Oxygen XML Developer
Eclipse Plugin 22.1
User Guide

Contents

Chapter 1. Introduction **1**

Chapter 2. Getting Started **2**

What is Oxygen XML Developer Eclipse plugin 2

Getting Familiar with the Interface 2

Supported Document Types 3

Resources to Help You Get Started Using Oxygen XML Developer Eclipse plugin 4

Your First Document or Project 6

Creating a New Project 6

Getting Help 6

Help Menu 7

Chapter 3. Installation **10**

Installing Oxygen as an Eclipse Plugin 11

Site-Wide Deployment 12

Licensing 13

Choosing a License Type 13

Obtaining a License Key 13

Register a Named-User or Subscription License 14

Floating Licenses 15

Transferring a License Key 18

License Servers 18

Setting up an HTTP License Server (Floating or Named-User Licenses) 19

Setting up TCP Floating License Server (Deprecated) in 32-bit Windows 27

Setting up TCP Floating License Server (Deprecated) All-Platforms 29

Common Problems: License Server Errors 31

Upgrading 33

Upgrading Oxygen XML Developer Eclipse plugin on Windows/Linux 33

Upgrading Oxygen XML Developer Eclipse plugin on OS X 34

Uninstalling 35

Chapter 4. Configuration **36**

Preferences 36

Oxygen XML Developer Eclipse plugin License 37

Archive Preferences ... 37
CSS Validator Preferences ... 38
Custom Editor Variables Preferences ... 39
Data Sources Preferences ... 40
DIFF Preferences ... 45
DITA Preferences ... 47
Document Type Association Preferences ... 50
Editor Preferences ... 76
Fonts Preferences ... 108
Markdown Preferences ... 109
Network Connection Settings Preferences ... 109
Scenarios Management Preferences ... 111
View Preferences ... 112
XML Preferences ... 112
XML Structure Outline Preferences ... 144
Configuring Options ... 145
Customizing Default Options ... 145
Importing/Exporting/Resetting Global Options ... 146
Import/Export Transformation or Validation Scenarios ... 147
Editor Variables ... 148
Custom Editor Variables ... 155
Custom System Properties ... 155
Localizing of the User Interface ... 159
Chapter 5. Perspectives ... 161
Oxygen XML Perspective ... 161
Oxygen XSLT Debugger Perspective ... 162
Oxygen XQuery Debugger Perspective ... 164
Oxygen DB Perspective ... 165
Chapter 6. Editing Modes ... 168
Text Editing Mode ... 168
Grid Editing Mode ... 168
Design Editing Mode (XML Schema Diagram Editor) ... 169
Chapter 7. Working With Any Type of Document ... 171

Creating, Opening, Saving, and Closing Documents....171
Creating New Documents and Templates....171
Opening Documents....182
Saving Documents....183
Closing Documents....183
Working with Remote Documents....183
Open URL....184
Changing File Permissions on a Remote FTP Server....186
WebDAV over HTTPS....186
HTTP Authentication Schemes....189
Contextual Menu of the Current Editor Tab....189
Viewing File Properties....190
Simple Text Editor....190
Using Projects to Group Documents....191
Creating a New Project....191
Navigator View....192
Master Files Support....200
Search and Find/Replace Features....202
Find All Elements Dialog Box....203
Regular Expressions Syntax....204
Spell Checking....205
Spell Check Dictionaries and Term Lists....207
Learned Words....212
Ignored Words (Elements)....213
Automatic Spell Check....213
Spell Check Multiple Files....214
Working with Special Characters and Encoding....215
Unicode Support....215
Opening and Saving Documents with Unsupported Characters....216
Unicode Fallback Font Support....216
Inserting Special Characters with the Character Map....217
Handling Read-Only Files....220
Viewing Status Information....220

