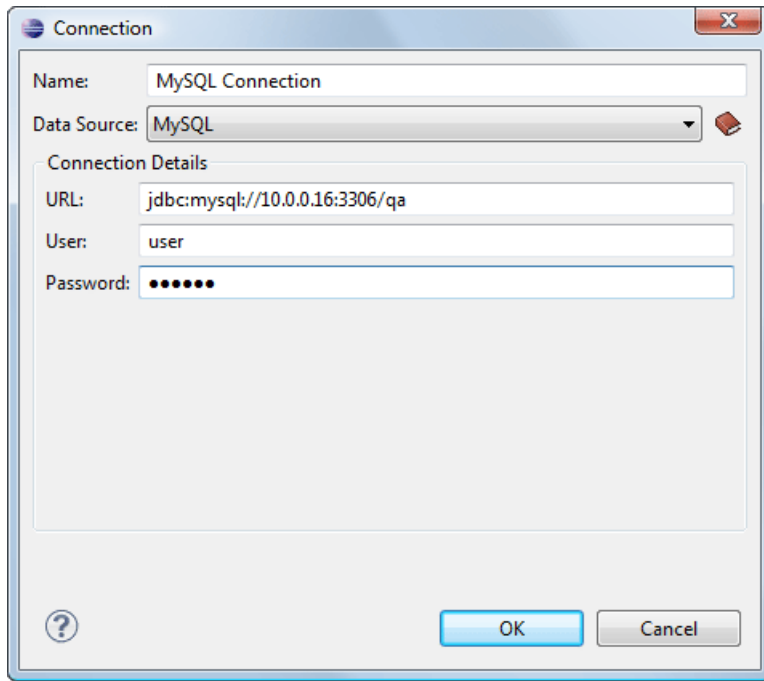


Figure 344: Connection Configuration Dialog Box

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.

Generic JDBC Database Connections

Oxygen XML Developer plugin includes support for Generic JDBC database connections.

Configuring a Generic JDBC Database Connection

To configure the support for a generic JDBC database, follow this procedure:

1. [Configure Generic JDBC Data Source drivers.](#)
2. [Configure a Generic JDBC Connection.](#)
3. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure Generic JDBC Data Source Drivers

Starting with version 17, Oxygen XML Developer plugin comes bundled with Java 8, which does not provide built-in access to JDBC-ODBC data sources. To access such sources, you need to find an alternative JDBC-ODBC bridge or use a platform-independent distribution of Oxygen XML Developer plugin along with a Java VM version 7 or 6.

To configure a generic JDBC data source, follow these steps:

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.
8. Continue on to [configure a generic JDBC connection](#).

How to Configure a Generic JDBC Connection

To configure a connection to a generic JDBC database, follow these steps:

1. [Open the Preferences dialog 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.

JDBC-ODBC Database Connections

Oxygen XML Developer plugin includes support for JDBC-ODBC database connections.

How to Configure a JDBC-ODBC Connection

Starting with version 17, Oxygen XML Developer plugin comes bundled with Java 8, which does not provide built-in access to JDBC-ODBC data sources. To access such sources, you need to find an alternative JDBC-ODBC bridge or use a platform-independent distribution of Oxygen XML Developer plugin along with a Java VM version 7 or 6.

To configure a connection to an ODBC data source, follow these steps:

1. [Open the Preferences dialog box](#) and go to **Data Sources**.
2. In the **Connections** panel, click the **+ New** button.

The dialog box for configuring a database connection is displayed.

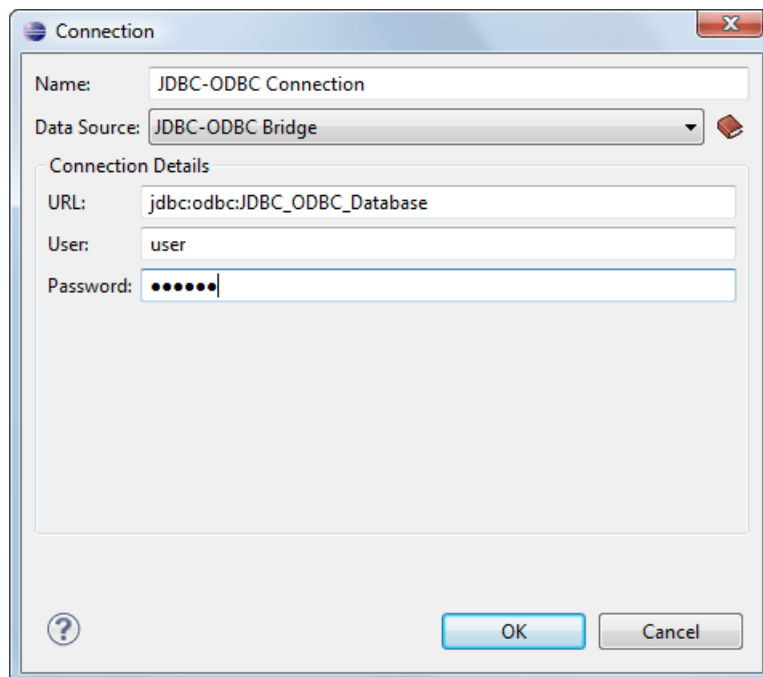


Figure 345: Connection Configuration Dialog Box

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.
7. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

BaseX Database Connections

Oxygen XML Developer plugin includes support for BaseX database connections using a WebDAV connection. BaseX is a light-weight XML database engine and XQuery processor. Oxygen XML Developer plugin allows you to browse the structure of a BaseX database in the [Data Source Explorer view](#) and perform XQuery executions.

How to Configure a BaseX Connection

To configure a BaseX connection, follow these steps:

1. 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 which port the HTTP server using. The default port for BaseX WebDAV is 8984.
2. 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`.
3. Once you are sure that the BaseX WebDAV service is working, you can configure the WebDAV connection in Oxygen XML Developer plugin as described in [How to Configure a WebDAV Connection](#) on page 729. 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`
4. Once the WebDAV connection is created, to view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

BaseX Contextual Menu Actions

While browsing BaseX connections in the **Data Source Explorer** view, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Disconnect

Stops the connection.

New Folder

Creates a new folder on the connection.

Import Files

Allows you to add a new file on the connection, in the current folder.

Refresh

Performs a refresh on the selected node.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Folder Level Nodes

New File

Creates a new file on the connection, in the current folder.

New Folder

Creates a new folder on the connection.

Import Folders

Imports folders on the server.



Import Files

Allows you to add a new file on the connection, in the current folder.

Export

Allows you to export the folder on the remote connection to a local 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

Renames the current resource



Delete

Deletes the current container.



Refresh

Performs a refresh on the selected node.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Resource Level Nodes

Open

Opens the selected resource in the editor.



Cut

Removes the current selection and places it in the clipboard.



Copy

Copies the current selection into the clipboard.

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.

Rename

Renames the current resource



Delete

Deletes the current container.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.

Compare

Compares two selected resources.

Base X XQJ Connection

XQuery execution is possible in a BaseX connection 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 677. 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 678.

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 Developer plugin and create a *Transformation Scenario* as described in [XQuery Transformation](#) on page 538. 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.

WebDAV Connections

Oxygen XML Developer plugin includes support for WebDAV server connections. Oxygen XML Developer plugin allows you to browse the structure of a WebDAV connection in the [Data Source Explorer view](#) and perform various operations on the resources in the repository.

How to Configure a WebDAV Connection

By default, Oxygen XML Developer plugin contains built-in data source drivers for **WebDAV** connections. 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](#).

To configure a WebDAV connection, 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 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.
 7. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).
- To watch our video demonstration about the WebDAV support in Oxygen XML Developer plugin, go to https://www.oxygenxml.com/demo/WebDAV_Support.html.

WebDAV Contextual Menu Actions

While browsing WebDAV connections in the **Data Source Explorer** view, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Disconnect

Stops the connection.

New Folder

Creates a new folder on the connection.

Import Files

Allows you to add a new file on the connection, in the current folder.

Refresh

Performs a refresh on the selected node.

Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.

Folder Level Nodes

New File

Creates a new file on the connection, in the current folder.

New Folder

Creates a new folder on the connection.

Import Folders

Imports folders on the server.

Import Files

Allows you to add a new file on the connection, in the current folder.

Export

Allows you to export the folder on the remote connection to a local 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

Renames the current resource

**Delete**

Deletes the current container.

**Refresh**

Performs a refresh on the selected node.

**Find/Replace in Files**

Allows you to find and replace text in multiple files from the connection.

**Resource Level Nodes****Open**

Opens the selected resource in the editor.

**Cut**

Removes the current selection and places it in the clipboard.

**Copy**

Copies the current selection into the clipboard.

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.

Rename

Renames the current resource

**Delete**

Deletes the current container.

**Refresh**

Performs a refresh on the selected node.

**Properties**

Shows various properties of the current container.

**Find/Replace in Files**

Allows you to find and replace text in multiple files from the connection.

Compare

Compares two selected resources.

SQL Execution Support

The database support in Oxygen XML Developer plugin includes support for writing SQL statements, 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 in the [Database Connection Support](#) section).
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.

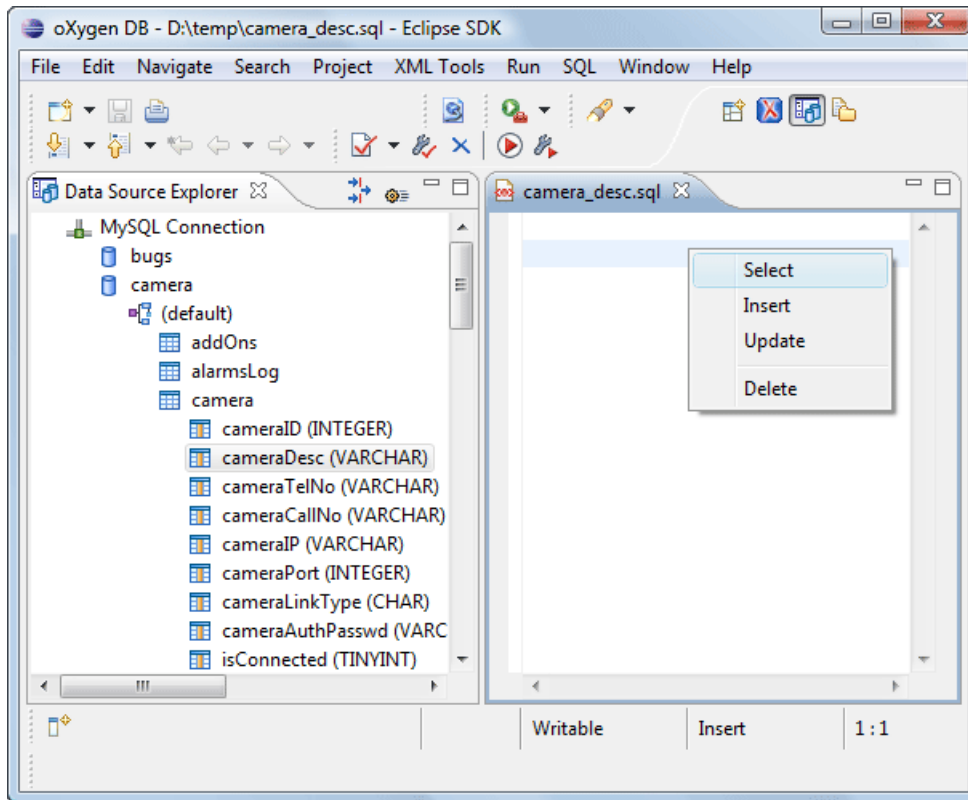


Figure 346: SQL Statement Editing with Drag and Drop

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 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 Developer plugin window.

4. View more complex return values of the SQL transformation in a separate editor panel.

A more complex value returned by the SQL query (for example, an *XMLTYPE* or *CLOB* value) cannot be displayed entirely in the result table.

- a) Right-click the cell containing the complex value.
- b) Select the action **Copy cell** from the contextual menu.
The action copies the value in the clipboard.
- c) Paste the value into an appropriate editor.

For example, you can paste the value in an opened XQuery editor panel of Oxygen XML Developer plugin.

XQuery and Databases

XQuery is a native XML query language that is useful for querying XML views of relational data to create XML results. It also provides the mechanism to efficiently and easily extract information from Native XML Databases (NXD) and relational data. The following database systems supported in Oxygen XML Developer plugin offer XQuery support:

- *Native XML Databases:*
 - Berkeley DB XML
 - eXist
 - MarkLogic (validation support available starting with version 5)
 - Documentum xDB (X-Hive/DB) 10
- *Relational Databases:*
 - IBM DB2
 - Microsoft SQL Server (validation support not available)
 - Oracle (validation support not available)

Build Queries with Drag and Drop from the Data Source Explorer View

When a query is edited in the XQuery editor, the XPath expressions can be composed quickly by dragging them from the **Data Source Explorer** view and dropping them into the editor panel.

1. *Configure the data source drivers* for the particular relational database in the *Data Sources preferences page*.
2. *Configure the connection* for the particular relational database in the *Data Sources preferences page*.
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 for Databases

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 the data source drivers and the connection](#) for the particular database.
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 toward the bottom of the dialog box.
- c) Select [XML Transformation with XQUERY](#).

The **New Scenario** dialog box for configuring an XQuery scenario is opened.

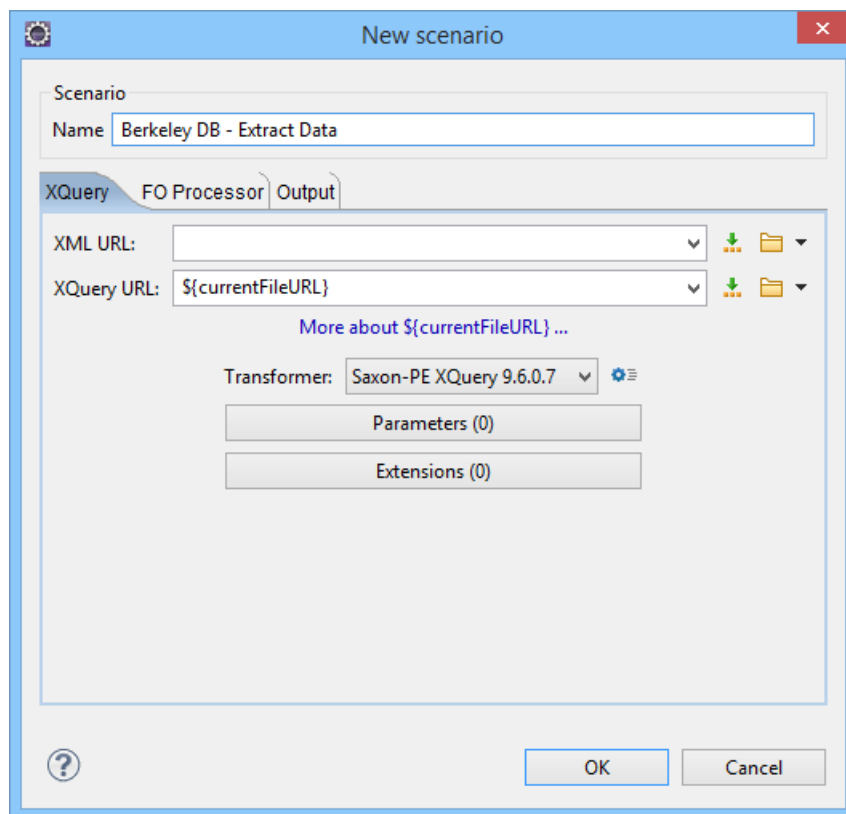


Figure 347: New Scenario Dialog Box

- d) Insert the scenario name in the dialog box for editing the scenario.
- e) Choose the database connection in the **Transformer** drop-down list.
- f) 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 [Size limit on Sequence view option](#) in the **XQuery** preferences page. If you choose **Perform FO Processing** in the **FO Processor** tab, the **Sequence** option is ignored.

- g) 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 transformation scenario.

To view a more complex value returned by the query that cannot be entirely displayed in the XQuery query result table at the bottom of the Oxygen XML Developer plugin window (for example, an XMLTYPE or CLOB value), do the following:

- Right-click 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 Developer plugin).

Related information

[XML Transformation with XQuery](#) on page 516

This type of transformation specifies the transform parameters and location of an XQuery file that is applied to the edited XML document.

XQuery Validation

With Oxygen XML Developer plugin, you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.6.0.7 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.

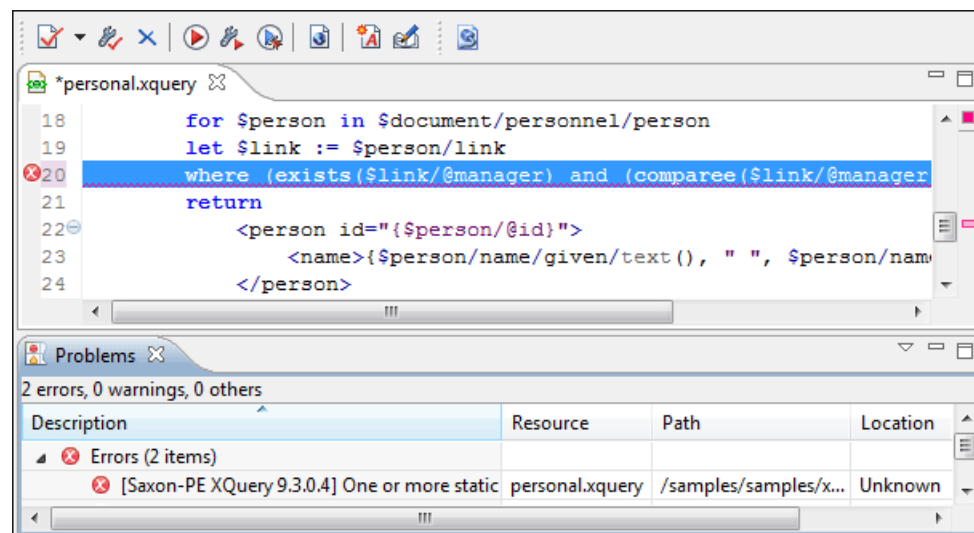


Figure 348: XQuery Validation

 **Note:** If you choose a processor that does not support XQuery validation, Oxygen XML Developer plugin displays a warning when trying to validate.

When you open an XQuery document from a connection that supports validation (for example, MarkLogic, or eXist), by default Oxygen XML Developer plugin uses this connection for validation. If you open an XQuery file using a MarkLogic connection, the validation better resolves imports.

XQuery Database Debugging

Oxygen XML Developer plugin includes a debugging interface that helps you to detect and solve problems with XQuery transformations that are executed against MarkLogic and Berkeley DB XML databases.

For more information about the debugging support in Oxygen XML Developer plugin, see [Debugging XSLT Stylesheets and XQuery Documents](#) on page 757.

Debugging with MarkLogic

Oxygen XML Developer plugin includes support for debugging XQuery transformations that are executed against a MarkLogic database.

To use a debugging session against the MarkLogic engine, follow these steps:

1. Configure a [MarkLogic data source](#) and a [MarkLogic connection](#).
2. Make sure that the debugging support is enabled in the MarkLogic server that Oxygen XML Developer plugin accesses. On the server side, debugging must be activated in the XDBC server and in the *Task Server* section of the server control console (the switch *debug allow*). If the debugging is not activated, the MarkLogic server reports a DBG - TASKDEBUGALLOW error.



Note: An XDBC application server must be running to connect to the MarkLogic server and this XDBC server will be used to process XQuery expressions against the server. You can change the XDBC application server that Oxygen XML Developer plugin uses to process XQuery expressions by selecting the [Use it to execute queries action](#) from the contextual menu in the **Data Source Explorer** view.

3. Open the XQuery file and start the debugging process.
 - If you want to debug an XQuery file stored on the MarkLogic server, we recommend you to use the [Data Source Explorer view](#) and open the file from the application server that is involved in the debugging process. This improves the resolving of any imported modules.
 - The MarkLogic XQuery debugger integrates seamlessly into the [XQuery Debugger perspective](#). If you have a MarkLogic validation scenario configured for the XQuery file, you can choose to [debug the scenario](#) directly.
 - Otherwise, 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.



Note: Before starting a debugging session, it is 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** view and run debugging sessions from them.

In a MarkLogic debugging session, you can use step actions and breakpoints to help identify problems. When you add a breakpoint on a line where the debugger never stops, Oxygen XML Developer plugin displays a warning message. These warnings are displayed for breakpoints you add either in the main XQuery (which you can open locally or from the server) or for breakpoints you add in any XQuery that is opened from the connection that participates in the debugging session. For more information, see [Using Breakpoints for Debugging Queries that Import Modules with MarkLogic](#) on page 717.

Remote Debugging with MarkLogic

Oxygen XML Developer plugin allows you to debug remote applications that use XQuery (for example, web applications that trigger XQuery executions). Oxygen XML Developer plugin connects to a MarkLogic server, shows you the running XQuery scripts and allows you to debug them. You can even pause the scripts so that you can start the debugging queries in the exact context of the application. You can also switch a server to debug mode to intercept all XQuery scripts.

Oxygen XML Developer plugin also 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.



Important: When using the remote debugging feature, the HTTP and the XDBC servers involved in the debugging session must have the same module configuration.

To watch our video demonstration about the XQuery debugger for MarkLogic, go to <https://www.oxygenxml.com/demo/XQueryDebuggerforMarkLogic.html>.

Related information

[MarkLogic Development in Oxygen XML Developer plugin](#) on page 715

[Configuring a MarkLogic Database Connection](#) on page 713

Using Breakpoints for Debugging Queries that Import Modules with MarkLogic

When debugging queries that imports modules stored in the database, it is recommended to place *breakpoints* in the modules. 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 all the modules will be considered. Also, make sure that there are no other opened modules that are not involved in the current debugging session.

To place *breakpoints* in the modules, use the following procedure:

1. In the **Data Source Explorer** view, open all the modules from the  **Modules** container of the *XDBC application server* that performs the debugging.
2. [Set breakpoints](#) in the module as needed.
3. [Continue debugging](#) the query.

If you get a warning that the breakpoints failed to initialize, try the following solutions:

- Check the [Breakpoints view](#) and make sure there are no older breakpoints (set on resources that are not part of the current debugging context).
- Make sure you open the modules from the context of the application server that does the debugging and place breakpoints there.

Related information

[MarkLogic Database Connections](#) on page 712

[MarkLogic Development in Oxygen XML Developer plugin](#) on page 715

Peculiarities and Limitations of the MarkLogic Debugger

MarkLogic debugger has the following peculiarities and limitations:

- Debugging support is only available 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'.
- 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 only 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** view.
 - 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 with Berkeley DB XML

The Berkeley DB XML database added a debugging interface starting with version 2.5. The current version is supported in the Oxygen XML Developer plugin XQuery Debugger. [The same restrictions and peculiarities](#) apply for the Berkeley debugger as for the MarkLogic debugger.

Content Management System (CMS) Integration

This chapter explains how you can use Oxygen XML Developer 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 Developer plugin.

Related information

[Working with Databases](#) on page 687

Integration with Documentum (CMS) (deprecated)

Important: Starting with version 17.0, the support for Documentum (CMS) is deprecated and will no longer be actively maintained.

Oxygen XML Developer plugin provides support for browsing and managing Documentum (CMS) connections in the [Data Source Explorer](#) view. You can easily create new resources on the repository, copy and move them using contextual actions or the drag and drop support, or edit and transform the documents in the editor.

Oxygen XML Developer plugin supports Documentum (CMS) version 6.5 and 6.6 with *Documentum Foundation Services* 6.5 or 6.6 installed.

Attention: It is recommended to use the latest 1.6.x Java version. It is possible that the Documentum (CMS) support will not work properly if you use other Java versions.

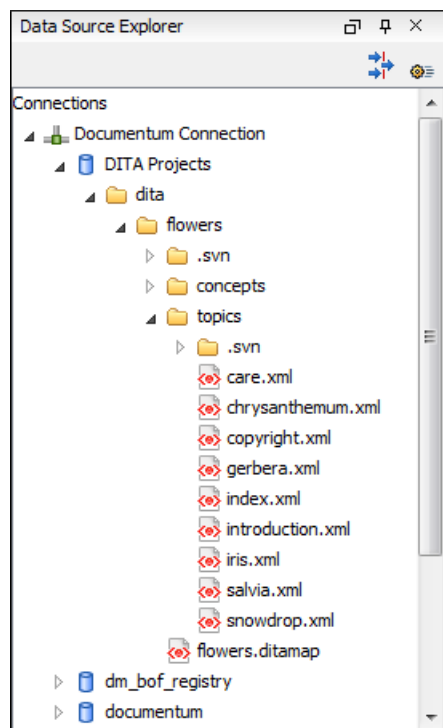


Figure 349: Documentum (CMS) Connection

Configuring a Documentum (CMS) Database Connection

Follow this procedure to configure the support for a Documentum (CMS) database:

1. [Configure Documentum xDB Data Source drivers](#).
2. [Configure a Documentum xDB Connection](#).
3. To view your connection, go to the **Data Source Explorer** view (if the view is not displayed, it can be opened from the **Window > Show View** menu) or switch to the **Database Perspective** (available from **Window > Open Perspective > Database**).

How to Configure Documentum (CMS) Data Source Drivers

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 Documentum (CMS) 6.5 (for example, *emc-dfs-sdk-6.5.zip*).

To configure a data source for Documentum (CMS), follow these steps:

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.
8. Continue on to [configure your Documentum connection](#).

How to Configure a Documentum (CMS) Connection

Available in the Enterprise edition only.

To configure a connection for a Documentum (CMS) server, 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 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 with Documentum (CMS)

The following are known issues with the Documentum (CMS):

- There is a known problem in the UCF Client implementation for Mac OS X from Documentum 6.5 that 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.

- For the Documentum driver to work faster on Linux, you need to specify to the JVM to use a weaker random generator, instead of the very slow native implementation. This can be done by modifying in the Oxygen XML Developer plugin startup scripts (or in the *.vmoptions file) the system property:

```
-Djava.security.egd=file:/dev/./urandom
```

Documentum (CMS) Contextual Menu Actions

While browsing Documentum (CMS) connections in the **Data Source Explorer** view, the various nodes include the following contextual menu actions:

Connection Level Nodes



Configure Database Sources

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Disconnect

Stops the connection.

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

Performs a refresh on the selected node.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.

Container (Cabinet) Level Nodes



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.

Export

Allows you to export the folder on the remote connection to a local folder.

Rename

Renames the current resource



Delete

Deletes the current container.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Resource Level Nodes



Edit

Checks out and opens the selected resource in the editor (if it is not already checked out).

Edit with

Checks out and opens the selected resource in the specified editor or tool (if it is not already checked out).

Open (Read-only)

Opens the selected resource in the editor.

Open with

Opens the selected resource in the specified editor or 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

Opens the **Check In** dialog box that allows you to check in the selected resource (commits changes) into the repository and configure some properties for the resource. The action is only available if the resource is checked out.

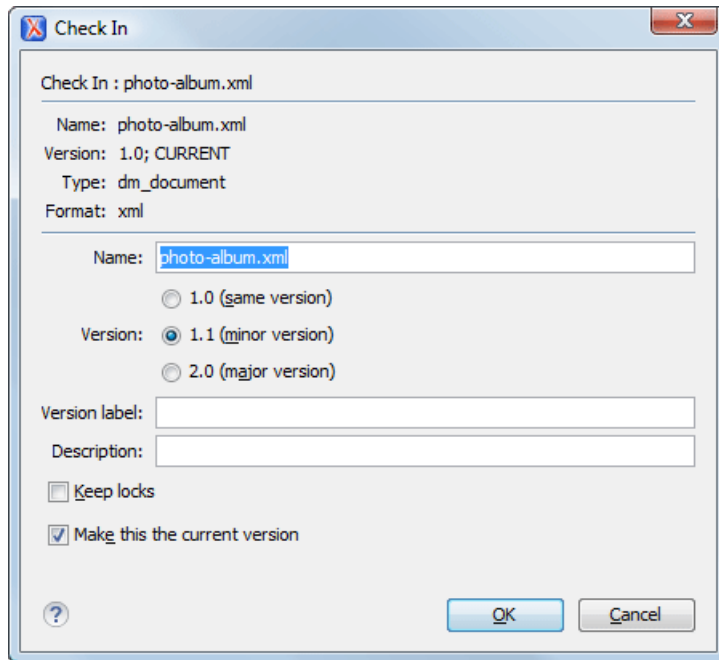


Figure 350: Check In Dialog Box

The following resource properties can be configured in this dialog box:

- **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 folder on the remote connection to a local folder.

Copy

Copies the selected resource to another location in the tree. This 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 to another 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.

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.

Rename

Renames the current resource

✕ Delete

Deletes the current container.

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.

Refresh

Performs a refresh on the selected node.

Properties

Shows various properties of the current container.

Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.

Compare

Compares two selected resources.

Integration with Microsoft SharePoint

Oxygen XML Developer plugin provides support for browsing and managing SharePoint connections in the [Data Source Explorer view](#). You can easily create new resources on the repository, copy and move them using contextual actions or the drag and drop support, or edit and transform the documents in the editor.

 **Note:** You can access documents stored on SharePoint Online for Office 365 sites that use either *Cloud identity (default)* or *Federated identity (ADFS)* as the authentication method.

 **Restriction:** The SharePoint connection is only available in the Enterprise edition of Oxygen XML Developer plugin.

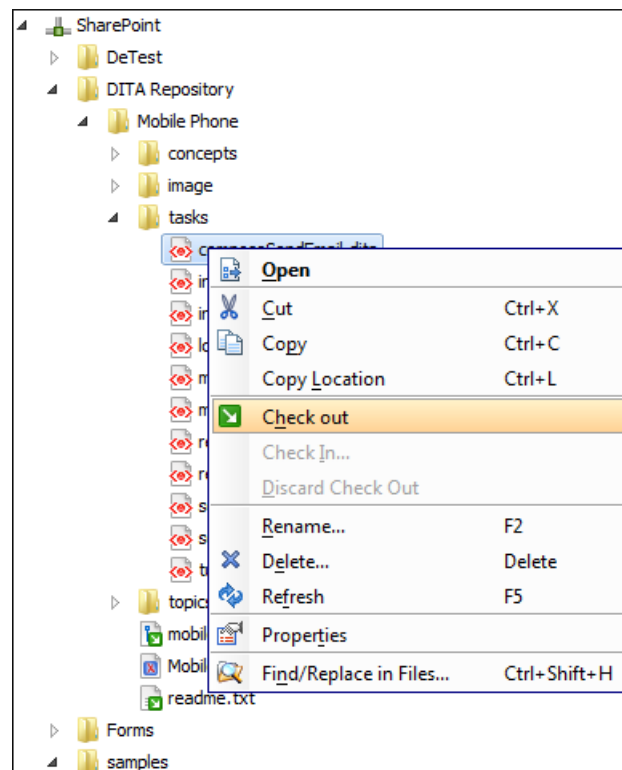


Figure 351: SharePoint Connection

Related information

[Working with Databases](#) on page 687

How to Configure a SharePoint Connection

By default, Oxygen XML Developer plugin contains built-in data source drivers for **SharePoint**. Use this data source to create a connection to a SharePoint server that will be available in the [Data Source Explorer view](#).

To configure a SharePoint connection, 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 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. If you are using a SharePoint 365 account, leave this field empty.
 - 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 https://www.oxygenxml.com/demo/SharePoint_Support.html.

SharePoint Contextual Menu Actions

While browsing SharePoint connections in the **Data Source Explorer** view, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the [Data Sources preferences page](#) where you can configure both data sources and connections.

Disconnect

Stops the connection.

New Folder

Creates a new folder on the connection.

Import Files

Allows you to add a new file on the connection, in the current folder.

Refresh

Performs a refresh on the selected node.

Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.

Folder Level Nodes

New File

Creates a new file on the connection, in the current folder.

New Folder

Creates a new folder on the connection.

Import Folders

Imports folders on the server.

Import Files

Allows you to add a new file on the connection, in the current folder.

Export

Allows you to export the folder on the remote connection to a local 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

Renames the current resource



Delete

Deletes the current container.



Refresh

Performs a refresh on the selected node.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.



Resource Level Nodes

Open

Opens the selected resource in the editor.



Cut

Removes the current selection and places it in the clipboard.



Copy

Copies the current selection into the clipboard.

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.

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

Renames the current resource



Delete

Deletes the current container.



Refresh

Performs a refresh on the selected node.



Properties

Shows various properties of the current container.



Find/Replace in Files

Allows you to find and replace text in multiple files from the connection.

Compare

Compares two selected resources.

Chapter

13

Importing Data

Topics:

- [Import from Text Files](#)
- [Import from MS Excel Files](#)
- [Import Database Data as an XML Document](#)
- [Import from HTML Files](#)
- [Import Content Dynamically](#)

Computer systems and databases contain data in incompatible formats and exchanging data between these systems can be very time consuming. Converting the data to XML can greatly reduce the complexity and create data that can be read by various types of applications.

Oxygen XML Developer plugin offers support for importing text files, MS Excel files, Database Data, and HTML files into XML documents. The XML documents can be further converted into other formats using the [Transform features](#).

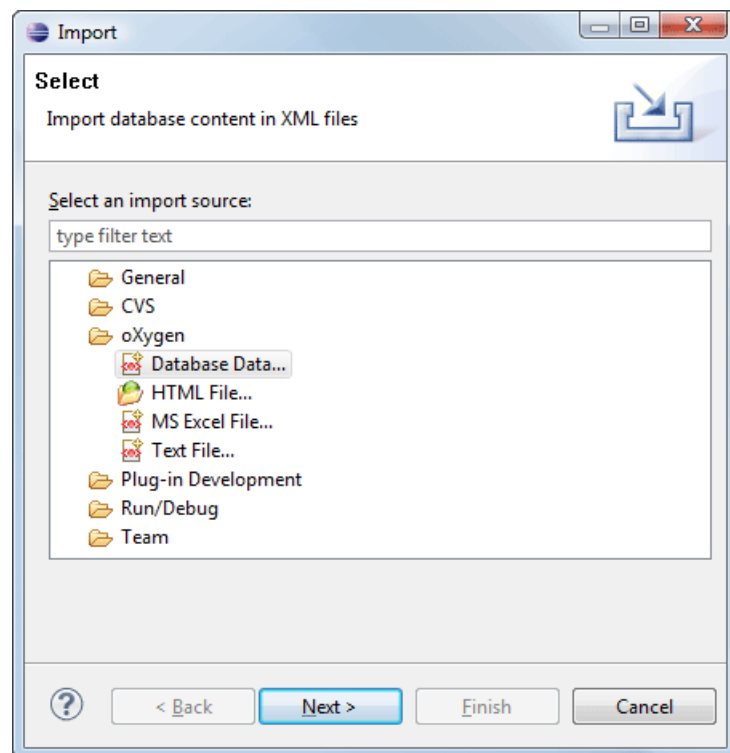


Figure 352: Import Wizards of the Oxygen XML Developer plugin Plugin

Import from Text Files

To import a text file into an XML file, follow these steps:

1. Go to **File > Import > Oxygen XML Developer plugin > Text File** and click **Next**.
A **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.

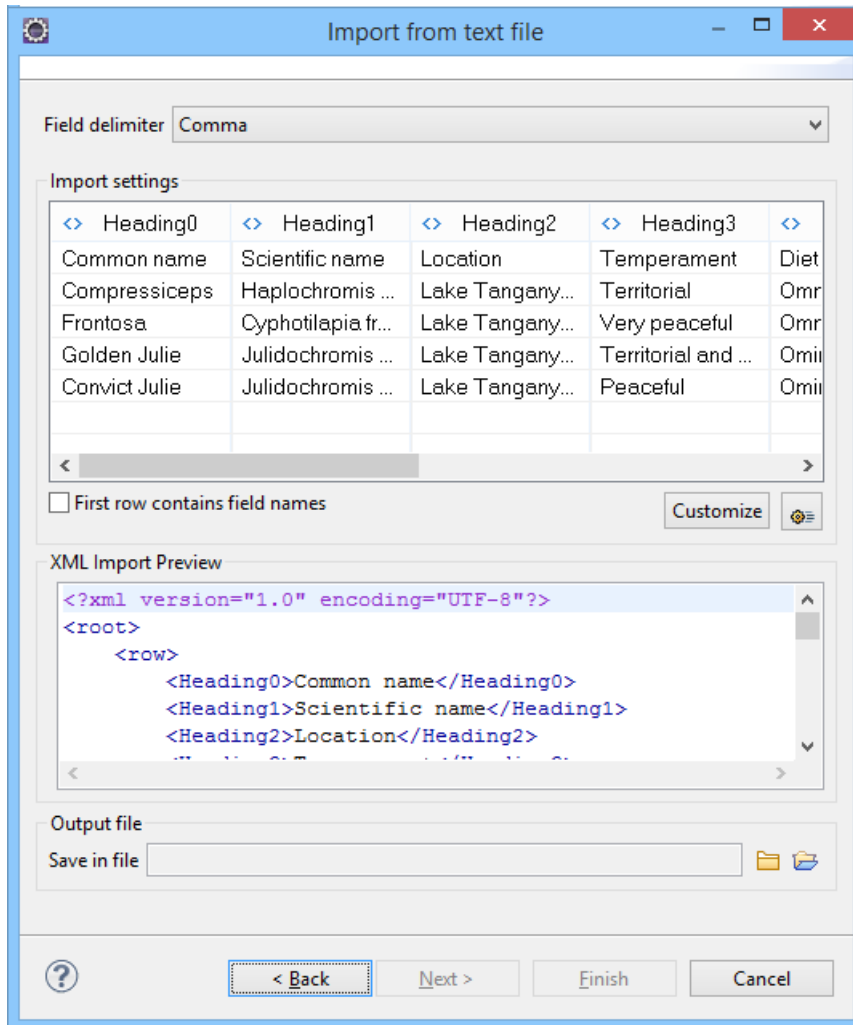


Figure 353: Import from Text File Dialog Box

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 (x 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 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 751 for instructions on adding more libraries.

Oxygen XML Developer 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 Developer 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.

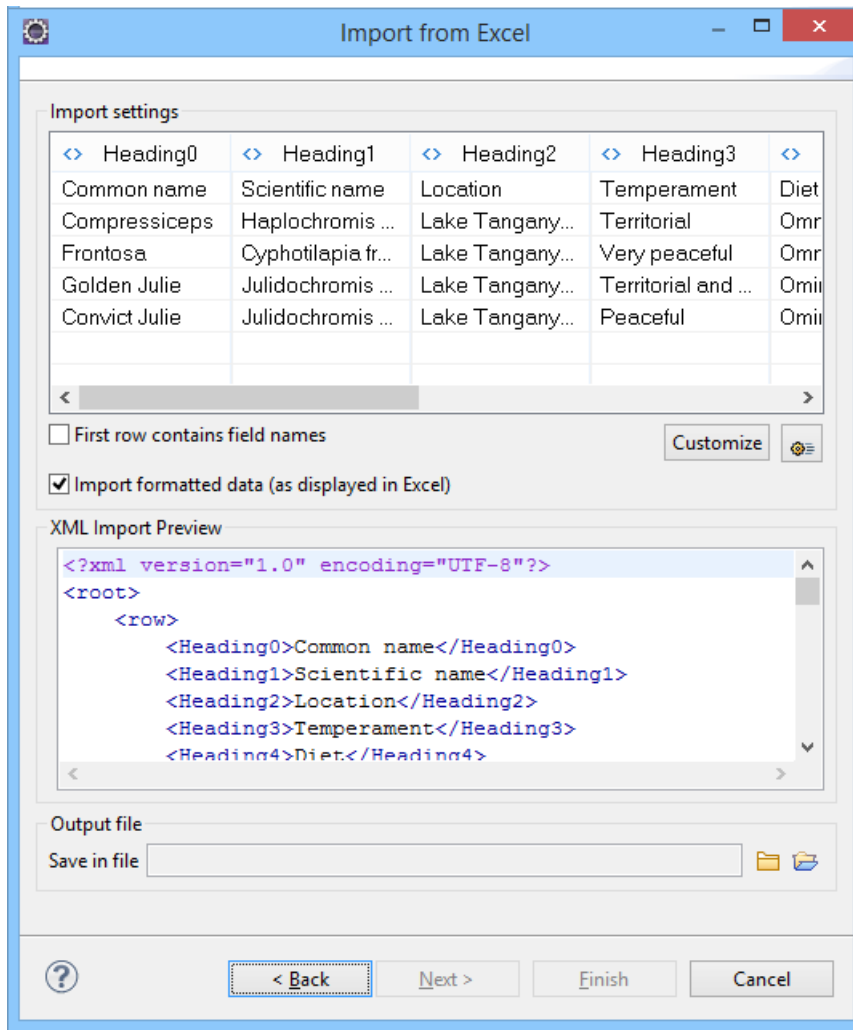


Figure 354: Import from Excel Dialog Box

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 (x 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 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 Developer plugin needs additional libraries from the release 3.10 of the Apache POI project.

To add the 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 Developer 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:

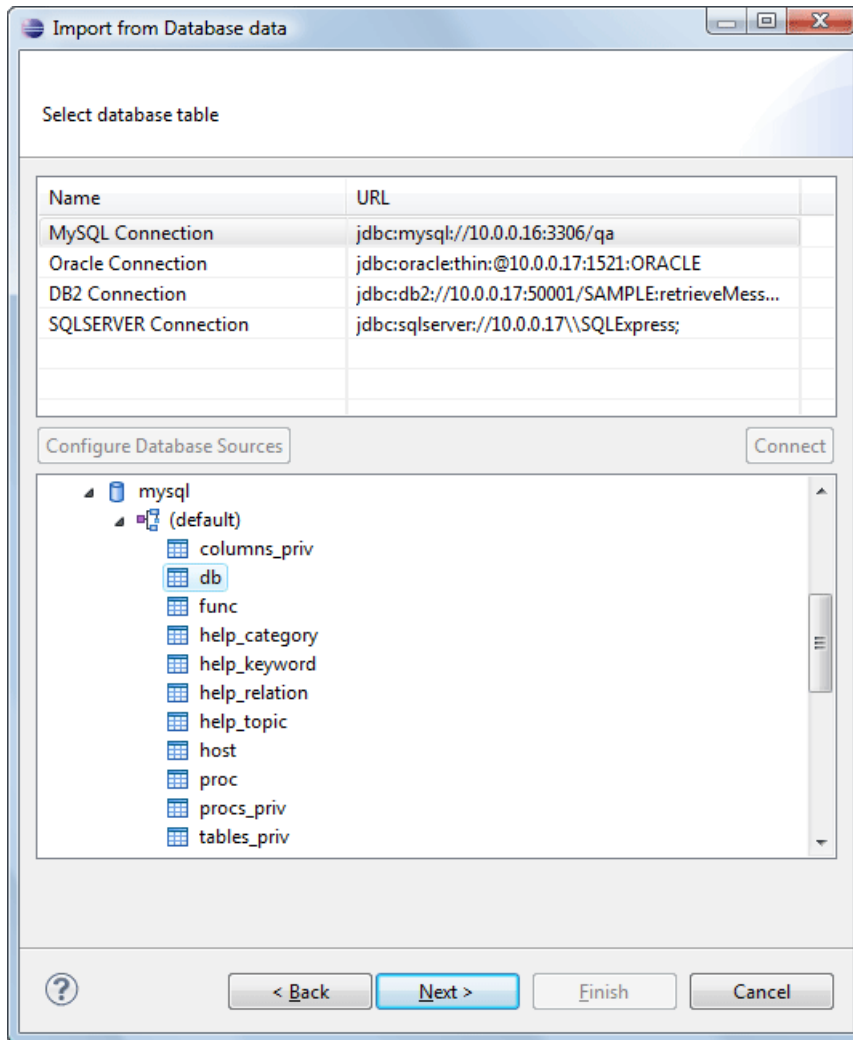


Figure 355: Select Database Table Dialog Box

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.

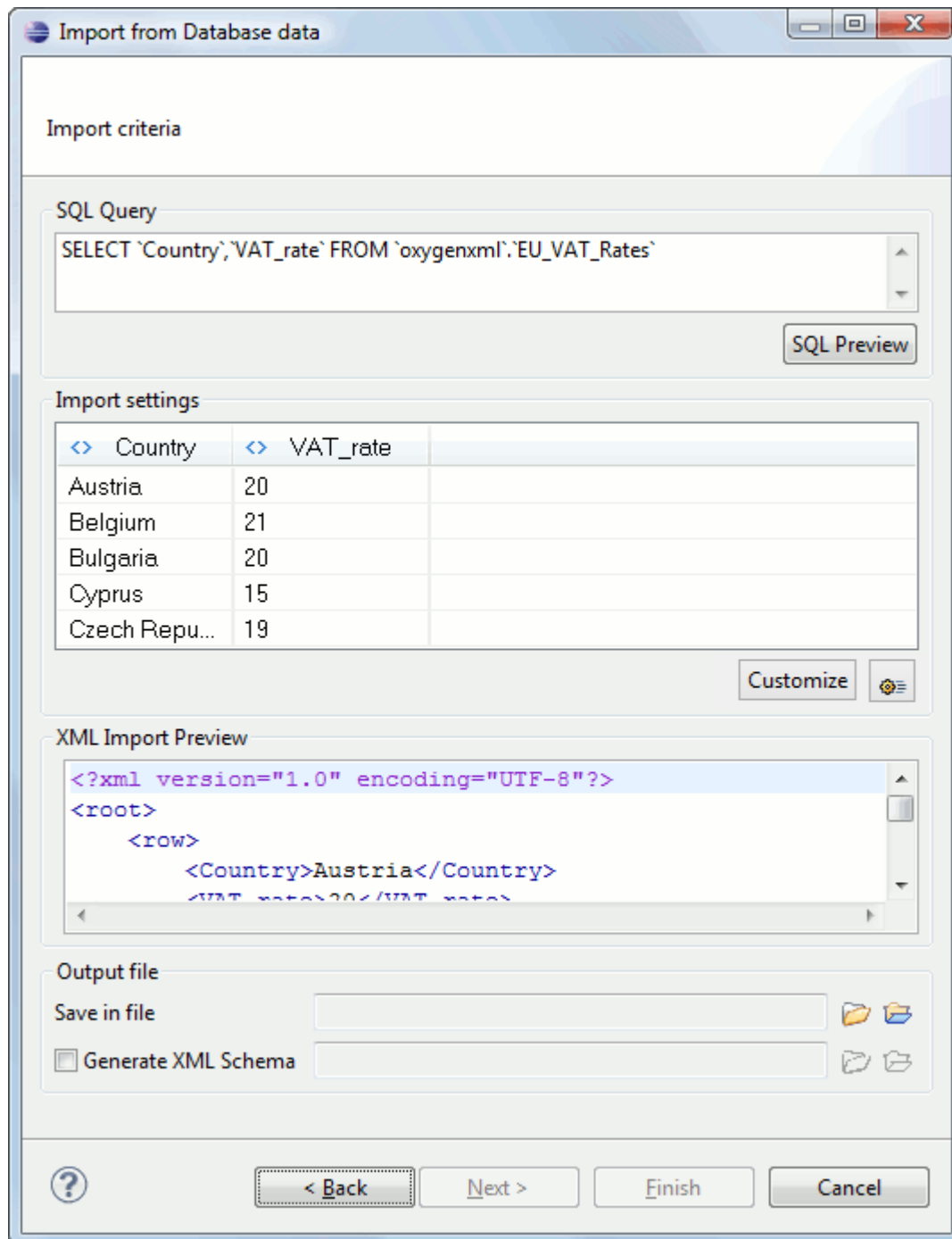


Figure 356: Import from Database Criteria Dialog Box

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 (x 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 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 Developer 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 Developer 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 Developer plugin also provides special support for a *convert* protocol that can be used to chain predefined processors to dynamically import content from various sources.

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 (`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 resulting content 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>
```

The target resource part of the conversion URL must always follow the `!/` pattern. It can be any of the following:

- An absolute URL that points to a resource.
- An identifier that will be resolved to an actual resource via the XML Catalog support in the application. In the example above, the `urn:files:sample.xls` target resource is resolved via the XML catalog.
- A relative location. This location can only be resolved to an actual resource URL when the application has enough information about the location where the URL is referenced.

For example, for a DITA map with a `topicref` such as:

```
<topicref href="convert:/.../processor=excel!/resources/sample.xls"/>
```

the `resources/sample.xls` path will be resolved relative to the DITA map location.

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 as a topic reference from a *DITA map*.

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 Developer plugin. 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 the Saxon EE XSLT processor. 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 the Saxon EE XQuery processor. 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 later be 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 that libraries will be loaded from. 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
```