Editor Highlights....220
Chapter 8. Editing Documents....222
Editing XML Documents....222
Editing XML Documents in Text Mode....222
Editing XML Documents in Grid Mode....268
Validating XML Documents....277
XML Quick Fixes....301
Associating a Schema to XML Documents....303
Working with XML Catalogs....312
Working with Modular XML Files in the Master Files Context....315
Search and Refactoring Actions for IDs and IDREFS....316
XML Resource Hierarchy/Dependencies View....318
Combining XML Content Using DTD Entities and XInclude....321
Refactoring XML Documents....325
XML Digital Signatures....356
Editing XSLT Stylesheets....363
Editing XSLT Stylesheets in the Master Files Context....363
Validating XSLT Stylesheets....364
XSLT Quick Fix Support....367
Content Completion in XSLT Stylesheets....369
Syntax Highlighting in XSLT....375
XSLT Outline View....376
XSLT Input View....380
XSLT Resource Hierarchy/Dependencies View....382
XSLT Component Dependencies View....384
Highlight Component Occurrences....385
Finding XSLT References and Declarations....386
XSLT Stylesheet Component Documentation Support....387
XSLT 3.0 Text Value Templates....388
XSLT Refactoring Actions....389
XSLT Quick Assist Support....394
XSLT Unit Test (XSpec)....396
Generating Documentation for an XSLT Stylesheet....398

Compiling an XSL Stylesheet for Saxon....404
Editing XML Schemas.... 406
Working with the XML Schema Design Mode (XML Schema Diagram Editor)....406
Editing XML Schema in Text Editing Mode....446
Editing XML Schema in the Master Files Context....446
Validating XML Schema Documents....447
Quick Fixes for DTD, XSD, and Relax NG Errors.... 448
Content Completion in XML Schema.... 448
Syntax Highlighting in XML Schema.... 449
XML Schema Outline View....450
XML Schema Attributes View.... 452
XML Schema Resource Hierarchy / Dependencies View....454
XML Schema Component Dependencies View.... 456
Highlight Component Occurrences....458
Searching and Refactoring Actions in XML Schemas.... 458
XML Schema Quick Assist Support.... 460
Generating Sample XML Files....461
Generating Documentation for an XML Schema....466
Converting Schema to Another Schema Language.... 475
Converting Database to XML Schema.... 478
Flatten an XML Schema.... 478
XML Schema Regular Expressions Builder Tool....480
XML Schema 1.1....482
Setting the XML Schema Version....484
Editing XQuery Documents.... 485
XQuery Validation....485
Content Completion in XQuery....486
Syntax Highlighting in XQuery....487
Formatting and Indenting XQuery Documents.... 487
Folding in XQuery Documents.... 488
XQuery Outline View.... 488
XQuery Builder View....490
XQuery Input View....493

Generating HTML Documentation for an XQuery Document..... 494
Transforming XML Documents Using XQuery..... 495
Editing WSDL Documents..... 500
Editing WSDL Documents in the Master Files Context..... 501
Validating WSDL Documents..... 501
Content Completion Assistance in WSDL Documents..... 501
WSDL Syntax Highlighting..... 502
WSDL Outline View..... 503
WSDL Resource Hierarchy/Dependencies View in WSDL Documents..... 507
WSDL Component Dependencies View..... 510
Highlight Component Occurrences in WSDL Documents..... 511
Searching and Refactoring Operations in WSDL Documents..... 511
Quick Assist Support in WSDL Documents..... 513
Generating Documentation for WSDL Documents..... 514
WSDL SOAP Analyzer Tool..... 519
Editing CSS Stylesheets..... 521
Validating CSS Stylesheets..... 522
Content Completion in CSS Stylesheets..... 522
Syntax Highlighting in CSS Files..... 523
CSS Outline View..... 523
Folding in CSS Stylesheets..... 524
Formatting and Indenting CSS Stylesheets (Pretty Print)..... 524
Minifying CSS Stylesheets..... 524
Editing Relax NG Schemas..... 525
Editing Relax NG Schema in the Master Files Context..... 526
Relax NG Schema Diagram Editor..... 526
Validating Relax NG Schema Documents..... 530
Content Completion in Relax NG Schemas..... 531
Syntax Highlighting in Relax NG Schemas..... 531
Quick Fixes for DTD, XSD, and Relax NG Errors..... 532
Relax NG Outline View..... 533
RNG Resource Hierarchy/Dependencies View..... 535
Relax NG Schema Component Dependencies View..... 538