- **cache Processor** - Caches the converted content obtained from the original document to a temporary file. The cache will be used on subsequent uses of the same URL, thus increasing the speed for the application returning the converted content. If the original URL points to the local disk, the cache will be automatically invalidated when the original file content gets modified. Otherwise, if the original URL points to a remote resource, the cache will need to be invalidated by reloading (**File** >  **Reload (F5)**) the URL content that is opened in the editor.

```
convert:/processor=cache/processor=xslt;...!/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 Developer 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 **Libraries** button in the **Advanced** tab of the transformation scenario.

Related information

<https://github.com/oxygenxml/dita-glass>

Chapter

14

Debugging XSLT Stylesheets and XQuery Documents

Topics:

- [Debugger Layout](#)
- [Working with the XSLT / XQuery Debugger](#)
- [Debugging Java Extensions](#)
- [Supported Processors for XSLT / XQuery Debugging](#)
- [Performance Profiling of XSLT Stylesheets and XQuery Documents](#)

Oxygen XML Developer plugin includes a powerful debugging interface that helps you to detect and solve problems with XSLT and XQuery transformations.

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 **Window > Open Perspective > Other > Oxygen XSLT Debugger**.

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 **Window > Open Perspective > Other > Oxygen XQuery Debugger**.

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.

Debugger Layout

The XML and XSL files are displayed in *Text mode*. The *Grid mode* is available only in *the Editor perspective*.

The debugger perspective contains four sections:

- **Source document view (XML)** - Displays and allows the editing of XML files (documents).
- **XSLT/XQuery document view (XSLT/XQuery)** - Displays and allows the editing of XSL files(stylesheets) or XQuery documents.
- **Output 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. Several actions are available in the contextual menu for this view, including **Find/Replace**, **Copy**, and **Format and Indent**.
- **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*

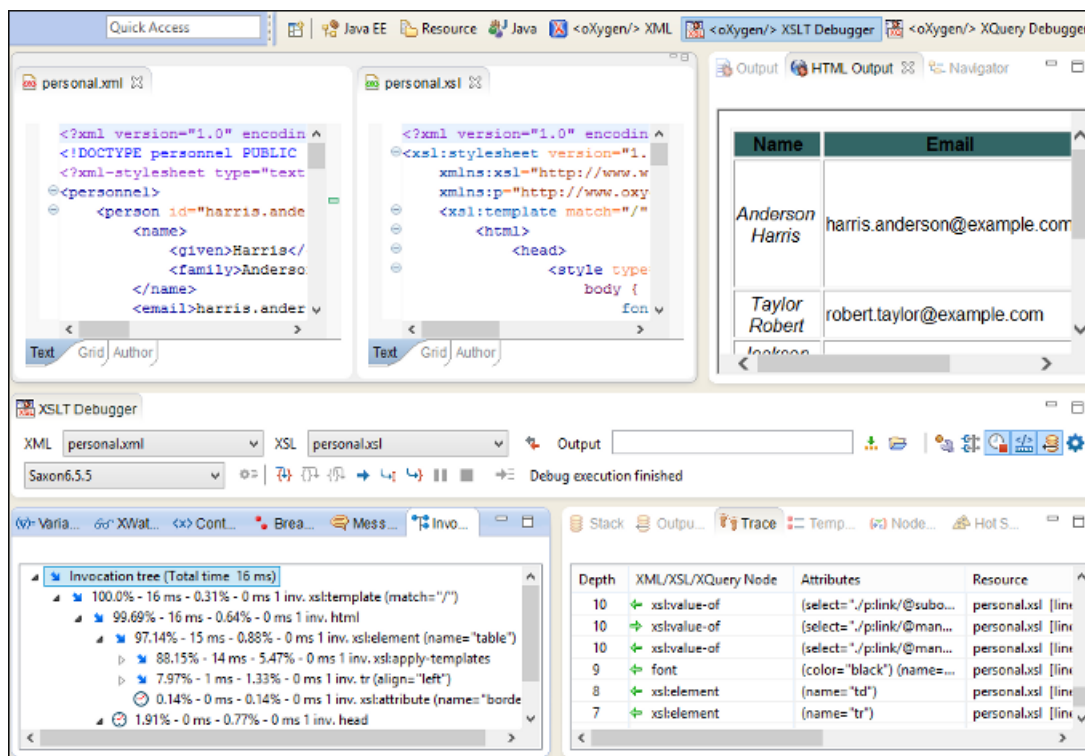


Figure 357: Debugger Mode Interface

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.

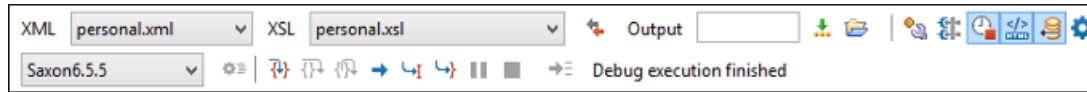


Figure 358: Control Toolbar

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. You can specify the path by using the text field, the  **Insert Editor Variables** button, or the  **Browse** button.

Configure parameters

Opens a dialog box that allows you to configure the XSLT / XQuery parameters to be used by the transformation.

Libraries

Allows you to add and remove the Java classes and jars used as XSLT extensions.

Turn on/off 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. 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 (Command + 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**.

Related information

[Advanced Saxon HE/PE/EE XQuery Transformation Options](#) on page 518

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. By using the debug controls, developers can easily isolate parts of stylesheet. Therefore, they may be more easily 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 **Context Node** view. If the view is not displayed, it can be opened from the **Window > Show View** menu.

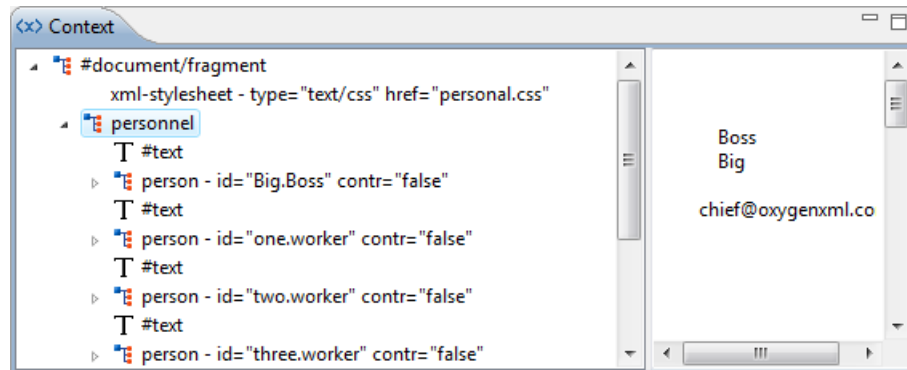


Figure 359: Context node view

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. If the view is not displayed, it can be opened from the **Window > Show View** menu.

Expressions are evaluated dynamically as the processor changes its source context. When you type an XPath expression in the **Expression** column, Oxygen XML Developer plugin supports you with syntax highlight and [content completion assistance](#).

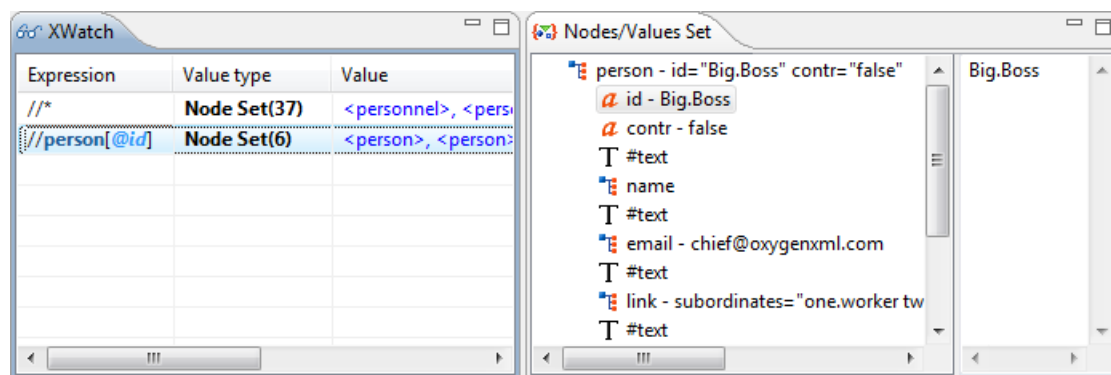


Figure 360: XPath Watch View

Table 32: XWatch columns

Column	Description
Expression	XPath expression to be evaluated (XPath 1.0 or 2.0 / 3.0 compliant).

Column	Description
Value	Result of XPath expression evaluation. Value has a type (see the possible values in the section Variables View on page 768). For <i>Node Set</i> 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

The **Breakpoints** view lists all breakpoints that are set on opened documents. If the view is not displayed, it can be opened from the **Window > Show View** menu. [Breakpoints can be inserted](#) in XSLT/XQuery documents and XML documents in XSLT/XQuery debugging sessions.

Once you insert a breakpoint, it is automatically added to the list in the **Breakpoints** view and you can edit its associated breakpoint *condition*. 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**.

Ena...	Resource	Condition
☒	personal.xml: 11	count(preceding::person)=2
☒	personal.xml: 9	local-name()='person'
☒	personal.xml: 8	(No condition)
☒	personal.xml: 6	(No condition)

Figure 361: 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.

Run to Breakpoint

Runs the debugger up to the point of this particular breakpoint and ignores the others (regardless of whether they were previously enabled or disabled).

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.

Related tasks

[Inserting Breakpoints](#) on page 770

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. The **Messages** view is available only for XSLT debugging sessions and shows all `xsl:message` calls executed by the XSLT processor during transformation. If the view is not displayed, it can be opened from the **Window > Show View** menu.

Message	Terminate	Resource
Message 1	no	personal.xml [line: 8]
Message 2	no	personal.xml [line: 12]
Message 3	no	personal.xml [line: 29]

Figure 362: Messages View

Table 33: Messages columns

Column	Description
Message	Message content.
Terminate	Signals if processor terminates the transformation or not once it encounters the message (yes/no respectively).
Resource	Resource file where <code>xsl:message</code> instruction is defined and the message line number.

The following actions are available in the contextual menu:

Go to

Highlight the XSL fragment that generated the message.

Copy

Copies to clipboard message details (system ID, severity info, description, start location, terminate state).

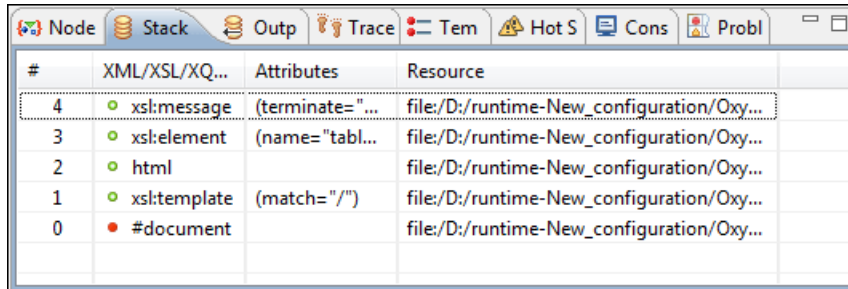
**Important:** Remarks

- 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

The **Stack** view shows the current execution stack of both source and XSLT/XQuery nodes. If the view is not displayed, it can be opened from the **Window > Show View** menu.

During transformation two stacks are managed: one of source nodes being processed and the other for XSLT/XQuery nodes being processed. Oxygen XML Developer plugin shows both node types into one common stack. The source (XML) nodes are preceded by a red color icon while XSLT/XQuery nodes are preceded by a green color icon. The advantage of this approach is that you can always see the source scope on which an XSLT/XQuery instruction is executed (the last red color node on the stack). The stack is oriented upside down.



#	XML/XSL/XQ...	Attributes	Resource
4	xsl:message	(terminate= "...	file:/D:/runtime-New_configuration/Oxy...
3	xsl:element	(name="tabl...	file:/D:/runtime-New_configuration/Oxy...
2	html		file:/D:/runtime-New_configuration/Oxy...
1	xsl:template	(match="/")	file:/D:/runtime-New_configuration/Oxy...
0	#document		file:/D:/runtime-New_configuration/Oxy...

Figure 363: 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 34: Stack columns

Column	Description
#	Order number, represents the depth of the node (0 is the stack base).
XML/XSLT/XQuery Node	Node from source or stylesheet document currently being processed. One particular stack node is the document root, noted as #document .
Attributes	Attributes of the node (a list of <code>id="va lue"</code> pairs).
Resource	Resource file where the node is located.



Important: Remarks:

- Clicking a record from the stack highlights that node's location inside resource.
- Using Saxon, the stylesheet elements are qualified with XSL proxy, while using Xalan you only see their names. (example: `xsl:template` using Saxon and `template` using Xalan).
- Only the Saxon processor shows element attributes.
- The Xalan processor shows also the built-in rules.

Output Mapping Stack View

The **Output Mapping Stack** view displays *context data* and presents the XSLT templates/XQuery elements that generated specific areas of the output. If the view is not displayed, it can be opened from the **Window > Show View** menu.

#	XML/XSL/XQuery Node	Attributes	Resource
8	xsl:value-of	(select="email/text()")	file:/D:/runtime-Eclipse
7	font	(name="verdana") (size="3")	file:/D:/runtime-Eclipse
6	xsl:element	(name="td")	file:/D:/runtime-Eclipse
5	xsl:element	(name="tr")	file:/D:/runtime-Eclipse
4	xsl:template	(match="//person")	file:/D:/runtime-Eclipse
3	xsl:apply-templates		file:/D:/runtime-Eclipse
2	xsl:element	(name="table")	file:/D:/runtime-Eclipse
1	html		file:/D:/runtime-Eclipse
0	xsl:template	(match="/")	file:/D:/runtime-Eclipse

Figure 364: 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 35: 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.

Related tasks

[Identify the XSLT / XQuery Expression that Generated Particular Output](#) on page 771

Related information

[Stack View](#) on page 764

[Trace History View](#) on page 765

[Templates View](#) on page 766

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 an XSLT/XQuery node.
-  - Leaving an XSLT/XQuery node.

The **Trace History** view catches all these events, so you can see how the process evolved. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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. Thus, you have the possibility of comparing the trace results from multiple debug sessions.

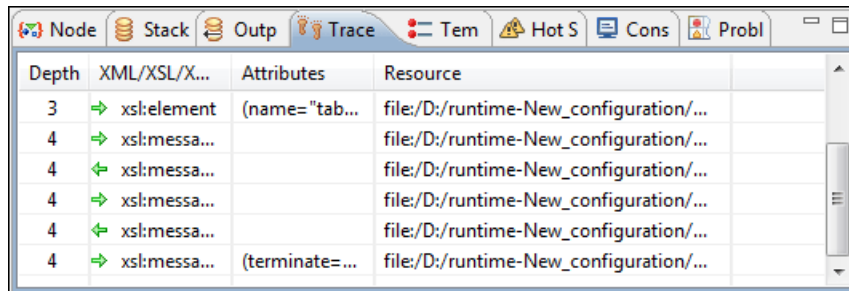


Figure 365: Trace History View

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 36: 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 <code>#document</code> . 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 <code>id="va lue"</code> 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. The **Templates** view is only available during XSLT debugging sessions and shows all `xsl:template` instructions used by the transformation. If the view is not displayed, it can be opened from the **Window > Show View** menu.

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.

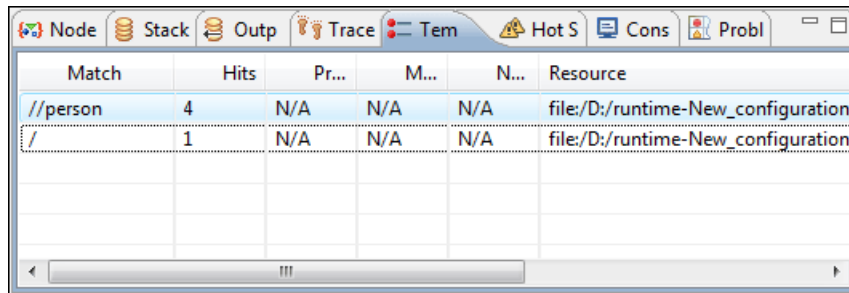


Figure 366: Templates view

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 37: Templates columns

Column	Description
Match	The <code>match</code> attribute of the <code>xsl:template</code> .
Hits	The number of hits for the <code>xsl:template</code> . Shows how many times the XSLT processor used this particular template.
Priority	The template priority as established by XSLT processor.
Mode	The <code>mode</code> attribute of the <code>xsl:template</code> .
Name	The <code>name</code> attribute of the <code>xsl:template</code> .
Resource	The resource file where the template is located.



Important: Remarks:

- Clicking a record highlights that template definition inside the resource.
- Saxon only shows the applied templates having at least one hit from the processor. Xalan shows all defined templates, with or without hits.
- Template table values can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in rules.

Nodes/Values Set View

The **Nodes/Values Set** view is always used in relation with [The Variables view](#) and [the XWatch view](#). If the view is not displayed, it can be opened from the **Window > Show View** menu. 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.

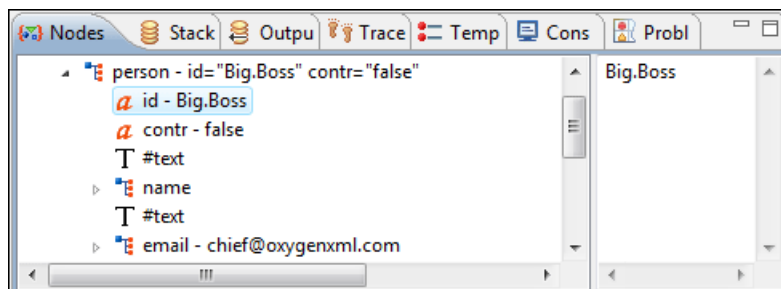


Figure 367: Node Set view

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

The **Variables** view displays variables and parameters (local and global), along with their values. If the view is not displayed, it can be opened from the **Window > Show View** menu.

Variables and parameters play an important role during an XSLT/XQuery transformation. Oxygen XML Developer plugin uses the following icons to differentiate variables and parameters:

-  - Global variable.
-  - Local variable.
-  - Global parameter.
-  - 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.

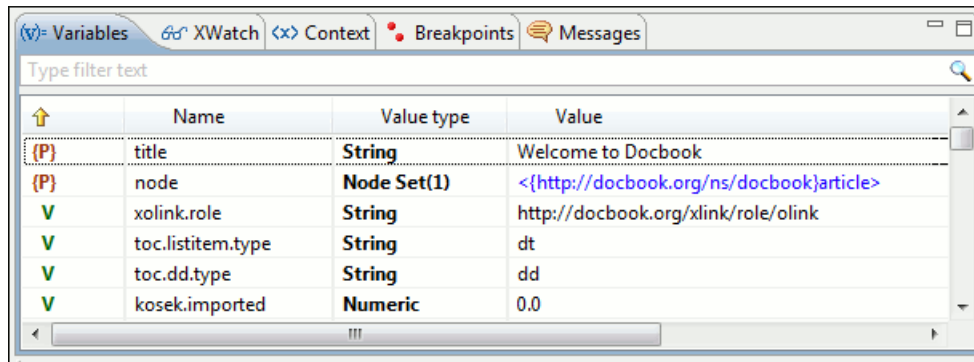


Figure 368: Variables View

Table 38: 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 that 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 multiple files 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 tab for each `xsl:result-document` instruction in the **Result Documents** view so that the output is not mixed while still being presented in multiple 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 https://www.oxygenxml.com/demo/XSLT_Debugger.html.

Steps in a Typical Debugging Process

To debug a stylesheet or XQuery document, follow this procedure:

1. Open the source XML document and the XSLT/XQuery document.
2. If you are in the Editor perspective, switch to the XSLT Debugger or XQuery Debugger perspective with one of the following actions:

- Select **Window > Open Perspective > Other > Oxygen XSLT Debugger/XQuery Debugger**.
 - Select the  **Debug scenario** action on the toolbar.. 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.
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 following 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.

Related tasks

Identify the XSLT / XQuery Expression that Generated Particular Output on page 771

Using Breakpoints

The Oxygen XML Developer plugin XSLT/XQuery Debugger allows you to interrupt XSLT/XQuery processing to gather information about variables and processor execution at particular points. To ensure breakpoints are persistent between work sessions, they are saved at project level. You can set a maximum of 100 breakpoints per project.

Related information

Breakpoints View on page 762

Inserting Breakpoints

Breakpoints can be set in XSLT/XQuery documents and XML documents in XSLT/XQuery debugging sessions.

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**.

Once you insert a breakpoint, it is automatically added to the list in the **Breakpoints** view and you can edit its associated breakpoint *condition*. 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**.

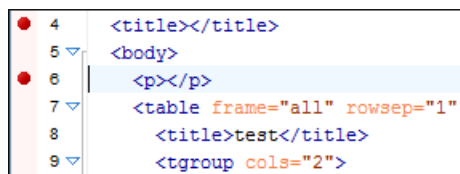


Figure 369: Example: Breakpoints

Related information

[Breakpoints View](#) on page 762

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**.

Identify the XSLT / XQuery Expression that Generated Particular Output

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.

- Switch to the XSLT Debugger or XQuery Debugger perspective with one of the following actions:
 - Select **Window > Open Perspective > Other > Oxygen XSLT Debugger/XQuery Debugger**.
 - Select the  **Debug scenario** action on the toolbar. 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.
- 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.
- Select the XSLT/XQuery document in the XSLT/XQuery selector of [the Control toolbar](#).
- Select the XSLT/XQuery engine in the XSLT/XQuery engine selector of [the Control toolbar](#).
- Set XSLT/XQuery parameters from the button available on [the Control toolbar](#).
- Apply the XSLT stylesheet or XQuery transformation using the  **Run to end** button that is available on [the Control toolbar](#).
- Inspect the mapping by clicking a section of the output from the **Output** view of the Oxygen XML Developer plugin XSLT Debugger or Oxygen XML Developer plugin XQuery Debugger perspectives.

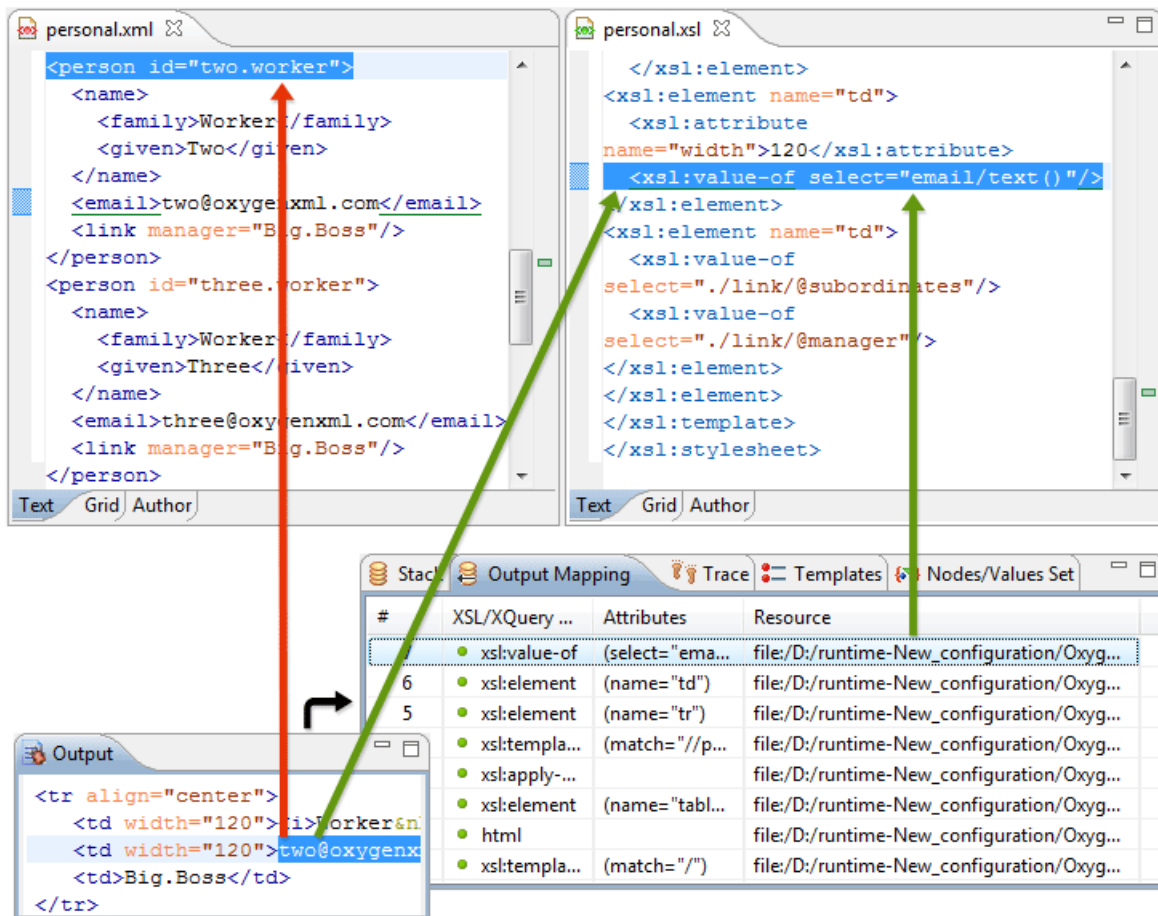


Figure 370: Text Output to Source Mapping

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](#).

Related information

[Output Mapping Stack View](#) on page 764

[Trace History View](#) on page 765

[Templates View](#) on page 766

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, you can set up a Java debug configuration in an IDE (such as 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_INSTALL_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.xml.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.

Performance Profiling of XSLT Stylesheets and XQuery Documents

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.

Working with XSLT/XQuery Profiler

Profiling activity is linked with debugging activity, so the first step in profiling is to switch to the debugging perspective and follow the corresponding procedure for debugging (see [Steps in a Typical Debugging Process](#) on page 769).

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.

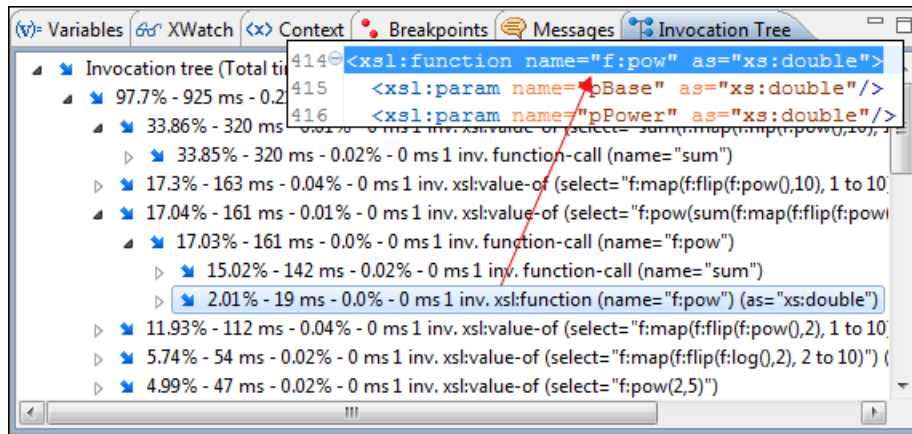


Figure 371: Source backmapping

In any of the above views you can use the backmapping feature to find the XSLT stylesheet or XQuery expression definition. Clicking the selected item cause Oxygen XML Developer plugin to highlight the XSLT stylesheet or XQuery expression source line where the instruction is defined.

When navigating through the trees by opening instruction calls, Oxygen XML Developer plugin automatically expands instructions that 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 Developer plugin distribution (see the subfolder `[OXYGEN_INSTALL_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 https://www.oxygenxml.com/demo/XSLT_Profiling.html.

Invocation Tree View

The **Invocation Tree** view shows a top-down call tree that represents how XSLT instructions or XQuery expressions are processed. If the view is not displayed, it can be opened from the **Window > Show View** menu.

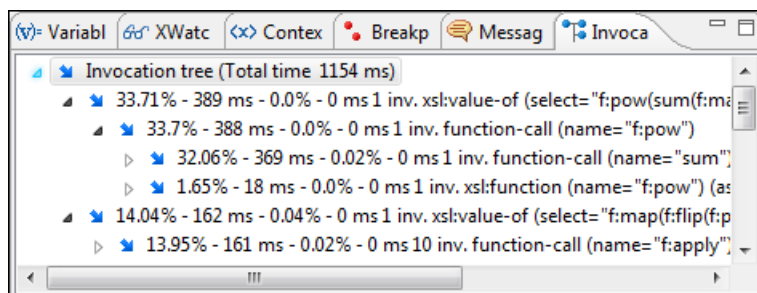


Figure 372: Invocation Tree View

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](#). If the view is not displayed, it can be opened from the **Window > Show View** menu.

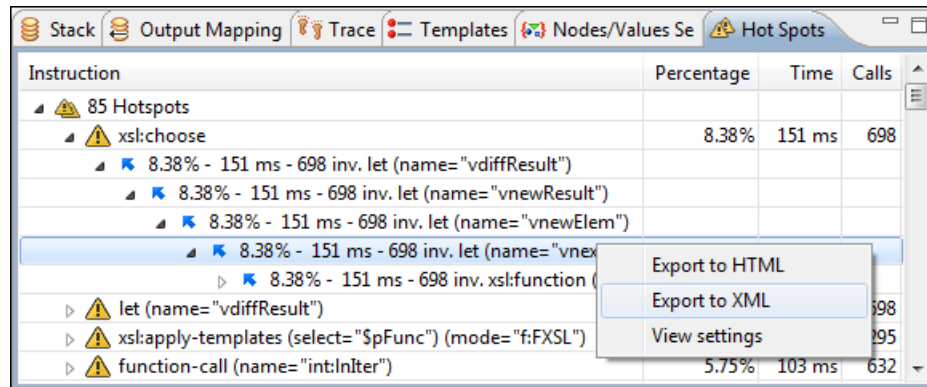


Figure 373: Hotspots View

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.

Chapter 15

Extending Oxygen XML Developer plugin Using the SDK

Topics:

- [*Extension points for Oxygen XML Developer plugin*](#)

Extension points for Oxygen XML Developer plugin

The Oxygen XML Developer plugin includes a number of extension points, which can be implemented by other Eclipse plugins that depend on it. All of them are listed in the `plugin.xml` file, along with samples of usage code. The following is a list with short descriptions for some of the most useful extension points:

Extension point: *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`.

Chapter

16

Tools

Topics:

- [*Refactoring XML Documents*](#)
- [*Generating Sample XML Files*](#)
- [*Converting Schema to Another Schema Language*](#)
- [*Converting Database to XML Schema*](#)
- [*XML to JSON Converter*](#)
- [*Generate Documentation*](#)
- [*Canonicalizing Files*](#)
- [*Signing Files*](#)
- [*Verifying Signature*](#)
- [*WSDL SOAP Analyzer*](#)
- [*XML Schema Regular Expressions Builder*](#)

Oxygen XML Developer plugin includes a variety of helpful tools to help you accomplish XML-related tasks. This section presents many of those tools. These tools are available in the **Tools** menu and some of them can be launched through keyboard shortcuts or command-line scripts.

Refactoring XML Documents

In the life cycle of XML documents there are instances when the XML structure needs to be changed to accommodate various needs. For example, when an associated schema is updated, an attribute may have been removed, or a new element added to the structure.

These types of situations cannot be resolved with a traditional *Find/Replace* tool, even if the tool accepts regular expressions. The problem becomes even more complicated if an XML document is computed or referenced from multiple modules, since multiple resources need to be changed.

To assist you with these types of refactoring tasks, Oxygen XML Developer plugin includes a specialized **XML Refactoring** tool that helps you manage the structure of your XML documents.

XML Refactoring Tool

The **XML Refactoring** tool is presented in the form of an easy to use wizard that is designed to reduce the time and effort required to perform various structure management tasks. For example, you can insert, delete, or rename an attribute in all instances of a particular element that is found in all documents within your project.

To access the tool, select the  **XML Refactoring** action from one of the following locations:

- The **XML Tools** menu.
- The **Refactoring** submenu from the contextual menu in the **Navigator** view.



Note: The predefined refactoring operations are also available from the **Refactoring** submenu in the contextual menu of **Text** mode. This is useful because by selecting the operations from the contextual menu, Oxygen XML Developer 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.

XML Refactoring Wizard

The XML Refactoring tool includes the following wizard pages:

Refactoring operations

The first wizard page presents the available operations, grouped by category. To search for an operation, you can use the filter text box at the top of the page.

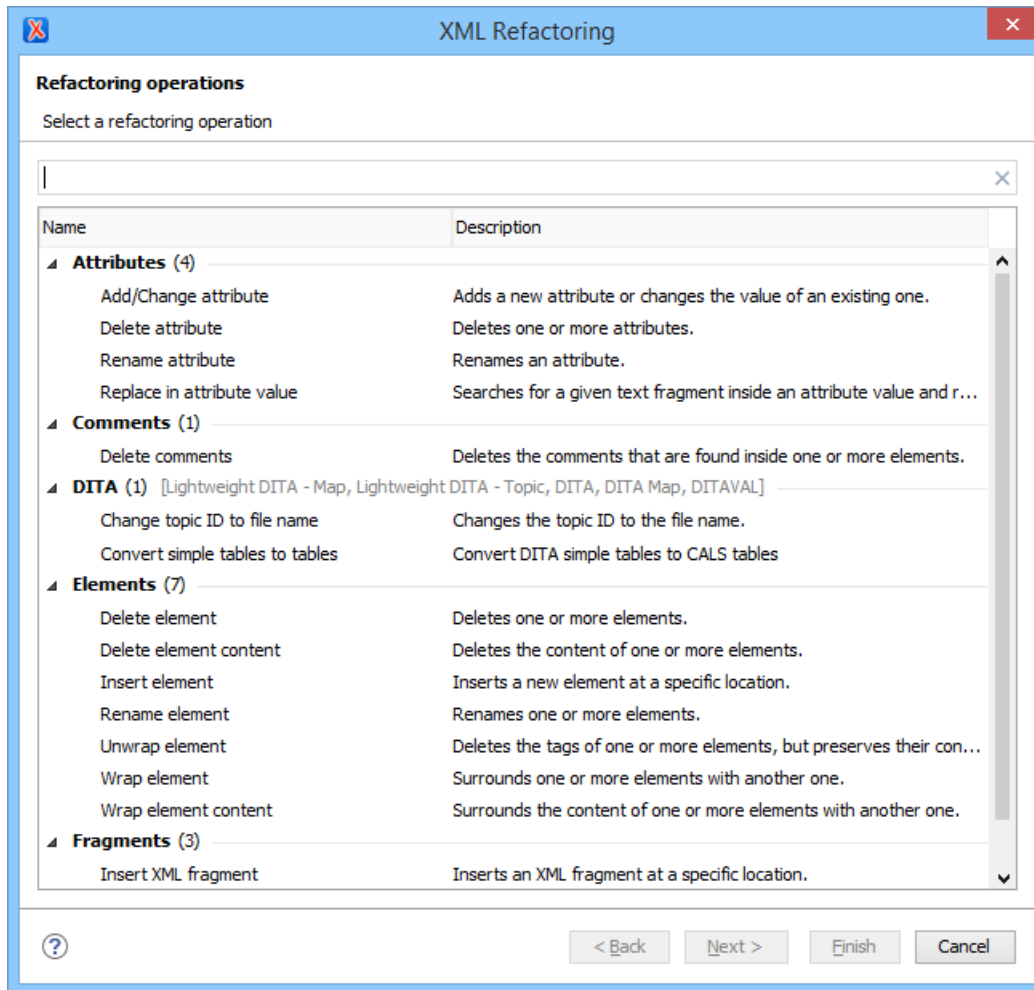


Figure 374: XML Refactoring Wizard

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.

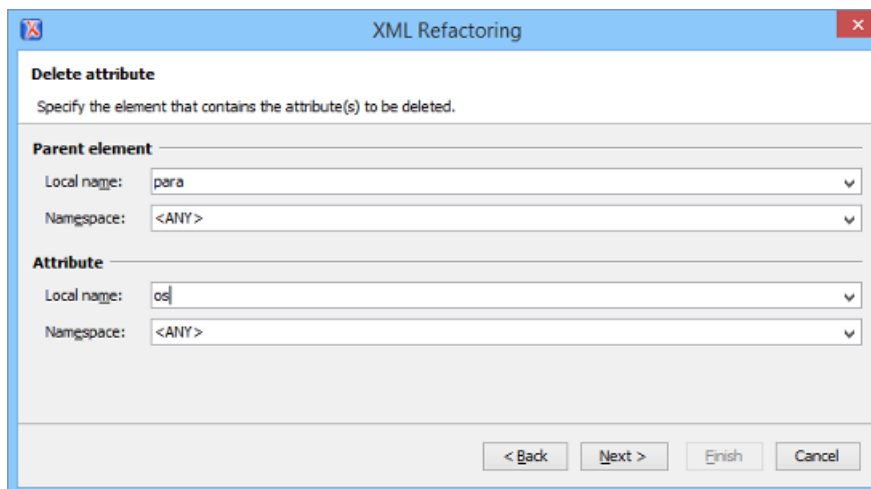


Figure 375: XML Refactoring 2nd Wizard Page (Delete Attribute Operation)

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.

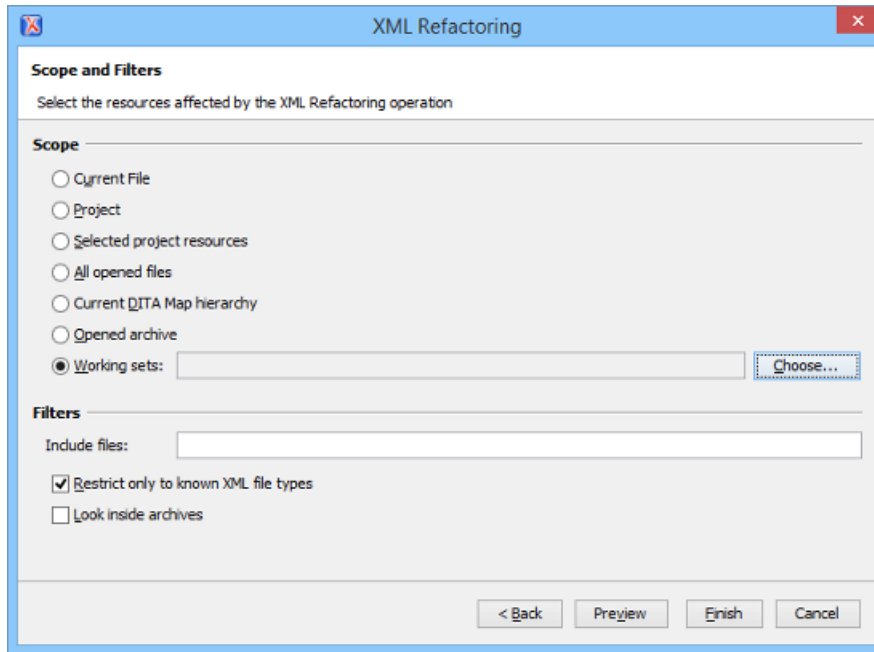


Figure 376: XML Refactoring - Scope and Filters Wizard Page

If an operation takes longer than expected you can use the  **Stop** button in the progress bar to cancel the operation.



Note: It is recommended that you use the **Preview** button to review all the changes that will be made by the refactoring operation before applying the changes.



Warning: After clicking the **Finish** button, the operation will be processed and Oxygen XML Developer plugin provides no automatic means for reverting the operations. Any **Undo** action will only revert changes on the current document.

Predefined Refactoring Operations

The XML Refactoring tool includes a variety of predefined operations that can be used for common refactoring tasks. They are grouped by category in the **Refactoring operations** wizard page. You can also access the operations from the **Refactoring** submenu in the contextual menu of **Text** mode. The operations are also grouped by category in this submenu. When selecting the operations from the contextual menu, Oxygen XML Developer 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.



Tip: Each operation includes a link in the lower part of the wizard that opens the **XML / XSLT-FO-XQuery / XPath** preferences page where you can configure XPath options and declare namespace prefixes.

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. This operation allows you to specify the following parameters:

- **Parent element** section
 - **Element** - 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.
- **Attribute** section
 - **Local name** - The local name of the affected attribute.
 - **Namespace** - The namespace of the affected attribute.
 - **Value** - The value for the affected attribute.
- **Options** section
 - You can choose between one of the following options for the **Operation mode**:
 - **Add the attribute in the parent elements where it is missing**
 - **Change the value in the parent elements where the attribute already exists**
 - **Both**

Delete attribute

Use this operation to remove one or more attributes. This operation requires you to specify the following parameters:

- **Element** - 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.
- **Attribute** - The name of the attribute to be deleted.

Rename attribute

Use this operation to rename an attribute. This operation requires you to specify the following parameters:

- **Element** - 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.
- **Attribute** - The name of the attribute to be renamed.
- **New local name** - The new local name of the attribute.

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. This operation allows you to specify the following parameters:

- **Target attribute** section
 - **Element** - 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.
 - **Attribute** - The name of the attribute to be modified.
- **Find / Replace** section
 - **Find** - The text fragments to find. You can use Perl-like regular expressions.
 - **Replace with** - The text fragment to replace the target with. This parameter can bind regular expression capturing groups (\$1, \$2, etc.) from the find pattern.

Refactoring Operations for *Comments*

Delete comments

Use this operation to delete comments from one or more elements. This operation requires you specify the following parameter:

- **Element** - The target element (or elements) for which comments will be deleted, in the form of a local name from any namespace, a local name with a namespace prefix, or an XPath expression.



Note: Comments that are outside the root element will not be deleted because the *serializer* preserves the content before and after the root.

Refactoring Operations for *DITA*

Change topic ID to file name

Use this operation to change the ID of a topic to be the same as its file name. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Convert simple tables to CALS tables

Use this operation to convert DITA simple tables to CALS tables. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Refactoring Operations for *Elements*

Delete element

Use this operation to delete elements. This operation requires you to specify the following parameter:

- **Element** - 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.

Delete element content

Use this operation to delete the content of elements. This operation requires you to specify the following parameter:

- **Element** - The target element whose 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.

Insert element

Use this operation to insert new elements. This operation allows you to specify the following parameters:

- **Element** section
 - **Local name** - The local name of the element to be inserted.
 - **Namespace** - The namespace of the element to be inserted.
- **Location** section
 - **XPath** - An XPath expression that identifies an existing element to which the new element is relative, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
 - **Position** - The position where the new element will be inserted, in relation to the specified existing element. The possible selections in the drop-down menu are: **After**, **Before**, **First child**, or **Last child**.

Rename element

Use this operation to rename elements. This operation requires you to specify the following parameters:

- **Target elements (XPath)** - The target elements to be renamed, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- **New local name** - The new local name of the element.

Unwrap element

Use this operation to remove the surrounding tags of elements, while keeping the content unchanged. This operation requires you to specify the following parameter:

- **Target elements (XPath)** - The target elements whose 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.

Wrap element

Use this operation to surround elements with element tags. This operation allows you to specify the following parameters:

- **Target elements (XPath)** - The target elements 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.
- **Wrapper element** section
 - **Local name** - The local name of the *Wrapper element*.
 - **Namespace** - The namespace of the *Wrapper element*.

Wrap element content

Use this operation to surround the content of elements with element tags. This operation allows you to specify the following parameters:

- **Target elements (XPath)** - The target elements whose content will 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.
- **Wrapper element** section
 - **Local name** - The local name of the *Wrapper element* that will surround the content of the target.
 - **Namespace** - The namespace of the *Wrapper element* that will surround the content of the target.

Refactoring Operations for *Fragments*

Insert XML fragment

Use this operation to insert an XML fragment. This operation allows you to specify the following:

- **XML Fragment** - The XML fragment to be inserted.
- **Location** section
 - **XPath** - An XPath expression that identifies an existing element to which the inserted fragment is relative, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
 - **Position** - The position where the fragment will be inserted, in relation to the specified existing element. The possible selections in the drop-down menu are: **After**, **Before**, **First child**, or **Last child**.

Replace element content with XML fragment

Use this operation to replace the content of elements with an XML fragment. This operation allows you to specify the following parameters:

- **Target elements (XPath)** - The target elements whose 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.
- **XML Fragment** - The XML fragment with which to replace the content of the target element.

Replace element with XML fragment

Use this operation to replace elements with an XML fragment. This operation allows you to specify the following parameters:

- **Target elements (XPath)** - The target elements to be replaced, in the form of a local name from any namespace, a local name with a namespace prefix, or other XPath expressions.
- **XML Fragment** - The XML fragment with which to replace the target element.

Refactoring Operations for JATSKit

Add BITS DOCTYPE - NLM/NCBI Book Interchange 2.0

Use this operation to add an NLM 'BITS' 2.0 DOCTYPE declaration. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Add Blue DOCTYPE - NISO JATS Publishing 1.1

Use this operation to add a JATS 'Blue' 1.1 DOCTYPE declaration. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Normalize IDs

Use this operation to normalize assigned IDs and assigned IDs to elements that are missing them.. This operation allows you to choose a scope for the operation and some filters:

- **Scope** - Select from a variety of options to define the scope for which resources will be affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, or just the **Current file**.
- **Filters** section
 - **Include files** - Specifies files to be excluded from the operation. You can specify multiple files by separating them with commas and the patterns can include wildcards (such as * or ?).
 - **Restrict to known XML file types only** - Excludes non-XML file types from the operation.

Additional Notes



Note: There are some operations that allow <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

While Oxygen XML Developer plugin includes a variety of predefined XML refactoring operations to help you accomplish particular tasks, you can also create custom operations according to your specific needs. For example, you could create a custom refactoring operation to convert an attribute to an element and insert the element as the first child of the parent element.

An XML Refactoring operation is defined as a pair of resources:

- An *XQuery Update script* or *XSLT stylesheet* that Oxygen XML Developer plugin will run to refactor the XML files.
- An *XML Operation Descriptor* file that contains information about the operation (such as the name, description, and parameters).

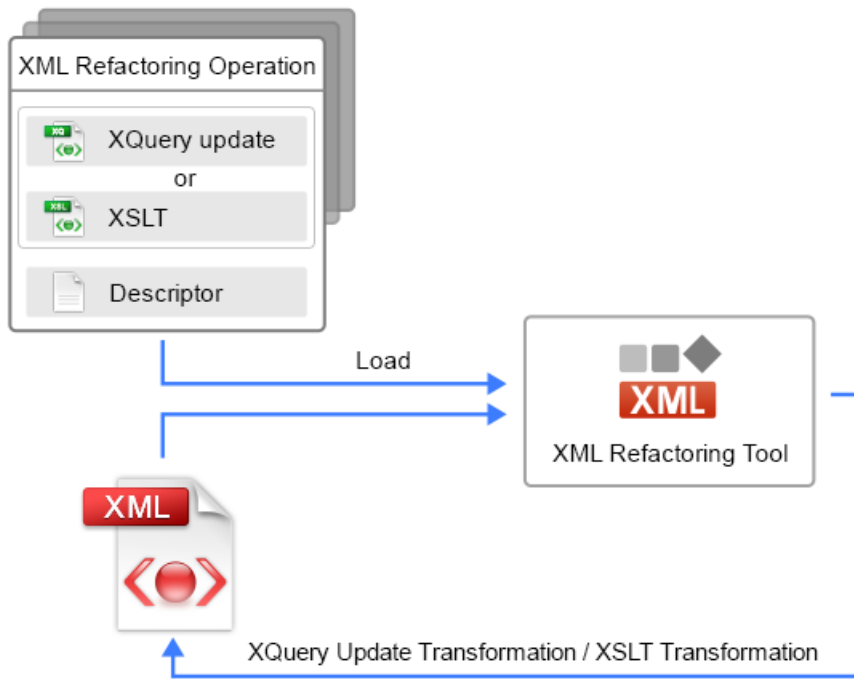


Figure 377: Diagram of an XML Refactoring Operation

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 Developer 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://www.oxygenxml.com/ns/xmlRefactoring/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 Developer plugin](#) scans when loading the custom operations.

Result: Once you run the **XML Refactoring** tool again, the custom operation appears in [the Refactoring Operations wizard page](#).

Related information

[Storing and Sharing Refactoring Operations](#) on page 242

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 Developer 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](#).

Related information

[Example of an XML Refactoring Operation](#) on page 239

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 Developer plugin will use a text input with corresponding content completion and syntax highlighting.



Note: The value of this parameter is transferred as plain text to the XQuery Update 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 Developer plugin will display a text area specialized for inserting XML documents.
- **NC_NAME** - The parameter for NC_NAME values. It is useful when you want to specify the local part of a *QName* for an element or attribute.
- **BOOLEAN** - Used to edit boolean parameters.
- **TEXT_CHOICE** - It is useful for parameters whose value should be from a list of possible values. Oxygen XML Developer plugin renders each possible value as a radio button 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 Developer 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>Local name of the parent element.</description>
  </localName>
  <namespace label="Namespace" name="element_namespace" allowsAny="true">
    <description>Local name of the 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>Namespace of the attribute to be converted.</description>
  </namespace>
</attributeParameter>
```



Note: All predefined operations are loaded from the `[OXYGEN_INSTALL_DIR]/refactoring` folder.

Related information

[Example of an XML Refactoring Operation](#) on page 239

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.

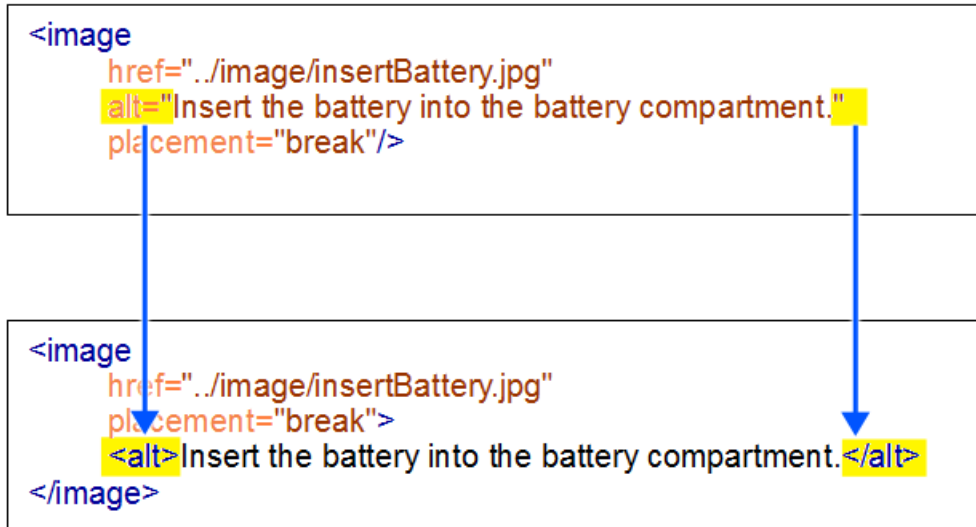


Figure 378: Example: Custom XML Refactoring Operation

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/attribute[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>Local name of the parent element.</description>
        </localName>
        <namespace label="Namespace" name="element_namespace" allowsAny="true">
          <description>Local name of the parent element</description>
        </namespace>
      </elementParameter>
    </section>
    <section label="Attribute">
      <attributeParameter dependsOn="elemID">
        <localName label="Name" name="attribute_localName">
          <description>Name of the attribute to be converted.</description>
        </localName>
        <namespace label="Namespace" name="attribute_namespace" allowsAny="true">
          <description>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 Developer plugin when it loads the custom operation*. When the XML Refactoring tool is started again, you will see the created operation.

Since various parameters can be specified, this custom operation can also be used for other similar tasks. The following image shows the parameters that can be specified in our example of the custom operation to convert an attribute to an element:

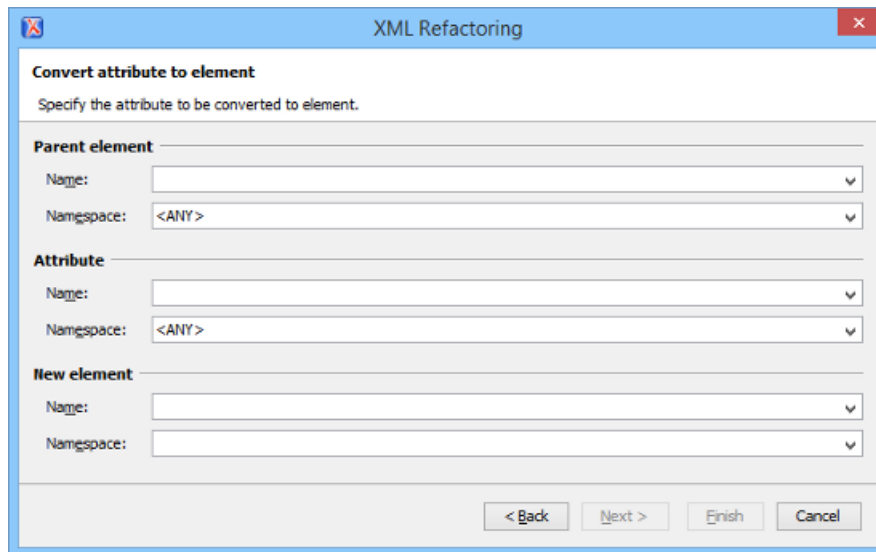


Figure 379: Example: XML Refactoring Wizard for a Custom Operation

Storing and Sharing Refactoring Operations

Oxygen XML Developer plugin scans the following locations when looking for XML Refactoring operations to provide flexibility:

- A refactoring folder, created inside a directory that is associated to a *framework* you are customizing.
- Any folder. In this case, you need to *open the Preferences dialog box*, go to **XML > XML Refactoring**, and specify the same folder in the *Load additional refactoring operations from text box*.
- The refactoring folder from the Oxygen XML Developer plugin installation directory (`[OXYGEN_INSTALL_DIR]/refactoring/`).

Sharing Custom Refactoring Operations

The purpose of Oxygen XML Developer plugin scanning multiple locations for the XML Refactoring operations is to provide more flexibility for developers who want to share the refactoring operations with the other team members. Depending on your particular use case, you can attach the custom refactoring operations to other resources, such as frameworks or projects.

After storing custom operations, you can share them with other users by sharing the resources.

Localizing XML Refactoring Operations

Oxygen XML Developer plugin includes localization support for the XML refactoring operations.

The translation keys for the built-in refactoring operations are located in `[OXYGEN_INSTALL_DIR]/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 possible values for each parameter.

Translated refactoring information uses the following form:

```
${i18n(translation_key)}
```

Oxygen XML Developer plugin scans the following locations to find the `translation.xml` files that are used to load the translation keys:

- A `refactoring/i18n` folder, created inside a directory that is associated to a customized *framework*.
- A `i18n` folder, created inside a directory that is associated to a customized *framework*.
- An `i18n` folder inside any specified folder. In this case, you need to [open the Preferences dialog box](#), go to **XML > XML Refactoring**, and specify the folder in the **Load additional refactoring operations from** text box.
- The `refactoring/i18n` folder from the Oxygen XML Developer plugin installation directory (`[OXYGEN_INSTALL_DIR]/refactoring/i18n`).

Example of a Refactoring Operation Descriptor File with *i18n* Support

```
<?xml version="1.0" encoding="UTF-8"?>
<refactoringOperationDescriptor
  xmlns="http://www.oxygenxml.com/ns/xmlRefactoring" 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>
```

Generating Sample XML Files

Oxygen XML Developer plugin offers support to generate sample XML files both from XML schema 1.0 and XML schema 1.1, depending on the XML schema version set in [XML Schema preferences page](#).

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 action opens the **Generate Sample XML Files** dialog box that allows you to configure a variety of options for generating the files.

For more information about this tool, watch our video demonstration about generating sample XML files at https://www.oxygenxml.com/demo/Generate_Sample_XML_Files.html.

The **Generate Sample XML Files** dialog box contains three tabs with various configurable options. Default values for these options can be set in the [Sample XML Files Generator preferences page](#).

Schema Tab (Generate Sample XML Files Tool)

The  **Generate Sample XML Files** tool includes a dialog box that allows you to configure a variety of options for generating the XML files. The first set of options are found in the **Schema** tab.

The screenshot shows the 'Generate Sample XML Files' dialog box with the 'Schema' tab selected. The 'W3C XML Schema' section contains the following fields:

- URL:** `http://www.w3.org/2004/10/inkml/inkml.xsd`
- Namespace:** `http://www.w3.org/2003/InkML`
- Root Element:** `ink`
- Output folder:** `D:\projects\userguide-private\DTA\topics\out`
- Filename prefix:** `instance`
- Extension:** `xml`
- Number of instances:** `1`
- ☒ **Open first instance in editor**

The **Namespaces** section includes a 'Default Namespace' dropdown set to `<NO_NAMESPACE>` and a table:

Prefix	Namespace
mml	<code>http://www.w3.org/1998/Math/MathML</code>
inkml	<code>http://www.w3.org/2003/InkML</code>

At the bottom are buttons for '?', 'Export settings', 'Load settings', 'OK', and 'Cancel'.

Figure 380: Generate Sample XML Files Dialog Box (Schema Tab)

This tab includes the following options:

URL

Specifies the URL of the Schema location. You can specify the path by using the text field, the history drop-down menu, or the browsing tools in the ▾ **Browse** drop-down list.

Namespace

Displays the namespace of the selected schema.

Root Element

After the schema is selected, this drop-down menu 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

You can specify the prefix and extension for the file name that will be generated. Generated file names have the following format: `prefixN.extension`, where N represents an incremental number from 0 up to the specified **Number of instances**.

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 the editor.

Namespaces section

You can specify the **Default Namespace**, as well as the prefixes for the namespaces.

Load settings

Use this button to load previously exported settings.

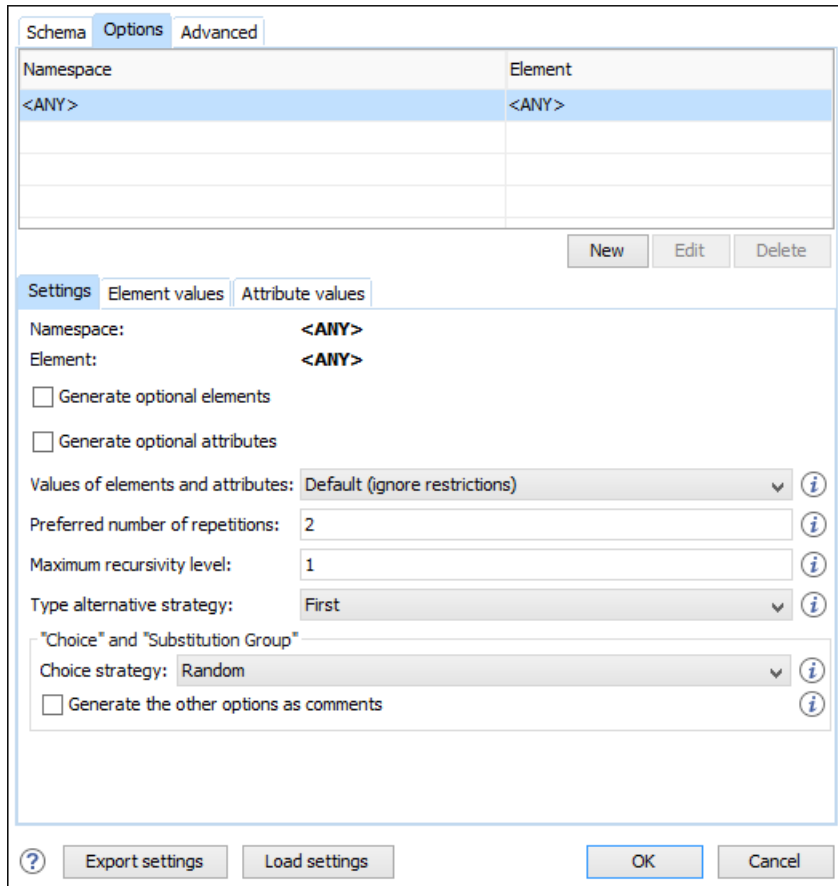
Export settings

Use this button to save the current settings for future use.

You can click **OK** at any point to generate the sample XML files.

Options Tab (Generate Sample XML Files Tool)

The  **Generate Sample XML Files** tool includes a dialog box that allows you to configure a variety of options for generating the XML files. The **Options** tab allows you to set specific options for namespaces and elements.



Namespace	Element
<ANY>	<ANY>

New Edit Delete

Settings Element values Attribute values

Namespace: <ANY>

Element: <ANY>

☐ Generate optional elements

☐ Generate optional attributes

Values of elements and attributes: Default (ignore restrictions) i

Preferred number of repetitions: 2 i

Maximum recursivity level: 1 i

Type alternative strategy: First i

"Choice" and "Substitution Group" i

Choice strategy: Random i

☐ Generate the other options as comments i

? Export settings Load settings OK Cancel

Figure 381: Generate Sample XML Files Dialog Box (Options Tab)

This tab includes the following options:

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.
- All elements from a specific namespace.
- A specific element from a specific namespace.

Settings subtab

Namespace

Displays the namespace specified in the table at the top of the dialog box.

Element

Displays the element specified in the table at the top of the dialog box.

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. The following 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 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.

Preferred number of repetitions

Allows you to set the preferred number of repeating elements related to **minOccurs** and **maxOccurs** facets defined in the 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.

Type alternative strategy

Used for the **xs:alternative** element from XML Schema 1.1. The possible strategies are:

- **First** - The first valid alternative type is always used.
- **Random** - A random alternative type is used.

Choice strategy

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

If enabled, generates 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.

Element values subtab

Allows you to add values that are used to generate the content of elements. If there are multiple values, then the values are used in a random order.

Attribute values subtab

Allows you to add values that are used to generate the content of attributes. If there are multiple values, then the values are used in a random order.

Load settings

Use this button to load previously exported settings.

Export settings

Use this button to save the current settings for future use.

You can click **OK** at any point to generate the sample XML files.

Advanced Tab (Generate Sample XML Files Tool)

The  **Generate Sample XML Files** tool includes a dialog box that allows you to configure a variety of options for generating the XML files. The **Advanced** tab allows you to set some options in regards to output values and performance.

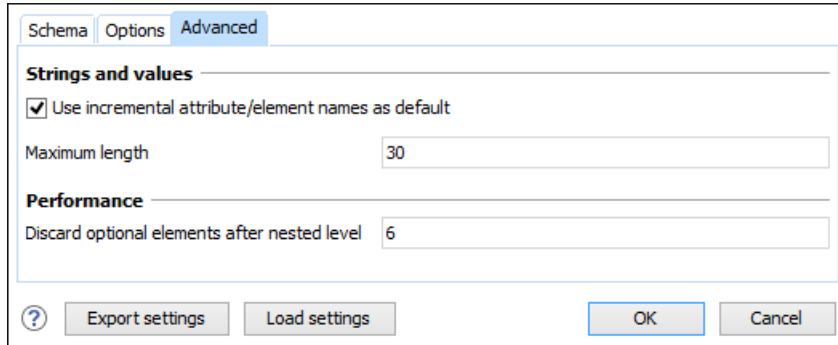


Figure 382: Generate Sample XML Files Dialog Box (Advanced Tab)

This tab includes the following options:

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.

Converting Schema to Another Schema Language

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

To use this tool, select the  **Generate/Convert Schema (Ctrl + Shift + BackSlash (Command + Shift + BackSlash on OS X))** action from the **XML Tools** menu. This action opens the **Generate/Convert Schema** dialog box that allows you to configure various options for conversion.

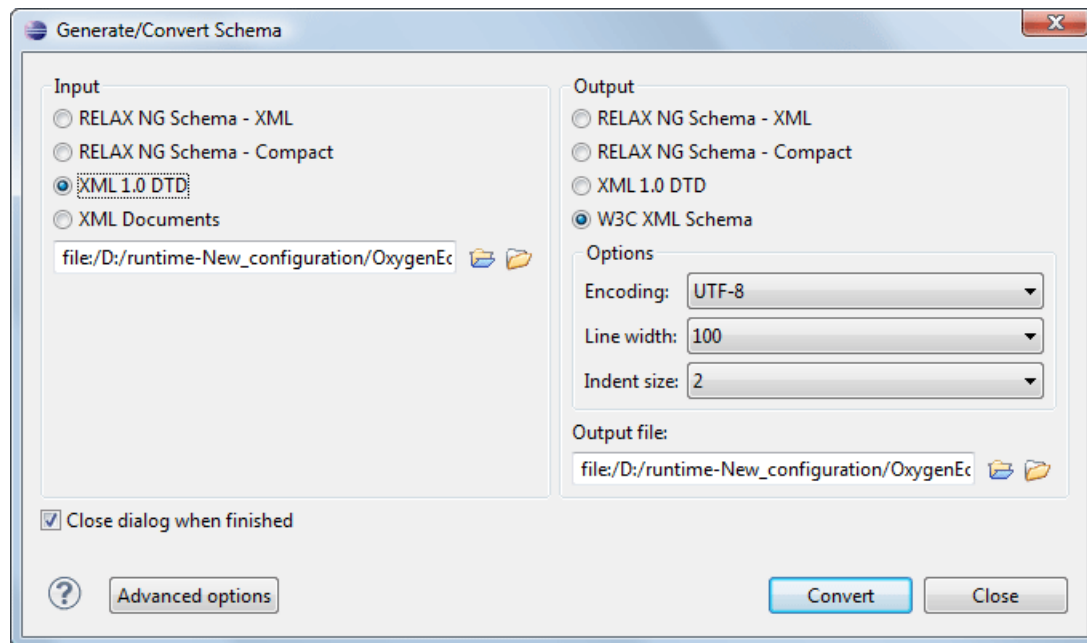


Figure 383: Generate/Convert Schema Dialog Box

The **Generate/Convert Schema** dialog box includes the following options:

Input section

Allows you to select the language of the source schema. If the conversion is based on a set of XML files, rather than just a single XML file, select the **XML Documents** option and use the file selector to add the XML files involved in the conversion.

Output section

Allows you to select the language of the target schema.

Options

You can choose the **Encoding**, the maximum **Line width**, and the **Indent size** (in number of spaces) for one level of indentation.

Output file

Specifies the path for the output file that will be generated.

Close dialog when finished

If you deselect this option, the dialog box will remain opened after the conversion so that you can easily continue to convert more files.

Advanced options

If you select **XML 1.0 DTD** for the input, you can click this button to access more advance options to further fine-tune the conversion. The following advanced options are available:

XML 1.0 DTD Input section

These options apply to the source DTD:

- **xmlns** - Specifies the default namespace, that is the namespace used for unqualified element names.
- **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.
- **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.
- **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.

- **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.
- **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.
- **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.
- **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 or not 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.
- **xmlns mappings table** - Each row specifies the prefix used for a namespace in the input schema.

W3C XML Schema Output section

This section is available if you select **W3C XML Schema** for the output.

- **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).

Converting Database to XML Schema

Oxygen XML Developer 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.

Result: 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**.

Result: 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**.

Result: The database structure is converted to an XML Schema and it is opened for viewing and editing.

XML to JSON Converter

Oxygen XML Developer plugin includes a useful and simple tool for converting XML files to JSON. It can be found in the **XML Tools** menu.

To convert an XML document to JSON, follow these steps:

1. Select the **XML to JSON** action from the **XML Tools** menu.
The **XML to JSON** dialog box is displayed:

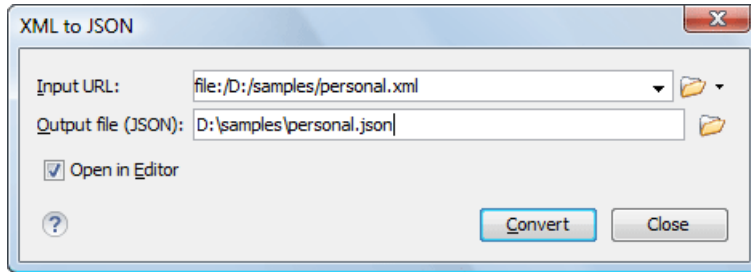


Figure 384: XML to JSON Dialog Box

2. Choose or enter the **Input URL** of the XML document.
3. Choose the path of the **Output file** that will contain the conversion JSON result.
4. Check the **Open in Editor** option to open the JSON result of the conversion in the Oxygen XML Developer plugin JSON Editor.
5. Click the **Convert** button.

The original XML document is now converted to a JSON document.

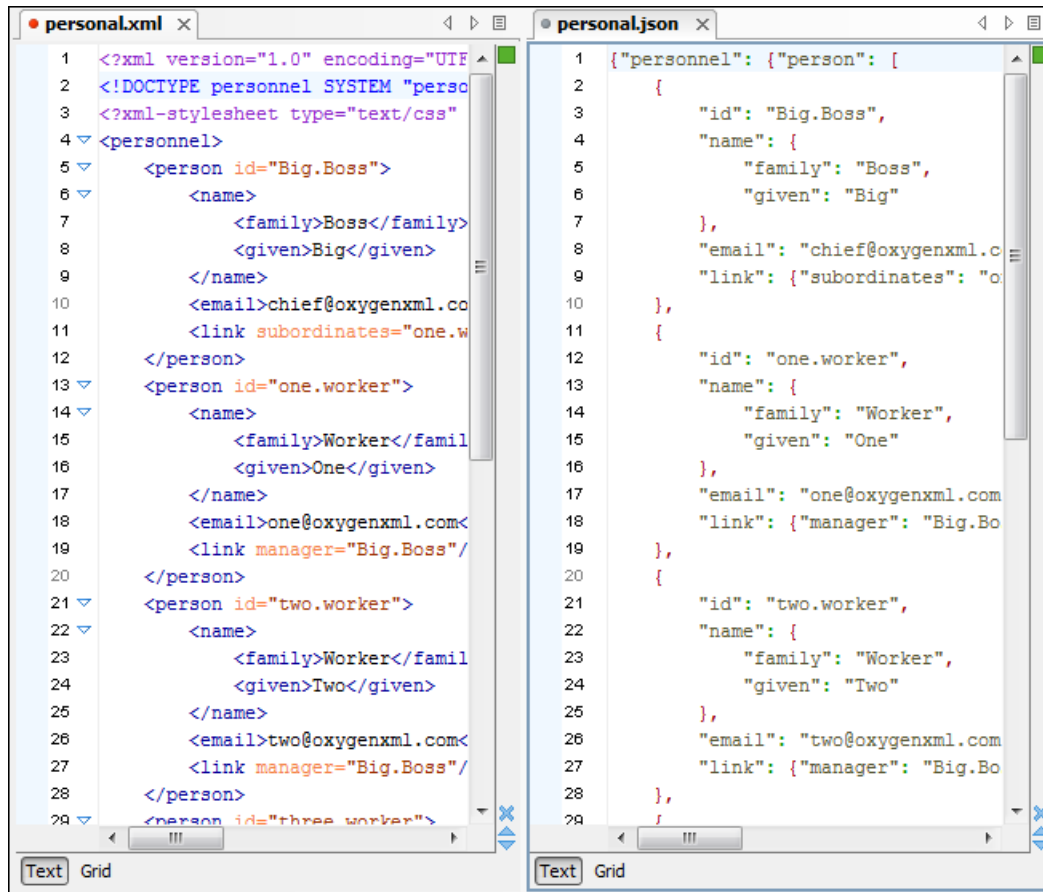


Figure 385: Example: XML to JSON Operation Result

Generate Documentation

Oxygen XML Developer plugin includes a tool for generating documentation for XSLT, XML Schema, XQuery, and WSDL documents.

Generating Documentation for an XML Schema

Oxygen XML Developer 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.

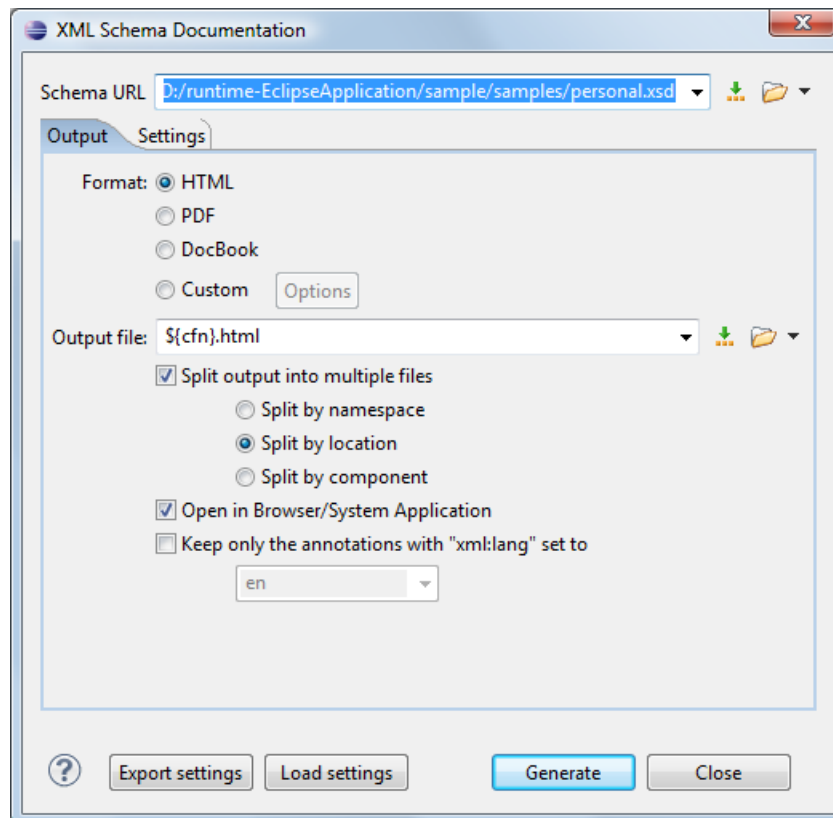


Figure 386: XML Schema Documentation Dialog Box

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.

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, go to **Window > Preferences > General > Web Browser** and specify it there. 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.).

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.

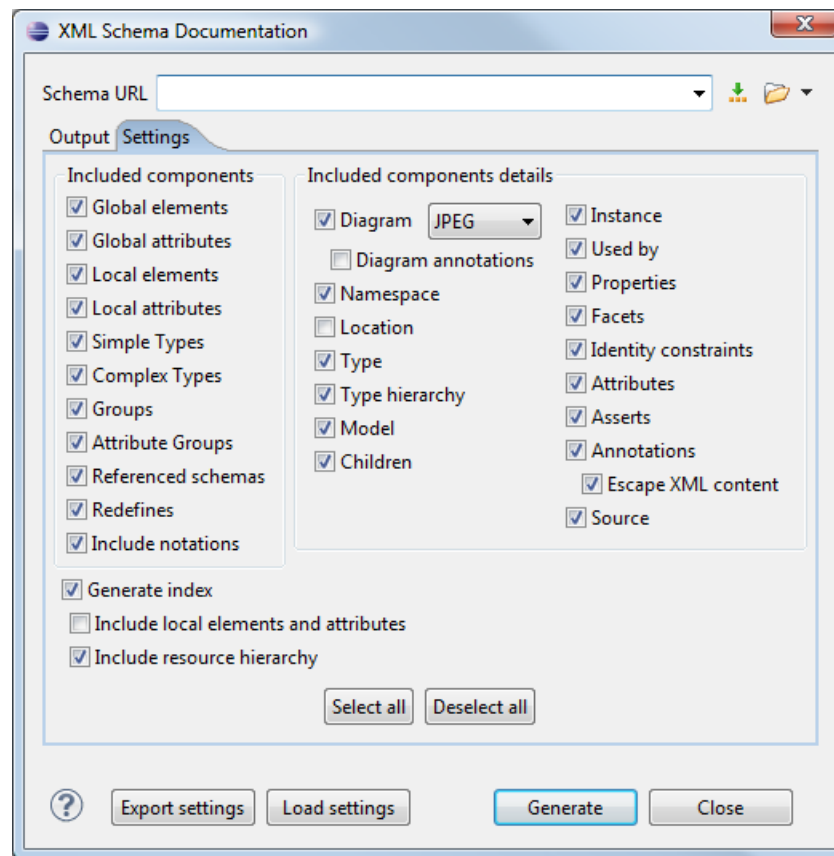


Figure 387: Settings Tab of the XML Schema Documentation Dialog Box

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 or not 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. The separator characters that are used depend upon 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 or not the resource hierarchy for an XML Schema documentation is generated. It is disabled by default.

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.

Related information

[Customizing the PDF Output of Generated XML Schema Documentation](#) on page 325

Generating Documentation for an XSLT Stylesheet

You can use Oxygen XML Developer plugin to generate detailed documentation in HTML format for the elements (top-level elements whose names are in the XSLT namespace) of an XSLT stylesheet. You can 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.

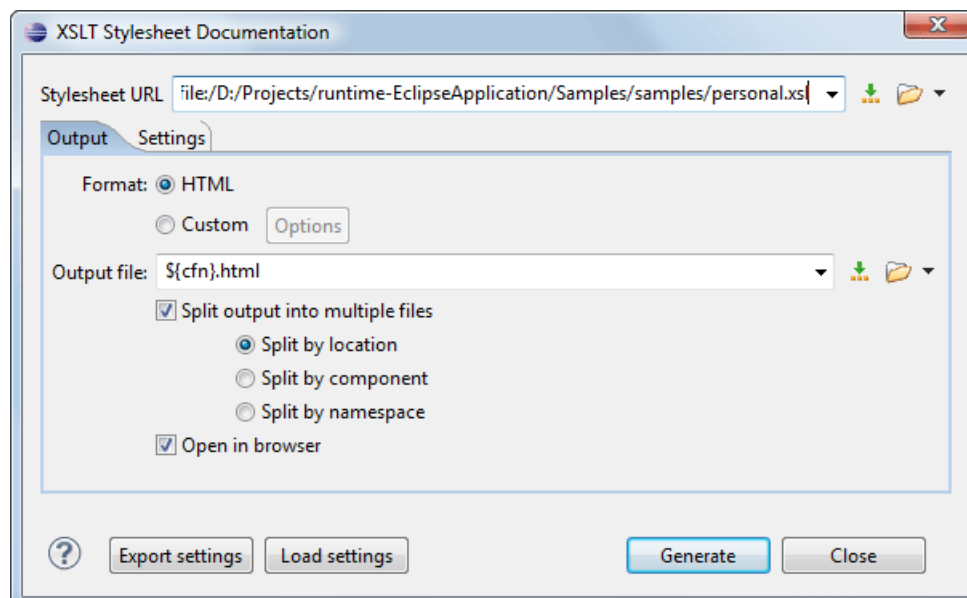


Figure 388: XSLT Stylesheet Documentation Dialog Box

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.

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, choosing another split criterion may generate smaller output files, providing 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, go to **Window > Preferences > General > Web Browser** and specify it there. This will take precedence over the default system application settings.

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.

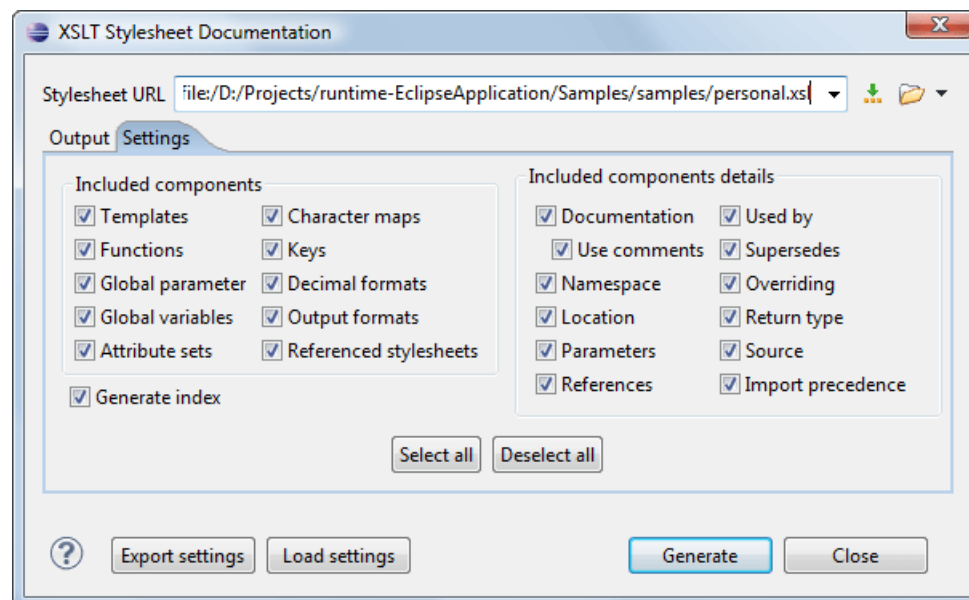


Figure 389: Settings Tab of the XSLT Stylesheet Documentation Dialog Box

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

Other XSLT documentation blocks that are not recognized will just be serialized inside an HTML `pre` element. You can change this behavior by using a *custom format* instead of the built-in *HTML format* and providing your own XSLT stylesheets.
 - **Use comments** - Controls whether or not the comments that precede an XSLT element is treated as documentation for the element they precede. Comments that precede or succeed the `xml: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.

Related information

[XSLT Stylesheet Component Documentation Support](#) on page 260

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.

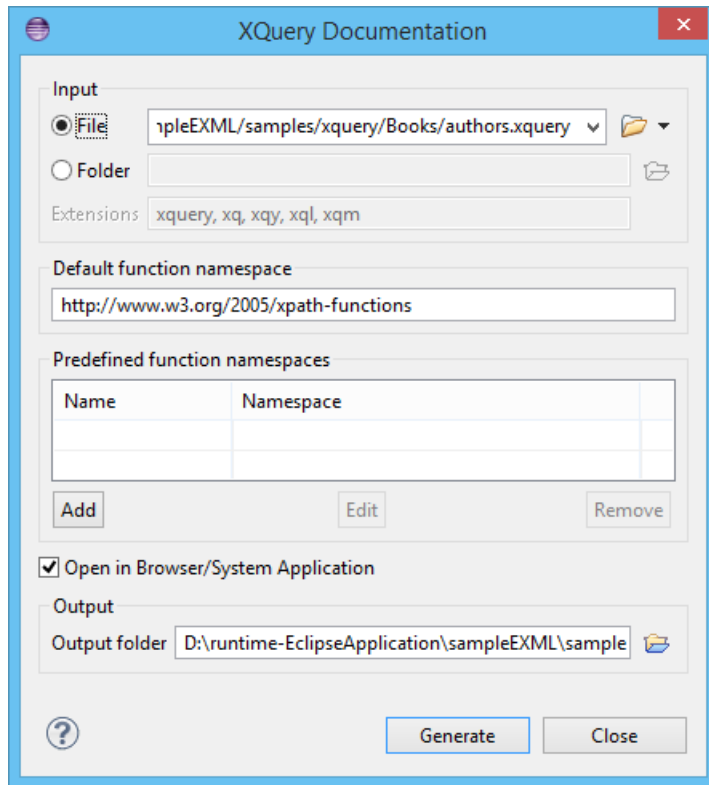


Figure 390: XQuery Documentation Dialog Box

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, go to **Window > Preferences > General > Web Browser** and specify it there. This will take precedence over the default system application settings.

- **Output** - Allows you to specify where the generated documentation is saved on disk.

Generating Documentation for WSDL Documents

You can use Oxygen XML Developer plugin to generate detailed documentation for the components of a WSDL document in HTML format. You can select the WSDL components to include in your output and the level of details to present for each of them. Also, the components are hyperlinked. 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.

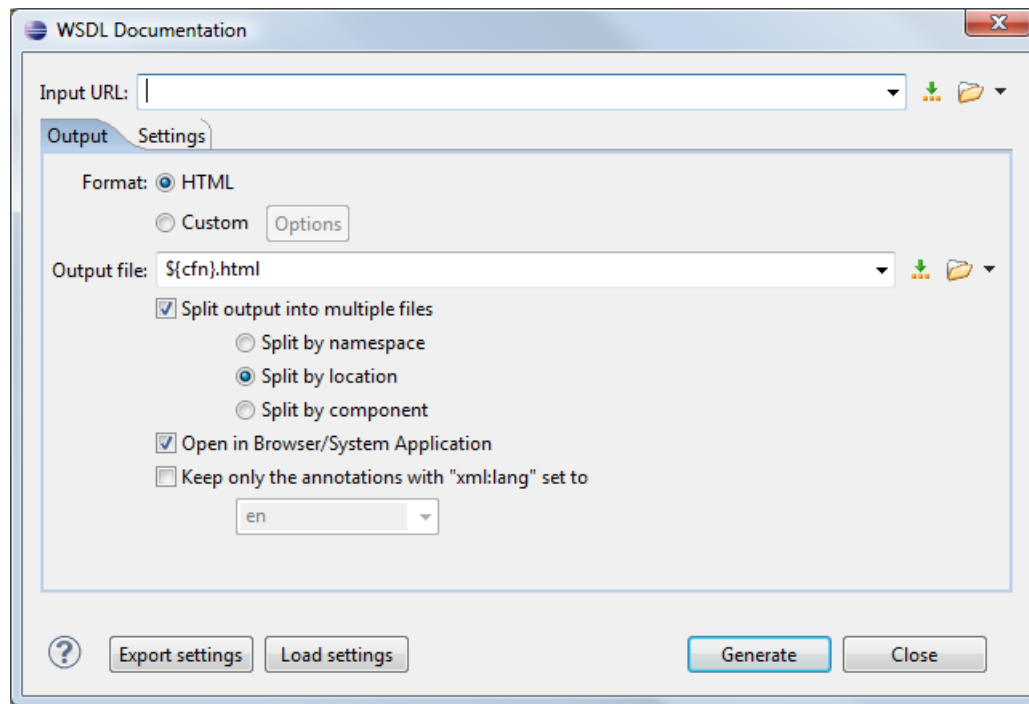


Figure 391: WSDL Documentation Dialog Box

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.

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, go to **Window > Preferences > General > Web Browser** and specify it there. 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.).

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.

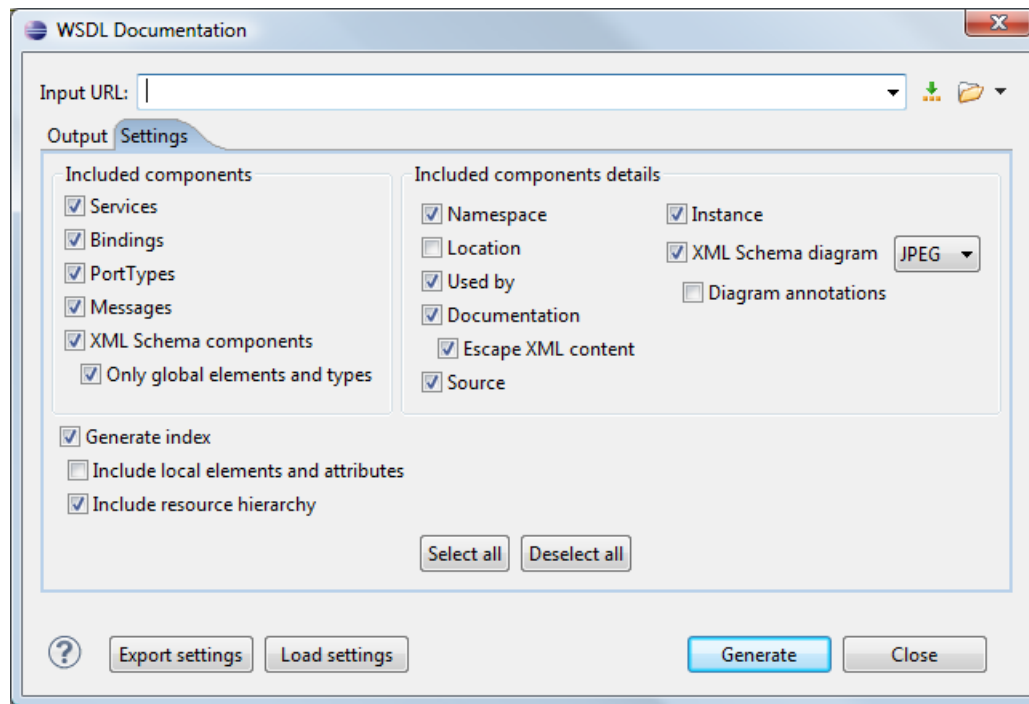


Figure 392: Settings Tab of the WSDL Documentation Dialog Box

The **Settings** tab allows you to choose whether or not to include the following:

- **Components**
 - **Services** - Specifies whether or not the generated documentation includes the WSDL services.
 - **Bindings** - Specifies whether or not the generated documentation includes the WSDL bindings.
 - **Port Types** - Specifies whether or not the generated documentation includes the WSDL port types.
 - **Messages** - Specifies whether or not the generated documentation includes the WSDL messages.
 - **XML Schema Components** - Specifies whether or not the generated documentation includes the XML Schema components.
 - **Only global elements and types** - Specifies whether or not 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 or not 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 or not the resource hierarchy for an XML Schema documentation is generated. It is disabled by default.

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.

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.

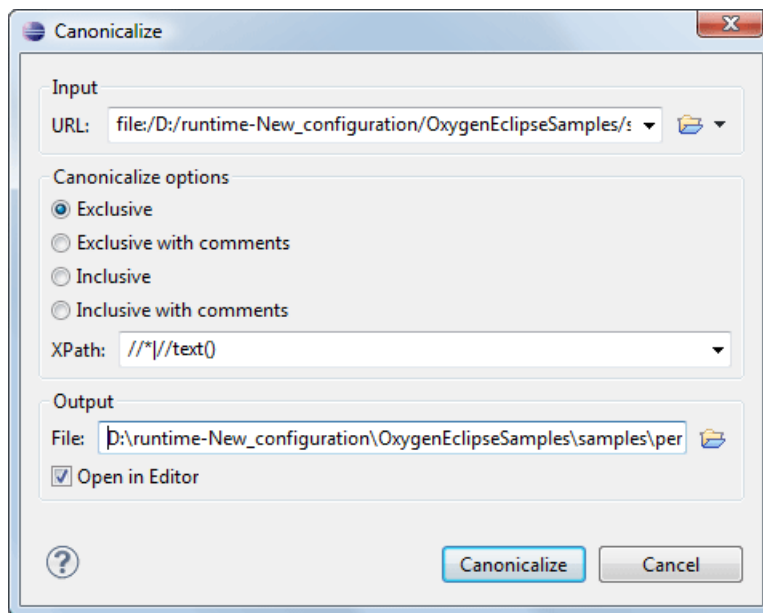


Figure 393: Canonicalization Settings Dialog Box

The **Canonicalize** dialog box allows you to set the following options:

- **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 various 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 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.

Related information

[Digital Signatures Overview](#) on page 425

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.

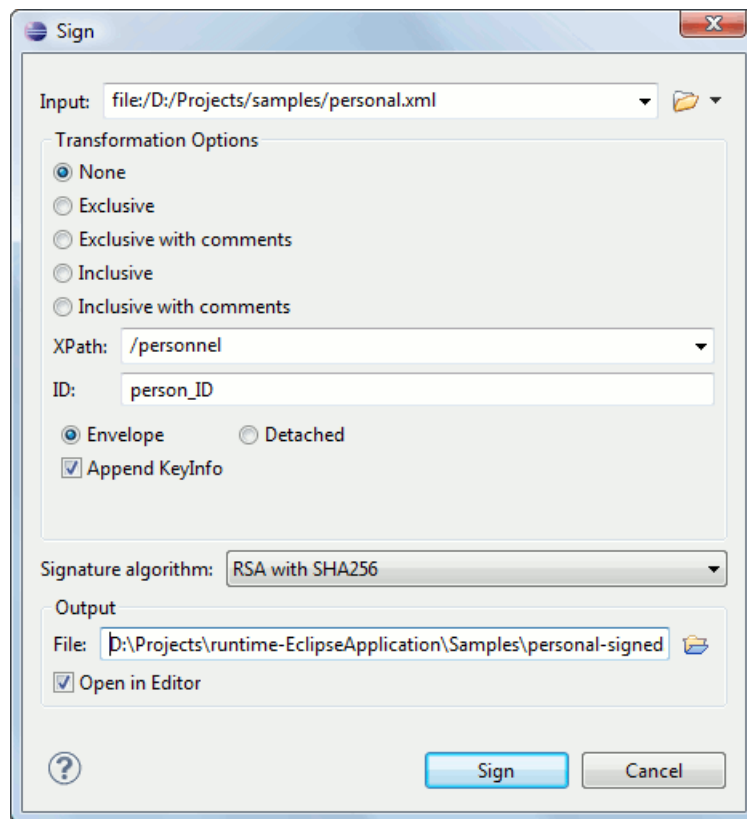


Figure 394: Signature Settings Dialog Box

The following options are available:

- **Note:** If Oxygen XML Developer 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.

Could not obtain a valid certificate. You must 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 various 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 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 Developer plugin.

Related tasks

[Example of How to Digitally Sign XML Files or Content](#) on page 430

Related information

[Digital Signatures Overview](#) on page 425

[Verifying Signature](#) on page 430

Verifying 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.

Related tasks

[Example of How to Digitally Sign XML Files or Content](#) on page 430

Related information

[Digital Signatures Overview](#) on page 425

[Signing Files](#) on page 428

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 by using the integrated **WSDL SOAP Analyzer** tool (available from the toolbar or **WSDL** menu).

Composing a SOAP Request

WSDL SOAP Analyzer is a tool that helps you test if the messages defined in a Web Service Descriptor (WSDL) are accepted by a Web Services server.

Oxygen XML Developer plugin provides two ways of testing, one for the currently edited WSDL document and another for the remote WSDL documents that are published on a web server. To open the **WSDL SOAP 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.

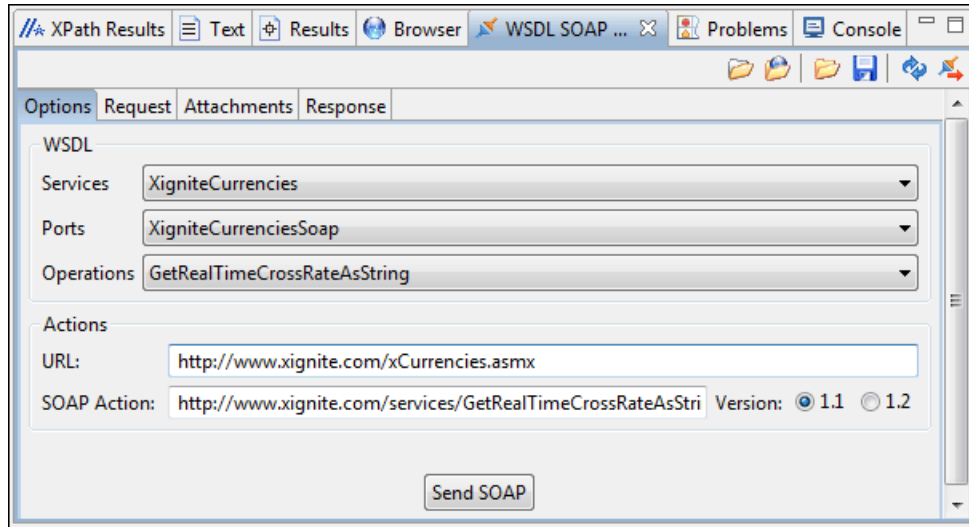


Figure 395: WSDL SOAP Analyzer View

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 Developer plugin tries to generate as much content as possible for the SOAP request. The envelope of the SOAP request has the correct namespace for the selected SOAP version, that is *http://schemas.xmlsoap.org/soap/envelope/* for SOAP 1.1 or *http://www.w3.org/2003/05/soap-envelope* for SOAP 1.2. Usually you just have to change a few values 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 various operations, Oxygen XML Developer plugin remembers the modified request for each one. You can press the **Regenerate** button 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 Developer plugin prompts you to save them, then tries to open them with the associated system application.
- **Errors List** - There may be situations where the WSDL file is respecting the WSDL XML Schema, but it fails to be valid (for example, in the case of a message that is defined by means of an element that is not found in the types section of the WSDL). In such a case, the errors are listed here. This list is presented only when there are errors.
- **Send Button** - Executes the request. A status dialog box is displayed when Oxygen XML Developer plugin is connecting to the server.

The testing of a WSDL file is straight-forward: click the WSDL analysis button, then select the service, the port, and the operation. The editor generates the skeleton for the SOAP request. You can edit the request, eventually attach files to it and

send it to the server. Watch the server response in the response area. You can find more details in the [Testing Remote WSDL Files](#) section.



Note: SOAP requests and responses are automatically validated in the **WSDL SOAP 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 Developer 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.

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.

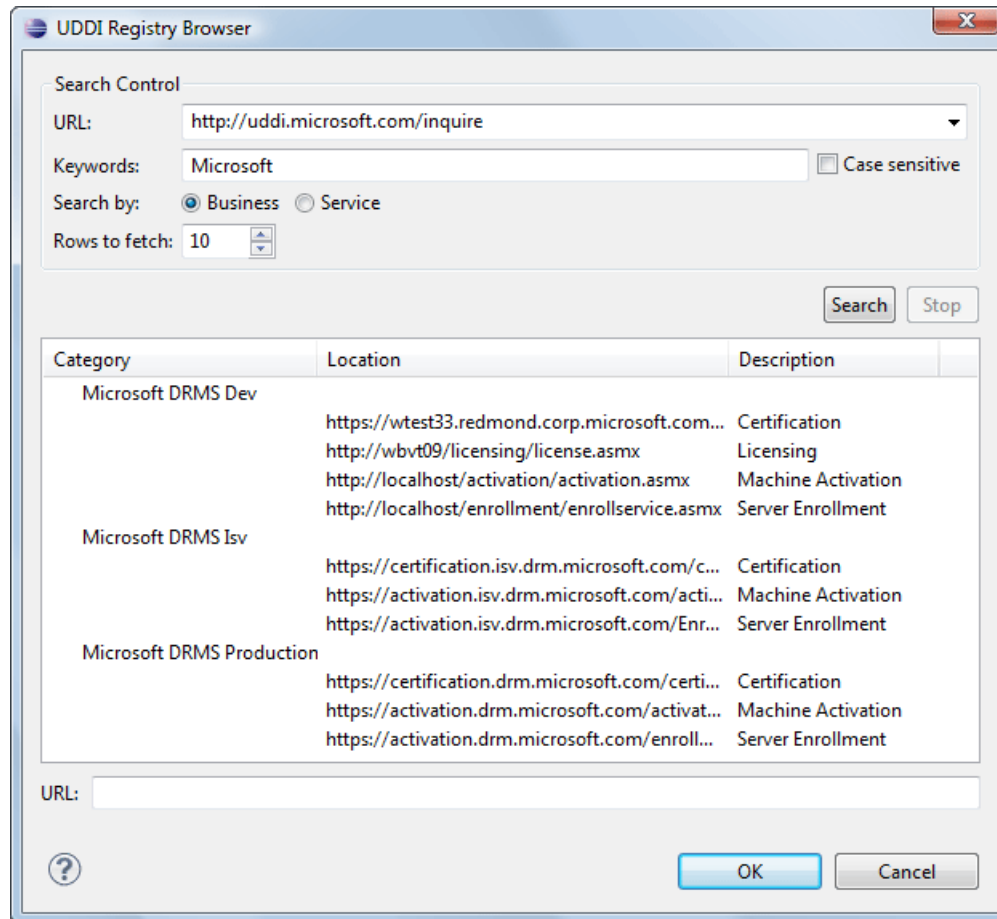


Figure 396: UDDI Registry Browser Dialog Box

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.

XML Schema Regular Expressions Builder

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

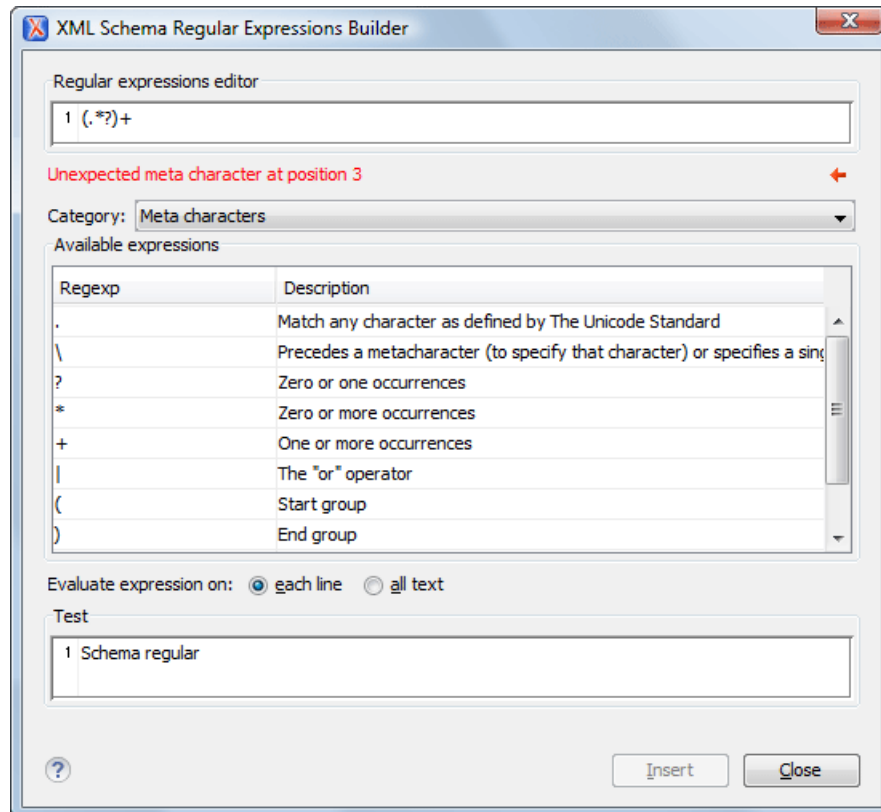


Figure 397: XML Schema Regular Expressions Builder Dialog Box

The dialog box contains the following:

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

You can choose from several categories of predefined expressions. The selected category influences the displayed expressions in the **Available expressions** table.

Available expressions

This table includes the available regular expressions and a short description for each of them. The set of expressions depends on the category selected in the previous **Category** 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.

Evaluate expression on

You can choose between 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

A text editor that allows you to enter a text sample for 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 mode* or *schema Design mode*. Accordingly, the **Insert** button will be disabled if the current document is edited in these modes.



Note: Some regular expressions may indefinitely block 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.

Chapter 17

Common Problems

Topics:

- [Performance Problems and Solutions](#)
- [Misc Problems and Solutions](#)

This section provides a collection of common performance and other types of problems that might be encountered when using Oxygen XML Developer plugin, along with their possible solutions.

Performance Problems and Solutions

This section contains solutions for some common performance problems that may appear when running Oxygen XML Developer plugin.

Out of Memory on External Processes

Problem

Oxygen XML Developer plugin throws an *Out Of Memory* error when trying to generate PDF output with the built-in Apache FOP processor.

Solution

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.

Related information

[FO Processors Preferences](#) on page 85

[Custom Engines Preferences](#) on page 83

Performance Issues with Large Documents

Problem

The performance of the application slows down considerably over time when editing large documents.

Solution

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 Developer 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.



Note:

Misc Problems and Solutions

This chapter presents common problems that may appear when running the application along with solutions for these problems.

'Address Family Not Supported by Protocol Family; Connect' Error

Problem

I have experienced the following error: *"Address Family Not Supported by Protocol Family; Connect"*. How do I solve it?

Solution

This seems to be an IPv6 connectivity problem. By default, the Java runtime used by Oxygen XML Developer plugin prefers to create connections via IPv6, if the support is available. However, even though it is available in appearance, IPv6 sometimes happens to be configured incorrectly on some systems.

A quick fix for this problem is to set the `java.net.preferIPv4Stack` Java property to `true` (`java.net.preferIPv4Stack=true`), by following this procedure:

1. Create a file named `custom_commons.vmoptions` and add on a single line `-Djava.net.preferIPv4Stack=true`. Then save the file and copy it to the Oxygen XML Developer plugin installation folder (may need admin access).
2. Restart Oxygen XML Developer plugin.
3. Make sure the procedure was successful by going to **Help > About > System properties** and check that the value of the `java.net.preferIPv4Stack` property is `true`.

Alignment Issues of the Main Menu on Linux Systems Based on Gnome 3.x

Problem

On some Linux systems based on Gnome 3.x (Ubuntu 11.x, 12.x), the main menu of Oxygen XML Developer plugin has alignment issues when you navigate it using your mouse.

Solutions

This is a known problem caused by Java SE 6 1.6.0_32 and earlier. You can resolve this problem using the latest Java SE 6 JRE from Oracle. To download the latest version, go to <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

To bypass the JRE bundled with Oxygen XML Developer plugin, go to the installation directory of Oxygen XML Developer plugin and rename or move the `jre` folder. If Oxygen XML Developer plugin does not seem to locate the system JRE, either set the `JAVA_HOME` environment variable to point to the location where you have installed the JRE, or you can simply copy that folder with the JRE to the installation directory and rename it to `jre` to take the place of the bundled JRE.

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.

Damaged File Associations on OS X

Problem

After upgrading OS X and Oxygen XML Developer plugin, it is no longer associated to the appropriate file types (such as XML, XSL, XSD, etc.) How can I create the file associations again?

Solution

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 Developer plugin installations on your hard drive.
2. Delete them by dragging them to the Trash.
3. Clear the Trash.
4. Unpack the Oxygen XML Developer plugin installation kit on your desktop.
5. Copy the contents of the archive into the folder `/ Applications / Oxygen`.
6. Run the following command in a Terminal:

```
/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 an XML or XSD file on your desktop. It should have the Oxygen XML Developer plugin icon.
9. Double-click the file.
10. Accept the confirmation.

Result: When you start Oxygen XML Developer plugin, the file associations should work correctly.

Details to Submit in a Request for Technical Support Using the Online Form

Problem

What details should I add to my request for technical support on the online form in the product website?

Solution

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.

If the text content of an XML document you want to send to the support team contains sensitive or private information, you can use the [Randomize XML text content action](#) to create filler content. Before using this action, you need to copy the initial XML resources and save them in a separate folder. Otherwise, you might lose your original information.

To generate an Oxygen XML Developer 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.

DITA Map Transformation Fails (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.

DITA PDF Transformation Fails

To generate the PDF output, Oxygen XML Developer plugin uses the DITA Open Toolkit.

You can analyze 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.fop` parameter to **yes**. Run the transformation, go to the temporary directory of the transformation, open the `topic.fop` 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 that 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.

DITA to CHM Transformation Fails

Oxygen XML Developer plugin uses the DITA Open Toolkit and the HTML Help compiler (part of the Microsoft HTML Help Workshop) to transform DITA content into *Compiled HTML Help* (or *CHM* in short).

However, the execution of the transformation scenario may still fail. Reported errors include:

- [exec] HHC5010: Error: Cannot open "fileName.chm". Compilation stopped. - This error occurs when the CHM output file is opened and the transformation scenario cannot rewrite its content. To solve this issue, close the CHM help file and 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.

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.

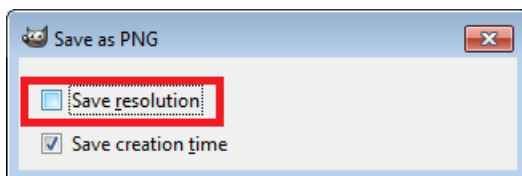
PDF output from XML content is obtained by first obtaining a intermediary XML format called XSL-FO and then applying an XSL-FO processor to it to obtain the PDF. This stretching problem is caused by the fact that all XSL-FO processors take into account the DPI (dots-per-inch) resolution when computing the size of the rendered image.

The PDF processor that 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 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 that output PDF is located in the `[OXYGEN_INSTALL_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 the same as 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:** It is best to save your images without any DPI resolution information. For example, when saving a PNG image in the open-source GIMP image editor, you do not want to save the resolution.



This allows you to control the image resolution from the configuration file for all referenced images.

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.

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.

Oxygen XML Developer plugin Takes Several Minutes to Start

Problem: Some anti-virus software can cause Java applications, such as Oxygen XML Developer plugin, to start very slowly due to scanning of compressed archives (such as the JAR libraries that all Java applications use).

Solution: The solution is to add the Oxygen XML Developer plugin folder to the list of exceptions in the anti-virus software settings.

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 this:

```
[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 this:

```
[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:

- Edit the transformation scenario and set the parameter *clean.temp* to *no*.
- Run the transformation, open the *topic.fo* file in Oxygen XML Developer plugin, and look in it at the line specified in the error message (See position 179:-1).
- Look around that line in the XSL - FO file to find relevant text content that 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 this:

```
[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 Developer plugin, and search in it for the `unique_4_Connect_42_wrongID` id.
5. Look around that line in the XSL - FO file to find relevant text content that 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.

Signature Verification Failed Error on a Resource from Documentum

Problem

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.

Solution

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
```

SVG 1.2 Rendering Issues

Oxygen XML Developer plugin uses the *Apache Batik* open source library to render SVG images and it only has partial support for SVG 1.2. For more information, see <http://xmlgraphics.apache.org/batik/dev/svg12.html>.

This partial support could lead to some rendering issues in Oxygen XML Developer plugin. For example, if you are using the *Inkscape* SVG editor, it is possible for it to save the SVG as 1.1, while using SVG 1.2 elements (such as `flowRoot`) inside it. This means that the image will not be properly rendered inside the application.

Syntax Highlight Not Available in Eclipse Plugin

Problem

I associated the `.ext` extension with Oxygen XML Developer plugin in Eclipse. Why does an `.ext` file opened with the Oxygen XML Developer plugin not have syntax highlight?

Solution

Associating an extension with Oxygen XML Developer plugin in Eclipse versions 3.6-3.8, 4.2-4.6 requires three steps:

1. Associate the `.ext` extension with the Oxygen XML Developer 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 Developer plugin to the list of **Associated editors** and make it the default editor..
2. Associate the .ext extension with the **Oxygen XML** content type.
 - a. *Open the **Preferences dialog box*** and go to **General > Content Types**.
 - b. Add *.ext to the **File associations** list for the **Text > XML > Oxygen XML Developer plugin** content type.
3. Press the **OK** button in the Eclipse preferences dialog box.

Result: Now when an *.ext file is opened the icon of the editor and the syntax highlight should be the same as for XML files opened with the Oxygen XML Developer plugin plugin.

TocJS Transformation Does not Generate All Files for a Tree-Like TOC

Problem

The *TocJS* transformation of a DITA map does not generate all the files needed to display the tree-like table of contents.

Solution

To get a complete working set of output files, 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 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".

Topic References Outside the Main DITA Map Folder

Referencing to a DITA topic, map or to a binary resource (for example: image) that 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 it well when links to topics that 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 that 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 or not the application tries 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. The allowed values are "false" and "true". The default value is false.



Important: The **fix.external.refs.com.oxygenxml** parameter is only supported when the DITA OT transformation process is started from Oxygen XML Developer plugin)

XSLT Debugger Is Very Slow

Problem

When I run a transformation in the **XSLT Debugger** perspective it is very slow. Can I increase the speed?

Solution

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).

Chapter

18

Glossary

Topics:

- [*Active cell*](#)
- [*Apache Ant*](#)
- [*Block element*](#)
- [*Bookmap*](#)
- [*DITA Map*](#)
- [*DITA_OT_DIR*](#)
- [*Inline element*](#)
- [*Java Archive*](#)
- [*Named User*](#)

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.

Block element

A block element 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 Developer plugin > DITA preferences page](#).

For example, if you are using DITA OT 2.3.3, `[OXYGEN_INSTALL_DIR]/frameworks/dita/DITA-OT2.x` (or if you are using DITA OT 1.8.5, the default DITA OT directory is: `[OXYGEN_INSTALL_DIR]/frameworks/dita/DITA-OT`). You can also specify a custom directory.

Inline element

An inline element 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 or an archive tool.

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 Developer plugin) to use the software regardless of whether or not 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 Developer plugin [End User License Agreement](#).

Index

A

- Action dialog box 39
- Add nodes in Grid Mode 190
- Add Schematron Quick Fix 396
- Annotation preferences 53
- Antenna House Formatter processor 468, 506
- Ant transformation scenario 528, 529, 530
- Options tab 529
- Output tab 530
- Parameters tab 529
- Archive preferences 28
- Archives 681, 682, 683, 685
- Browse 682
- Edit files 685
- EPUB 683
- File browser 682
- Migrate OOXML to DITA 685
- Migrate OOXML to TEI 685
- Modify 682
- ODF 683
- OOXML 683
- Associate schema directly in XML documents 215
- Associate schema in a validation scenario defined in framework 214
- Associate schema in Document Type Association 217
- Associate schema through a validation scenario 211
- Associate schema to an XML document 211
- Attributes view in Design mode 133, 307
- Attributes view in Text mode 119, 173
- Author editing mode 120, 122, 172, 175
- Entities view 122, 175
- Model view 120, 172
- Automatic Spell Check 424
- Automatic validation 194

B

- BaseX database connection 727
- BaseX database contextual menu actions 727
- BaseX XQJ connection 729
- Batch transformation 547
- Berkeley DB XML database connection 705
- Berkeley DB XML database contextual menu actions 706
- Berkeley DB XML debugging 709, 737
- Bidirectional text in Grid mode 128

C

- Canonicalize files tool 427, 811
- Certificates preferences 81
- Change file permissions on remote FTP server 150
- Change language for interface 104
- Changing the font size in Text mode 162
- Character Map dialog box 138
- Check Spelling 418
- Check Spelling in Files 424
- Check Well-Formedness action 193
- Clear content in Grid Mode 190
- Close file 152
- Code template preferences 67

- Code templates 176
- Common problems and solutions 820
- Configure the Application 74
- scenarios management 74
- Configure transformation scenario 543
- Configure Transformation Scenario dialog box 545
- Configuring options 96
- Configuring Oxygen 26
- Console view 225
- Console view preferences 74
- Content Completion Assistant in Grid Mode 192
- Content completion helper views 172
- Content completion in Text Mode 169
- Content completion preferences 52
- Content Management System integration 737
- Contextual menu actions in Text Mode 183
- Contextual menu of current editor tab 153
- Convert database to XML Schema 329, 800
- Convert schema to another schema language 327, 798
- Convert XML to JSON 379, 801
- Copy/Paste in Grid Mode 191
- Create DTD from learned document structure 218
- Create Markdown documents 405
- Create new transformation scenario 509
- Create new validation scenario 199
- Create Schematron Quick Fix 396
- CSS stylesheets 355, 356, 357
- Content completion 356
- Custom CSS properties 356
- Editing features 355
- Folding 357
- Format and indent (pretty print) 357
- Minifying 357
- Outline view 357
- Validation 356
- CSS validation preferences 29
- Custom editor variables 29
- Customize document templates 146
- Customizing default options 97
- Custom validation engine preferences 55
- Custom XML refactoring operations 234, 785
- Custom XSLT/XQuery transformation engine preferences 83

D

- Database connection preferences 30
- Database drivers 33
- Database perspective 108
- Databases 676, 687, 688, 690, 691, 693, 694, 696, 697, 699, 701, 704, 705, 706, 709, 711, 712, 715, 717, 718, 720, 722, 723, 725, 726, 727, 729, 730, 731, 733, 734, 735, 736, 737, 743, 744
- Connections 690, 691, 693, 694, 696, 697, 699, 701, 704, 705, 706, 709, 711, 712, 715, 717, 718, 720, 722, 723, 725, 726, 727, 729, 730, 736, 737, 743, 744
- BaseX 727, 729
- Contextual menu actions 727
- XQJ 729
- Berkeley DB XML 705, 706, 709, 737
- Contextual menu actions 706
- Debugging 709, 737

Databases (continued)

- Connections (continued)
 - Documentum xDB 718, 720, 722
 - Contextual menu actions 720
 - Parser configuration 722
 - eXist 709, 711
 - Contextual menu actions 711
- Generic JDBC 725
 - IBM DB2 691, 693
 - Contextual menu actions 693
- JDBC-ODBC 726
 - MarkLogic 712, 715, 717, 736
 - Contextual menu actions 717
 - Debugging 715, 736
- MarkLogic development 715
 - Microsoft SQL Server 694, 696
 - Contextual menu actions 696
- MySQL 723
 - Oracle 697, 699
 - Contextual menu actions 699
 - PostgreSQL 701, 704
 - Contextual menu actions 704
 - SharePoint 743, 744
 - Contextual menu actions 744
 - WebDAV 729, 730
 - Contextual menu actions 730
- Data Source Explorer view 687
- SQL support 731
- Table Explorer view 688
 - XQuery 676, 709, 715, 733, 734, 735, 736, 737
 - Debugging 709, 715, 735, 736, 737
 - Berkeley DB XML 709, 737
 - MarkLogic 715, 736
 - Drag and drop from Data Source Explorer 733
 - Transformations 734
 - Validation 676, 735
- Data Source Explorer view 687
- Data Sources preferences page 30
- Debugger preferences 84
- Debugging XQuery 757, 758, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775
- Breakpoints 770
- Identify expressions 771
 - Information views 760, 761, 762, 763, 764, 765, 766, 767, 768
 - Breakpoints view 762
 - Context Node view 761
 - Messages view 763
 - Nodes/Values Set view 767
 - Output Mapping Stack view 764
 - Stack view 764
 - Templates view 766
 - Trace History view 765
 - Variables view 768
 - XPath Watch view 761
- Java extensions 772
- Layout 758
 - Profiling 773, 774, 775
 - Hotspots view 775
 - Invocation Tree view 774
- Steps in a typical debugging process 769
- Toolbar 758
- Debugging XSLT 757, 758, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775
- Breakpoints 770
- Identify expressions 771

Debugging XSLT (continued)

- Information views 760, 761, 762, 763, 764, 765, 766, 767, 768
- Breakpoints view 762
- Context Node view 761
- Messages view 763
- Nodes/Values Set view 767
- Output Mapping Stack view 764
- Stack view 764
- Templates view 766
- Trace History view 765
- Variables view 768
- XPath Watch view 761
- Java extensions 772
- Layout 758
 - Profiling 773, 774, 775
 - Hotspots view 775
 - Invocation Tree view 774
- Steps in a typical debugging process 769
- Toolbar 758
- Debug PDF transformation 550
- Design editing mode 130, 131, 133, 135, 136, 275, 276, 277, 282, 283, 286, 288, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 302, 303, 305, 307
- Attributes view 133, 307
 - Components 282, 283, 286, 288, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299
 - Grouping components 299
- xs:all 294
- xs:alternative 291
- xs:any 295
- xs:anyAttribute 296
- xs:assert 298
- xs:attribute 286
- xs:attributeGroup 288
- xs:choice 294
- xs:complexType 288
- xs:element 283
- xs:field 298
- xs:group 292
- xs:import 293
- xs:include 293
- xs:key 297
- xs:keyRef 297
- xs:notation 294
- xs:openContent 299
- xs:override 293
- xs:redefine 293
- xs:schema 283
- xs:selector 298
- xs:sequence 294
- xs:simpleType 290
- xs:unique 296
- Contextual menu actions 277
- Editing actions 276
- Facets view 135, 303
- Navigation 130, 275
- Outline view 131, 305
- Palette view 136, 302
- Design mode preferences 58
- Detect Master Files 160
- Detect Master Files from Project 160
- Digital Signature 425, 427, 428, 430, 811, 812, 813
- Canonicalize files 427, 811
- Certificates 427
- Sign files 428, 812

- Digital Signature (*continued*)
- Verify signature 430, 813
- Digital Signatures 425, 430
- Example of how to digitally sign XML content 430
- Overview 425
- DITA 463, 465, 467, 468, 500, 502, 503, 504, 505, 506
 - Output 463, 465, 467, 468, 500, 502, 503, 504, 505, 506
 - DITA Map to PDF WYSIWYG transformation 468, 506
 - PDF output customization 463, 465, 467, 468, 500, 502, 503, 504, 505
 - Add watermark 465, 503
 - Creating a customization directory 463, 500
 - Customize 'Note' images 467, 505
 - Customize header/footer 465, 502
 - Force page breaks 467, 504
 - Set font 468, 505
 - Show comments and tracked changes 468, 505
 - PDF output customization 463, 465, 467, 468, 500, 502, 503, 504, 505
 - Add watermark 465, 503
 - Creating a customization directory 463, 500
 - Customize 'Note' images 467, 505
 - Customize header/footer 465, 502
 - Force page breaks 467, 504
 - Set font 468, 505
 - Show comments and tracked changes 468, 505
- DITA and Markdown documents 409
- DITA Map document type 448
- DITA Maps 449, 487
- Transformation scenarios 449, 487
- DITA Map to CHM (Compiled Help) transformation 470, 507
- DITA Map to Kindle transformation 470, 508
- DITA Map to PDF WYSIWYG transformation 468, 506
- DITA Map to WebHelp Classic Mobile transformation 461, 498
- DITA Map to WebHelp Classic transformation 456, 493
- DITA Map to WebHelp Responsive transformation 450, 487
- DITA Map to WebHelp Responsive with Classic transformation 458, 496
- DITA Map to WebHelp Responsive with Feedback transformation 453, 490
- DITA Map transformations 449, 487
- DITA OT preferences 34
- DITA OT transformation scenario 520, 521, 522, 523, 524, 525, 527
 - Advanced tab 525
 - Filters tab 525
 - FO Processor tab 523
 - Output tab 527
 - Parameters tab 524
 - Skins tab 521
 - Templates tab 522
- DITA preferences 34
- DITA Topic document type 447
- DITA Topic to PDF transformation 448, 487
- DITA Topic to XHTML transformation 448, 487
- DocBook 4 document type 434, 479
- Transformation scenarios 434, 479
- DocBook 4 to WebHelp transformation 434, 479
- DocBook 4 transformations 434, 479
- DocBook 5 document type 440, 441, 483
- Transformation scenarios 441, 483
- DocBook 5 to WebHelp transformation 441, 483
- DocBook 5 transformations 441, 483
- DocBook olink 438, 445
- DocBook Targetset document type 447
- DocBook to DITA transformation 438, 445, 482, 486
- DocBook to EPUB transformation 437, 444, 482, 486
- DocBook to PDF transformation 437, 443, 481, 485
- Document checking preferences 56
- Document template preferences 72
- Document templates 145, 146
 - Creating 145, 146
- Customizing 146
- Document Type Association (Framework) 44
- Localizing 44
- Document type association preferences 35
- Document type configuration dialog box 36, 37, 38, 39, 45, 46, 47, 48, 49, 50, 51
- Association rules tab 37
 - Author tab 38, 39, 45, 46, 47
 - Actions subtab 39
 - Content Completion subtab 47
 - Contextual Menu subtab 46
 - CSS subtab 38
 - Menu subtab 45
 - Toolbar subtab 47
- Catalogs tab 49
- Classpath tab 37
- Extensions tab 51
- Schema tab 37
- Templates tab 48
- Transformation tab 49
- Validation tab 50
- Document Types 432
- Document types (Frameworks) 433
- Documentum xDB database connection 718
- Documentum xDB database contextual menu actions 720
- Documentum xDB database parser configuration 722
- Documentum xDB troubleshooting 723
- Drag and Drop in Grid Mode 190
- Drag and Drop in Text Mode 167
- DTD Entities for editing large XML documents 223
- Duplicate nodes in Grid Mode 190
- Duplicate transformation scenario 545

E

- Edit 141
 - create new documents 141
- Edit cell value in Grid Mode 190
- Editing Modes 111, 112, 126, 129, 274
- Design 129, 274
- Grid 126
- Text 112
- Editing XML markup in Text Mode 163
- Edit mode preferences 56
- Editor perspective 106
- Editor preferences 51
- Editor variables 99, 104
 - Custom editor variables 104
- Edit validation scenario 202
- Elements view in Text mode 122, 174
- Entities view 122, 175
- EPUB document type 475
- Executing SQF in Other Documents 401
- eXist database connection 709
- eXist database contextual menu actions 711
- Export Global Options 97
- Export Global Transformation Scenarios 98
- Export Global Validation Scenarios 98
- Export Markdown documents 405

Extension points 777

F

Facets view in Design mode 135, 303
File extension association 98
File properties 153
Find/Replace action 207
Find/Replace dialog box 207
Find/Replace text 207
Find All Elements action 208
Find All Elements dialog box 208
Flatten Schema tool 329
Folding in Text Mode 167
Font preferences 72
Font size configuration in Text mode 162
FO processor preferences 85
Format/Indent multiple files 182
Format and Indent Files tool 182
Format and Indent in Text Mode 179
Formatting preferences 59
Formatting Schematron Quick Fix Content 400
Framework directory locations 36
Frameworks 432
FTP connection settings 72

G

Generate Documentation tools 802
Generate Sample XML Files tool 315, 316, 317, 319, 794, 796, 798
 Advanced tab 319, 798
 Options tab 317, 796
 Schema tab 316, 794
Generate WSDL Documentation tool 349, 808
Generate XML Schema Documentation tool 319, 322, 325, 326, 802
 Customize PDF output 325
 From command line 326
 Output formats 322
Generate XSLT Stylesheet Documentation tool 261, 263, 265, 266, 805
 Custom format 265
 From command line 266
 HTML format 263
Generic JDBC database connection 725
GET parameters for WebHelp Classic 631, 651
GET parameters for WebHelp Responsive 600
Getting Started with Oxygen 2, 3, 4, 6
Help 6
Layout 3
Resources to help you with Oxygen 4
Supported document types 3
Grid editing mode 126, 128, 189, 190, 191, 192
 Add nodes 190
 Bidirectional text 128
 Clear content 190
 Content Completion Assistant 192
 Copy/Paste 191
 Drag and Drop 190
 Duplicate nodes 190
 Edit cell value 190
 Insert columns 190
 Insert rows 190
 Refresh layout 190
 Sort columns 190

Grid editing modes 127
Layout 127
Navigation 127
Grid mode preferences 57

H

Help 6
Support related resources 6
Using the Help menu 6
Highlight ID occurrences in Text Mode 183
Highlights in XML documents 226
HTTP authentication schemes 152
HTTP connection settings 73
HTTPS troubleshooting 150

I

IBM DB2 database connection 691
IBM DB2 database contextual menu actions 693
Import data dynamically 754
Import Global Options 97
Import Global Transformation Scenarios 98
Import Global Validation Scenarios 98
Importing data 747
 Importing data from a database 751
 Importing data from Excel 749
 Importing data from HTML files 754
 Importing Data from text files 748
Import preferences 75
Incorrect Function error 22
Indenting Schematron Quick Fix Content 400
Insert columns in Grid Mode 190
Insert rows in Grid Mode 190
Installing Oxygen 8, 9, 10, 11, 13, 16, 17, 19, 20, 21, 22, 23, 24
Floating license server 16
Group deployment 13
 HTTP floating license server 17, 19, 20
 Activity report 20
 Replacing 19
 Upgrading 19
Installation options 9
Licensing 13
Linux Installation 11
Mac OS X Installation 10
 TCP floating license server 20, 21, 22, 23
 All Platforms distribution 22, 23
 Replacing 23
 Upgrading 23
 Windows 32-bit 20, 21, 22
 Incorrect Function error 22
 Process Terminated Unexpectedly error 22
 Replacing 22
 Upgrading 21
Transfer license key 23
Upgrading 24
Windows Installation 9
Integrate Schematron Quick Fixes in a framework 404
Introduction 1

J

JATS document type 474, 475, 509
Transformation scenarios 475, 509

- JATS transformations 475, 509
- JavaScript documents 381, 383, 384
- Content Completion 383
- Editing features 381
- Outline view 384
- Validation 383
- JDBC-ODBC database connection 726
- Jenkins integration with WebHelp plugin 659
- JSON documents 376, 377, 378, 379, 406, 408, 801
- Folding 377
- Grid mode editing features 378
- Outline view 379
- Syntax highlighting 377, 406, 408
- Text mode editing features 376
- Validation 379
- XML to JSON converter 379, 801

K

- Kerberos 152

L

- License key configuration 28
- License options 28
- Licensing Oxygen 13, 14, 15, 16
 - Floating license 14, 15, 16
 - HTTP license server 15
 - Multiple users 16
 - Release floating license 16
 - TCP license server 15
- Named-user license 14
- Localizing frameworks 44
- Localizing the user interface 104
- Localizing XML refactoring operations 242, 793

M

- Manage highlighted content in Text Mode 183
- Manual validation actions 194
- Markdown documents 405, 407, 409
 - Actions in Markdown Editor 407
- DITA 409
- Markdown Editor 405
- Rules and specifications 409
- Validation 409
- Markdown Editor 405
- MarkLogic database connection 712
- MarkLogic database contextual menu actions 717
- MarkLogic debugging 715, 736
- MarkLogic for the developer 715
- Master Files 159, 160
 - Add files 160
 - Detecting 160
 - Enabling 160
 - Overview 159
- Microsoft SQL Server database connection 694
- Microsoft SQL Server database contextual menu actions 696
- Model view 120, 172
- Modular XML files 218
- Move XML resources 221
- MSXML.NET preferences 92
- MSXML 3.0 and 4.0 preferences 91
- MySQL database connection 723

N

- Navigating in Text Mode 112, 161
- Navigator view 113, 154
- Navigator view preferences 74
- Network connection preferences 72
- New document from templates 145, 146
- New Document Wizard 141
- New from Templates Wizard 145, 146
- New project 5, 153
- NISO Journal Article Tag Suite document type 474
- NTLM 152
- NVDL schemas 371, 372, 373, 374, 375
- Component Dependencies view 374
 - Diagram Editor 371, 372, 373
 - Contextual menu actions 373
 - Full Model view 371
 - Introduction 371
 - Logical Model view 372
- Outline view 373
- Refactoring actions 375
- Search actions 375
- Validation 373

O

- Olink element 438, 445
- Open file 147
- Open file in system application 148
- Open preferences 64
- Open remote document 148
- Open using FTP/SFTP/WebDAV dialog box 148
- Options Menu 27
- Preferences 27
- Oracle database connection 697
- Oracle database contextual menu actions 699
- Outline view in Design mode 131, 305
- Outline view in Text mode 117, 176
- Outline view preferences 96
- Out Of Memory error 85, 820
- oxy:allows-child-element function 42
- oxy:allows-global-element function 43
- oxy:current-selected-element function 44

P

- Palette view in Design mode 136, 302
- Performance preferences 64
- Perspective 106
- XML 106
- Perspectives 105, 107, 108
- Database 108
- XQuery Debugger 107
- XSLT Debugger 107
- PostgreSQL database connection 701
- PostgreSQL database contextual menu actions 704
- Predefined Document Types 433, 434, 440, 447, 448, 471, 472, 473, 474, 475
 - DITA Map 448
 - DITA Topic 447
 - DocBook 4 434
 - DocBook 5 440
 - DocBook Targetset 447
 - EPUB 475

Predefined Document Types (*continued*)

- JATS 474
- TEI ODD 472
- TEI P4 472
- TEI P5 473
- XHTML 471
- Predefined XML refactoring operations 230, 781
- Preferences 27, 28, 29, 30, 33, 34, 35, 36, 37, 38, 39, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 63, 64, 65, 66, 67, 72, 73, 74, 75, 76, 77, 79, 80, 81, 82, 83, 84, 85, 88, 89, 90, 91, 92, 93, 94, 95, 96
 - Appearance 72
 - Fonts 72
- Archive 28
- CSS Validator 29
- Custom Editor Variables 29
 - Data Sources 30, 33
 - Table Filters 33
- DITA 34
 - Document Type Association 35, 36, 37, 38, 39, 45, 46, 47, 48, 49, 50, 51
 - Document type configuration dialog box 36, 37, 38, 39, 45, 46, 47, 48, 49, 50, 51
 - Association rules tab 37
 - Author tab 38, 39, 45, 46, 47
 - Actions subtab 39
 - Content Completion subtab 47
 - Contextual Menu subtab 46
 - CSS subtab 38
 - Menu subtab 45
 - Toolbar subtab 47
 - Catalogs tab 49
 - Classpath tab 37
 - Extensions tab 51
 - Schema 37
 - Templates tab 48
 - Transformation tab 49
 - Validation tab 50
- Locations 36
- Editor 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 63, 64, 65, 66, 67, 72
- Code Templates 67
 - Content Completion 52, 53, 54
 - Annotations 53
 - XPath 54
 - XSD 54
 - XSL 54
- Custom Validation Engines 55
- Document Checking 56
- Document Templates 72
 - Edit Modes 56, 57, 58, 59
 - Author 58
 - Schema Design 58
 - Properties 58
 - Grid 57
 - Text Diagram 59
 - Format 59, 61, 63, 64
 - CSS 61
 - JavaScript 61
 - XML 61, 63
 - Whitespaces 63
 - XPath 63
 - XQuery 64
- Mark Occurrences 64
- Open/Save 64

Preferences (*continued*)

- Editor (*continued*)
 - Spell Check 65, 66
 - Spell Check Dictionaries 66
 - Syntax Highlight 67
 - Elements/Attributes by Prefix 67
- Templates 67
- Network Connection Settings 72, 73
- (S)FTP 72
- HTTP(S)/WebDAV 73
- Trusted Hosts 73
- View 74
 - XML 75, 76, 77, 79, 80, 81, 82, 83, 84, 85, 88, 89, 90, 91, 92, 93, 94, 95
 - Import 75
 - Sample XML Files Generator 76
 - XML Catalog 77
 - XML Parser 79, 80
 - Relax NG 79
 - Schematron 80
 - XML Schema 80
 - XML Refactoring 81
 - XML Signing Certificates 81
 - XProc 82
 - XSLT-FO-XQuery 83, 84, 85, 88, 89, 90, 91, 92, 93, 94, 95
 - Custom Engines 83
 - Debugger 84
 - FO Processor 85
 - Profiler 88
 - XPath 88
 - XQuery 89, 90
 - Saxon-HE/PE/EE 89, 90
 - Advanced 90
 - XSLT 91, 92, 93, 94, 95
 - MSXML 91
 - MSXML.NET 92
 - Saxon6 95
 - Saxon-HE/PE/EE 93, 94
 - Advanced 94
 - XSLTProc 95
 - XML Structure Outline 96
 - Prince Print with CSS processor 468, 506
 - Problems and solutions 820
 - Process Terminated Unexpectedly error 22
 - Projects 5, 113, 153, 154, 158, 159, 160
 - Adding items to projects 5, 153
 - Creating new projects 5, 153
 - Managing resources 113, 154
 - Master files 159, 160
 - Add files 160
 - Detecting 160
 - Enabling 160
 - Overview 159
 - Move resources 158
 - Navigator view 113, 154
 - Rename resources 158
 - Publishing 477
 - Publish WebHelp on SharePoint 602, 633, 653

Q

- Quick Assist feature in Text Mode 178
- Quick fix support in XML documents 226

R

- Read-only files 425
- Rectangular Selection feature 168
- Refactoring XML Documents 228, 779
- Refresh layout in Grid Mode 190
- Regular expressions syntax 209
- Relax NG schemas 358, 359, 360, 361, 362, 364, 366, 368, 369, 370
- Component Dependencies View 366
- Custom datatype library 370
 - Diagram Editor 359, 360, 361
 - Contextual menu actions 361
 - Full Model view 359
 - Introduction 359
 - Logical Model view 359
 - Symbols 360
- Editing in Master Files context 358
- Moving resources 366
- Outline view 362
- Quick Assist feature 369
- Refactoring actions 368
- Renaming resources 366
- Resource Hierarchy/Dependencies view 364
- Search actions 368
- Validation 362
- Rename XML resources 221
- Reset Global Options 97
- Resolve schemas through XML catalogs 218, 223
- Resource Hierarchy/Dependencies view for XML documents 219
- Restricting Schematron Quick Fix operations 400
- Results view 123

S

- Sample XML File Generator preferences 76
- Save file 148
- Save preferences 64
- Save remote document 148
- Saxon 6 XSLT preferences 95
- Saxon HE/PE/EE advanced XQuery preferences 90
- Saxon HE/PE/EE advanced XSLT preferences 94
- Saxon HE/PE/EE XQuery preferences 89
- Saxon HE/PE/EE XSLT preferences 93
- Schema annotations in Text Mode 171
- Schema Design mode 129, 274
- Schema Diagram Editor 129, 274
- Schematron Quick Fixes 227, 395, 396, 397, 400, 401, 402, 403, 404
 - Content Completion 401
 - Customizing 396
 - Defining 396
 - Embedded 403
 - Executing SQF in Other Documents 401
 - Formatting and Indenting 400
 - Highlight occurrences 402
 - Integrating in a framework 404
 - Refactoring actions 402
 - Restricting operations 400
 - Search actions 402
 - SQF Operations 397
 - User Entry operation 400
 - Use-When Condition 400
 - Validation 401
- Schematron Quick Fix Operations 397
 - Add 397
 - Delete 397

- Schematron Quick Fix Operations (*continued*)
- Replace 397
- String Replace 397
- Schematron Schemas 386, 387, 388, 389, 390, 391, 393, 395
 - Content Completion 388
 - Editing features 386
 - Editing in Master Files context 387
 - Embedded 389
 - Highlight occurrences 393
 - Moving resources 393
 - Outline view 390
 - Quick Assist feature 395
 - Refactoring actions 393
 - Renaming resources 393
 - Resource Hierarchy/Dependencies view 391
 - Search actions 393
 - Validation 387
- Search actions for IDs 209
- Search dialog box 207
- Search in current file 207
- Selecting content in Text Mode 168
- Set indent to zero in Text Mode 181
- Set schema for content completion in Text Mode 170
- SFTP connection settings 72
- SharePoint contextual menu actions 744
- SharePoint database connection 743
- Share transformation scenarios 547
- Share validation scenarios 206
- Sharing XML refactoring operations 242, 793
- Shortcut actions in Text Mode 162
- Sign files tool 428, 812
- Signing XML document preferences 81
- Single sign-on 152
- Smart Editing in Text Mode 162
- Sort table columns in Grid Mode 190
- Special character preferences 64
- Spell checking 418, 420, 422, 423, 424
 - Add dictionaries 420
 - Add term lists 422, 423
 - Customizing dictionaries and term lists 420
 - Dictionaries and term lists 420
 - Forbidden words 422
 - Ignored words 423
 - Learned words 423
 - Multiple files 424
 - Replace dictionaries 422
 - Spell Checking 424
 - Automatic spell check 424
 - Spell check preferences 65
 - SQF 227, 395, 396, 397, 400, 401, 402, 403, 404
 - Content Completion 401
 - Customizing 396
 - Defining 396
 - Embedded 403
 - Executing SQF in Other Documents 401
 - Formatting and Indenting 400
 - Highlight occurrences 402
 - Integrating in a framework 404
 - Operations 397
 - Refactoring actions 402
 - Restricting operations 400
 - Search actions 402
 - User Entry operation 400
 - Use-When Condition 400
 - Validation 401

- SQL transformation scenario 543
- StackOverflowException error 56
- Storing XML refactoring operations 242, 793
- StratML documents 380
- Editing features 380
- Syntax highlight preferences 67
- Syntax highlights in Text mode 124

T

- Table Explorer view 688
- TEI ODD document type 472
- TEI ODD transformations 472, 508
- TEI P4 document type 472, 473, 508
- Transformation scenarios 473, 508
- TEI P4 transformations 473, 508
- TEI P5 document type 473, 474, 509
- Transformation scenarios 474, 509
- TEI P5 transformations 474, 509
- Text editing mode 112, 113, 117, 119, 120, 122, 124, 125, 161, 162, 163, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 178, 179, 181, 182, 183, 195
- Attributes view 119, 173
- Content completion 169
- Contextual menu actions 183
- Drag and Drop 167
- Editing XML markup 163
- Elements view 122, 174
- Entities view 122, 175
- Folding 167
- Format and indent 179
- Format and indent multiple files 182
- Highlight ID occurrences 183
- Manage highlighted content 183
- Model view 120, 172
- Navigation 112, 161
- Outline view 117, 176
- Quick Assist feature 178
- Rectangular selection 168
- Schema annotations 171
- Selecting content 168
- Set indent to zero 181
- Set schema for content completion 170
- Shortcut actions 162
- Smart Editing 162
- Syntax highlights 124
- Validation errors 125, 195
- Views 113
- Transformation Scenarios 434, 437, 438, 441, 443, 444, 445, 448, 449, 450, 453, 456, 458, 461, 468, 470, 471, 472, 473, 474, 475, 478, 479, 481, 482, 483, 485, 486, 487, 490, 493, 496, 498, 506, 507, 508, 509, 516, 520, 528, 530, 536, 538, 543, 545, 547, 550, 551, 553, 554
- Ant 528
- Batch transformation 547
- Built-in transformation scenarios 478
- Configure 543
- Configure Calabash with XEP 550
- Configure Transformation Scenario dialog box 545
- Configure XSLT processor extension paths 553
- Custom XSLT processor 553
- Debugging PDF transformations 550
- DITA map 449, 487
- DITA Map to CHM (Compiled Help 470, 507
- DITA Map to Kindle 470, 508

- Transformation Scenarios (*continued*)
- DITA Map to PDF WYSIWYG 468, 506
- DITA Map to WebHelp Classic 456, 493
- DITA Map to WebHelp Classic Mobile 461, 498
- DITA Map to WebHelp Classic with Feedback 458, 496
- DITA Map to WebHelp Responsive 450, 487
- DITA Map to WebHelp Responsive with Feedback 453, 490
- DITA OT 520
- DITA Topic to PDF 448, 487
- DITA Topic to XHTML 448, 487
- DocBook 4 434, 479
- DocBook 4 to DITA 438, 445, 482, 486
- DocBook 4 to EPUB 437, 444, 482, 486
- DocBook 4 to PDF 437, 443, 481, 485
- DocBook 4 to WebHelp 434, 479
- DocBook 5 441, 483
- DocBook 5 to DITA 438, 445, 482, 486
- DocBook 5 to EPUB 437, 444, 482, 486
- DocBook 5 to PDF 437, 443, 481, 485
- DocBook 5 to WebHelp 441, 483
- Duplicate 545
- Integrate external XProc engine 551
- JATS 475, 509
 - New 509, 516, 520, 528, 530, 536, 538, 543
 - Ant 528
 - DITA OT 520
 - SQL 543
 - XML with XQuery 516
 - XML with XSLT 509
 - XProc 536
 - XQuery 538
 - XSLT 530
- PDF output using Calabash XProc processor 550
- Sharing 547
- SQL 543
- Supported XSLT processors 551
- TEI ODD 472, 508
- TEI P4 473, 508
- TEI P5 474, 509
- XHTML to DITA 471, 508
- XML with XQuery 516
- XML with XSLT 509
- XProc 536
- XQuery 538
- XSL-FO processors 554
- XSLT 530
- Transformation Scenarios view 548
- Transformation types 543
- Transforming Documents 478
- Travis CI integration with WebHelp plugin 660
- Trusted hosts connection settings 73

U

- Unicode support 138, 140
- Character Map 138
- Fallback Font Support 140
- Inserting symbols 138
- Opening/Saving documents with unsupported characters 138
- Uninstalling Oxygen 25
- Upgrading Oxygen 24
- User Entry SQF operation 400
- Use-When SQF Condition 400

V

- Validating XML Documents 125, 192, 193, 194, 195, 196, 197, 199, 202, 206, 387
 - Against a schema 194
 - Against Schematron 387
 - Automatic validation 194
 - Check Well-formedness 193
 - Create new validation scenarios 199
 - Customizing error messages 196
 - Custom validators 197
 - Edit validation scenarios 202
 - Manual validation 194
 - Resolving references to remote schemas 206
 - Sharing scenarios 206
 - Text Mode 125, 195
 - Validation engine 'StackOverflowException' error 56
 - Validation errors in Text mode 125, 195
 - Validation preferences 56
 - Verify Signature tool 430, 813

W

- WebDAV connection 729
- WebDAV connection settings 73
- WebDAV contextual menu actions 730
- WebDAV over HTTPS 150
- WebHelp Classic Mobile system 634
- Customizing 634
- WebHelp Classic system 582, 583, 584, 586, 587, 588, 592, 593, 594, 599, 600, 602, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 628, 629, 630, 631, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 649, 650, 651, 653
 - Add button in code snippets (DITA) 614, 636
 - Add custom HTML content 613, 635
 - Add Facebook widget 620, 642
 - Add Favicon 583, 615, 636
 - Add Google+ widget 620, 642
 - Adding additional resources 582, 613, 634
 - Add logo image 583, 615
 - Add Twitter widget 621, 643
 - Add video and audio objects in DITA 584, 615, 636
 - Add videos in DocBook 616, 637
 - Changing icons in the table of contents 617, 638
 - Changing the style of WebHelp Mobile pages 637
 - CSS styling to customize WebHelp output 587, 617, 639
 - Customize highlights in the table of contents 618, 639
 - Customize ordered lists 586, 616, 638
 - Customizing 594, 613, 625, 646
 - Overriding the XSLT processing step 594, 625, 646
 - Disable caching 618, 640
 - Edit scoring in search results 588, 619, 640
 - Exclude DITA topics from search results 588, 619, 641
 - Flag DITA content 588, 619, 641
 - GET parameters 631, 651
 - Indexing Japanese content 593, 625, 646
 - Integrate Google Analytics 622, 643
 - Integrate Google Search 622, 644
 - Localize email notifications 592, 623
 - Localizing the interface for DITA transformations 593, 624, 645
 - Localizing the interface for DocBook transformations 624, 645
 - Overriding the XSLT processing step 594, 625, 646
 - Publishing on SharePoint Site 633, 653
 - Remove Previous/Next links 628, 649

- WebHelp Classic system (*continued*)
 - Right-to-Left languages 600, 630, 651
 - Search engine optimization 599, 629, 650
 - Skin Builder 630
 - Use custom CSS 618, 640
 - WebHelp Classic with Feedback system 563, 565, 566, 605, 609, 611
 - Admin Panel 566, 611
 - Deploying 563, 609
 - Installing 563, 609
 - Manage comments 566, 611
 - Refreshing content 565, 611
 - WebHelp Mobile system 634
 - WebHelp Plugin 653, 654, 655, 659, 660, 661, 663
 - DITA 654, 655, 659, 660
 - Configuring the comments database 659
 - Integrate Output with Jenkins 659
 - Integrate Output with Travis CI 660
 - Integration with DITA OT 654
 - Licensing 654
 - Running external transformation 655
 - Upgrading 654
 - DocBook 661, 663
 - Configuring the comments database 663
 - Integration with DocBook XSL 661
 - Licensing 661
 - Running external transformation 661
 - Upgrading 661
- WebHelp Responsive system 558, 567, 568, 569, 573, 578, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 597, 599, 600, 602, 613, 615, 616, 617, 619, 623, 624, 625, 629, 630, 634, 636, 638, 639, 640, 641, 645, 646, 650, 651
 - Add custom HTML content 582
 - Add Facebook widget 589
 - Add Favicon 583, 615, 636
 - Add Google+ widget 589
 - Adding additional resources 582, 613, 634
 - Add logo image 583, 615
 - Add Tweet widget 590
 - Add video and audio objects in DITA 584, 615, 636
 - Add welcome message 584
 - Change topics displayed in side TOC 597
 - Configure tiles 585
 - CSS styling to customize WebHelp output 587, 617, 639
 - Customization methods 578
 - Customize ordered lists 586, 616, 638
 - Customize the menu 587
 - Customizing 578, 594, 597, 625, 646
 - Change topics displayed in side TOC 597
 - Overriding the XSLT processing step 594, 625, 646
 - Edit scoring in search results 588, 619, 640
 - Exclude DITA topics from search results 588, 619, 641
 - Flag DITA content 588, 619, 641
 - GET parameters 600
 - Indexing Japanese content 593, 625, 646
 - Integrate Google Analytics 591
 - Integrate Google Search 591
 - Localize email notifications 592, 623
 - Localizing the interface for DITA transformations 593, 624, 645
 - Overriding the XSLT processing step 594, 625, 646
 - Publishing on SharePoint Site 602
 - Right-to-Left languages 600, 630, 651
 - Search engine optimization 599, 629, 650
 - Templates 567, 568, 569, 573
 - Components 573
 - Directory structure 568

- WebHelp Responsive system (*continued*)
 - Templates (*continued*)
 - HTML template files 569
 - Template page types 569
- WebHelp Responsive with Feedback system 561, 563, 565, 566, 609, 611
- Admin Panel 566, 611
- Deploying 563, 609
- Installing 563, 609
- Manage comments 566, 611
- Refreshing content 565, 611
- WebHelp System Output 557, 558, 561, 600, 602, 605, 632, 634, 651
- Context-Sensitive WebHelp Classic 632, 651
- Context-Sensitive WebHelp Responsive 600
- WebHelp Classic 602
- WebHelp Classic with Feedback 605
- WebHelp Mobile 634
- WebHelp Responsive 558
- WebHelp Responsive with Feedback 561
- WSDL documents 338, 339, 340, 343, 345, 346, 348, 349, 351, 352, 353, 354, 808, 813, 815
- Component Dependencies view 345
- Content completion 339
- Editing in Master Files context 338
 - Generate documentation for WSDL documents 349, 351, 352, 808
 - Custom format 352
 - From command line 352
 - HTML format 351
- Highlight occurrences 346
- Moving resources 345
- Outline view 340
- Quick Assist feature 348
- Refactoring actions 346
- Renaming resources 345
- Resource Hierarchy/Dependencies view 343
- Search actions 346
 - SOAP analyzer 353, 354, 813, 815
 - Compose SOAP request 353, 813
 - Test remote files 354, 815
 - UDDI Registry Browser 354, 815
- Validation 339
- WSDL SOAP Analyzer tool 353, 354, 813, 815
- Compose SOAP request 353, 813
- Test remote files 354, 815
- UDDI Registry Browser 354, 815

X

- XHTML documents 404
- Editing features 404
- XHTML document type 471
- XHTML to DITA transformations 471, 508
- XInclude for editing large XML documents 223
- XLIFF documents 381
 - Editing features 381
- XML catalog preferences 77
- XML Catalogs 221
- XML documents 112, 117, 119, 120, 122, 125, 160, 161, 162, 163, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 178, 179, 181, 182, 183, 189, 190, 191, 192, 193, 194, 195, 196, 197, 199, 202, 206, 207, 208, 209, 211, 214, 215, 217, 218, 219, 221, 223, 225, 226, 228, 230, 234, 242, 387, 779, 781, 785, 793
- Associate schema 211
- Associate schema directly in XML documents 215

- XML documents (*continued*)
 - Associate schema in a validation scenario defined in framework 214
 - Associate schema in Document Type Association 217
 - Associate schema through a validation scenario 211
 - Author Mode editing 120, 122, 172, 175
 - Entities view 122, 175
 - Model view 120, 172
 - Code templates 176
 - DTD Entities for editing large documents 223
 - Editing in Master Files context 218
 - Find/Replace dialog box 207
 - Find/Replace text 207
 - Find All Elements dialog box 208
 - Grid Mode editing 189, 190, 191, 192
 - Add nodes 190
 - Clear content 190
 - Content Completion Assistant 192
 - Copy/Paste 191
 - Drag and Drop 190
 - Duplicate nodes 190
 - Edit cell value 190
 - Insert columns 190
 - Insert rows 190
 - Refresh layout 190
 - Sort columns 190
 - Highlights 226
 - Learn document structure 218
 - Moving resources 221
 - Quick fix support 226
 - Refactoring 228, 230, 234, 242, 779, 781, 785, 793
 - Custom operations 234, 785
 - Localizing operations 242, 793
 - Predefined operations 230, 781
 - Sharing operations 242, 793
 - Storing operations 242, 793
 - Renaming resources 221
 - Resolve schemas through XML catalogs 218, 223
 - Resource Hierarchy/Dependencies view 219
 - Search actions for IDs 209
 - Status information 225
 - Text Mode editing 112, 117, 119, 120, 122, 161, 162, 163, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 178, 179, 181, 182, 183
 - Attributes view 119, 173
 - Content completion 169
 - Contextual menu actions 183
 - Drag and Drop 167
 - Editing XML markup 163
 - Elements view 122, 174
 - Entities view 122, 175
 - Folding 167
 - Format and indent 179
 - Format and indent multiple files 182
 - Highlight ID occurrences 183
 - Manage highlighted content 183
 - Model view 120, 172
 - Navigation 112, 161
 - Outline view 117, 176
 - Quick Assist feature 178
 - Rectangular selection 168
 - Schema annotations 171
 - Selecting content 168
 - Set indent to zero 181
 - Set schema for content completion 170
 - Shortcut actions 162

XML documents (*continued*)

- Text Mode editing (*continued*)
- Smart Editing 162
- Validation 125, 192, 193, 194, 195, 196, 197, 199, 202, 206, 387
- Against a schema 194
- Against Schematron 387
- Automatic validation 194
- Check Well-formedness 193
- Create new validation scenario 199
- Customizing error messages 196
- Custom validators 197
- Edit validation scenario 202
- Manual validation 194
- Resolving references to remote schemas 206
- Sharing scenarios 206
- Text Mode 125, 195
- XInclude for editing large documents 223
- XML catalogs 221
- XML documents without a schema 218
- XML parser preferences 79
- XML refactoring preferences 81
- XML Refactoring tool 228, 230, 234, 242, 779, 781, 785, 793
- Custom operations 234, 785
- Localizing operations 242, 793
- Predefined operations 230, 781
- Sharing operations 242, 793
- Storing operations 242, 793
- XML Schema Design mode 129, 274
- XML Schema Diagram Editor 129, 130, 131, 133, 135, 136, 274, 275, 276, 277, 282, 301, 302, 303, 305, 307
- Attributes view 133, 307
- Components 282
- Contextual menu actions 277
- Editing actions 276
- Edit namespaces 302
- Facets view 135, 303
- Navigation 130, 275
- Outline view 131, 305
- Palette view 136, 302
- Validation 301
- XML Schema Regular Expression Builder tool 331, 816
- XML Schemas 129, 130, 131, 133, 135, 136, 274, 275, 276, 277, 282, 283, 286, 288, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 301, 302, 303, 304, 305, 307, 309, 311, 313, 315, 316, 317, 319, 322, 325, 326, 327, 329, 331, 333, 334, 794, 796, 798, 800, 802, 816
- Attributes view 133, 307
- Component Dependencies view 311
- Content completion 305
- Convert DB Structure to XML Schema tool 329, 800
 - Design mode editing 129, 130, 135, 136, 274, 275, 276, 277, 282, 283, 286, 288, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 301, 302, 303
 - Components 282, 283, 286, 288, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299
 - Grouping components 299
 - xs:all 294
 - xs:alternative 291
 - xs:any 295
 - xs:anyAttribute 296
 - xs:assert 298
 - xs:attribute 286
 - xs:attributeGroup 288
 - xs:choice 294
 - xs:complexType 288

XML Schemas (*continued*)

- Design mode editing (*continued*)
- Components (*continued*)
- xs:element 283
- xs:field 298
- xs:group 292
- xs:import 293
- xs:include 293
- xs:key 297
- xs:keyRef 297
- xs:notation 294
- xs:openContent 299
- xs:override 293
- xs:redefine 293
- xs:schema 283
- xs:selector 298
- xs:sequence 294
- xs:simpleType 290
- xs:unique 296
- Contextual menu actions 277
- Editing actions 276
- Edit namespaces 302
- Facets view 135, 303
- Navigation 130, 275
- Palette view 136, 302
- Validation 301
- Editing in Master Files context 304
- Flatten Schema tool 329
- Generate/Convert Schema tool 327, 798
 - Generate documentation for XML Schema 319, 322, 325, 326, 802
 - Customize PDF output 325
 - From command line 326
 - Output formats 322
 - Generate Sample XML Files tool 315, 316, 317, 319, 794, 796, 798
 - Advanced tab 319, 798
 - Options tab 317, 796
 - Schema tab 316, 794
- Highlight occurrences 313
- Moving resources 311
- Outline view 131, 305
- Quick Assist feature 315
- Refactoring actions 313
- Regular Expressions Builder 331, 816
- Renaming resources 311
- Resource Hierarchy/Dependencies view 309
- Search actions 313
- Set version 334
- Text mode editing 304
- Validation 304
- XML Schema 1.1 support 333
- XML to JSON tool 379, 801
- XML transformation with XQuery 516
- XML transformation with XSLT 509
- XPath Expressions 665, 667, 668, 669, 671
- Catalogs 668
- Prefix mapping 669
- XPath/XQuery Builder view 665, 671
- XPath Results view 667
- XProc Scripts 385
- Editing features 385
- XProc transformation scenario 536, 537, 538
- Inputs tab 536
- Options tab 538

XProc transformation scenario (*continued*)

- Outputs tab 537
- Parameters tab 537
- XProc tab 536
- XQuery 253, 335, 336, 337, 665, 669, 670, 671, 673, 676, 677, 678, 680, 735, 807
- Content Completion 669
- Folding 336
- Format and Indent 336
- Generate HTML documentation 337, 807
- Input view 253, 673
- Outline view 335, 670
- Sequence view 678
- Syntax Highlights 669
 - Transformations 677, 678, 680
 - Sequence view 678
 - Updating XML documents 680
 - XQJ 677
- Updating XML documents 680
- Validation 676, 735
- XPath/XQuery Builder view 665, 671
- XQuery Debugger perspective 107, 757, 758, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775
- Breakpoints 770
- Identify expressions 771
 - Information views 760, 761, 762, 763, 764, 765, 766, 767, 768
 - Breakpoints view 762
 - Context Node view 761
 - Messages view 763
 - Nodes/Values Set view 767
 - Output Mapping Stack view 764
 - Stack view 764
 - Templates view 766
 - Trace History view 765
 - Variables view 768
 - XPath Watch view 761
- Java extensions 772
- Layout 758
 - Profiling 773, 774, 775
 - Hotspots view 775
 - Invocation Tree view 774
- Steps in a typical debugging process 769
- Toolbar 758
- XQuery Documentation tool 337, 807
- XQuery preferences 89
- XQuery Profiler preferences 88
- XQuery transformation scenario 516, 519, 538, 539, 542
- FO Processor tab 519, 542
- Output tab 519, 542
- XQuery tab 516, 539
- XSLT 243, 244, 245, 246, 249, 250, 253, 256, 258, 259, 260, 261, 263, 265, 266, 267, 269, 270, 272, 673, 805
- Component Dependencies view 258
- Component documentation 260
- Content completion 245

XSLT (*continued*)

- Content completion in XPath expressions 246
- Editing features 243
- Editing in Master Files context 243
 - Generate documentation for XSLT stylesheets 261, 263, 265, 266, 805
 - Custom format 265
 - From command line 266
 - HTML format 263
- Highlight occurrences 259
- Input view 253, 673
- Moving resources 258
- Outline view 250
- Quick Assist feature 267
- Quick fix support 269
- Refactoring actions 270
- Renaming resources 258
- Resource Hierarchy/Dependencies view 256
- Search declarations 260
- Search references 260
- Syntax highlighting 250
- Unit test (XSpec) 272
- Validation 244
- XPath Tooltip Helper 249
- XSLT Debugger perspective 107, 757, 758, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775
- Breakpoints 770
- Identify expressions 771
 - Information views 760, 761, 762, 763, 764, 765, 766, 767, 768
 - Breakpoints view 762
 - Context Node view 761
 - Messages view 763
 - Nodes/Values Set view 767
 - Output Mapping Stack view 764
 - Stack view 764
 - Templates view 766
 - Trace History view 765
 - Variables view 768
 - XPath Watch view 761
- Java extensions 772
- Layout 758
 - Profiling 773, 774, 775
 - Hotspots view 775
 - Invocation Tree view 774
- Steps in a typical debugging process 769
- Toolbar 758
- XSLT Import extension points for WebHelp 594, 625, 646
- XSLT Parameter extension points for WebHelp 594, 625, 646
- XSLT preferences 91
- XSLTProc preferences 95
- XSLT Profiler preferences 88
- XSLT transformation scenario 510, 513, 514, 530, 531, 535
- FO Processor tab 513, 535
- Output tab 514, 535
- XSLT tab 510, 531

Notice

Copyright

Oxygen XML Developer plugin User Manual

Syncro Soft SRL.

Copyright © 2002-2016 Syncro Soft SRL. All Rights Reserved.

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

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

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

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

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

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

Downloading documents. For the most current versions of documentation, see the [Oxygen XML Developer plugin website](#).

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