Searching and Refactoring Actions in RNG Schemas....539
RNG Quick Assist Support....541
Configuring a Custom Datatype Library for a RELAX NG Schema....542
Editing NVDL Schemas....542
NVDL Schema Diagram....543
Validating NVDL Schema Documents....546
Content Completion in NVDL Schemas....546
Syntax Highlighting in NVDL Schemas....547
NVDL Outline View....547
NVDL Schema Component Dependencies View....547
Searching and Refactoring Actions in NVDL Schemas....549
Editing JSON Documents....550
Editing JSON Documents in Text Mode....550
Editing JSON Documents in Grid Mode....551
Validating JSON Documents....552
Content Completion Assistant in JSON....563
Syntax Highlighting in JSON Documents....565
Folding in JSON....565
JSON Outline View....565
XSD to JSON Schema Converter....568
JSON to XML Converter....570
XML to JSON Converter....573
Generating Sample JSON Files....575
Contextual Menu Actions in JSON Documents....577
Transforming and Querying JSON Documents....578
Editing JSON Schema Documents....580
Validating JSON Schema Documents....580
Generating JSON Schema from a JSON File....581
Editing StratML Documents....582
Editing XLIFF Documents....582
Editing JavaScript Documents....583
JavaScript Editing Actions....583
Validating JavaScript Files....585

Content Completion in JavaScript Documents....585
Syntax Highlighting in JavaScript Documents....586
JavaScript Outline View....587
Editing XProc Scripts....588
Editing Schematron Schemas....589
Editing Schematron Schema in the Master Files Context....591
Presenting Schematron Validation Issues....591
Integrating Schematron Rules in a Framework and Sharing Them....592
Validating Schematron Documents....594
Content Completion in Schematron Documents....594
Syntax Highlighting in Schematron....595
Embedding Schematron Rules in XML Schema or RELAX NG....595
Schematron Outline View....597
Schematron Resource Hierarchy/Dependencies View....598
Highlight Component Occurrences in Schematron Documents....601
Searching and Refactoring Operations in Schematron Documents....601
Quick Assist Support in Schematron Documents....603
Schematron Unit Test (XSpec)....604
Editing Schematron Quick Fixes....606
Schematron Quick Fixes (SQF)....606
Defining Schematron Quick Fixes....607
Integrating SQF in a Framework and Sharing Them....616
Validating Schematron Quick Fixes....618
Content Completion in SQF....618
Highlight Quick Fix Occurrences in SQF....618
Searching and Refactoring Operations in SQF....619
Embedding Schematron Quick Fixes in Relax NG or XML Schema....620
Editing HTML Documents....621
HTML Editor....622
HTML Validation....623
HTML Content Completion Assistant....624
Syntax Highlighting in HTML Documents....624
Folding in HTML....625

Minifyng HTML Documents... 625
HTML Outline View... 626
Querying HTML Documents with XPath... 626
Editing Markdown Documents... 626
Markdown Editor... 627
Creating New Markdown Documents... 629
Actions Available in the Markdown Editor... 629
Syntax Highlighting in the Markdown Editor... 632
Automatic Validation in Markdown Documents... 632
Working with Markdown Documents in DITA... 633
Markdown Editor Syntax Rules and Specifications... 634
Chapter 9. Built-in XML Frameworks (Document Types)... 646
DocBook 4 Document Type (Framework)... 646
Inserting an Olink in DocBook Documents... 647
DocBook 5 Document Type (Framework)... 650
Inserting an Olink in DocBook Documents... 651
DocBook 5.1 Assembly... 655
DocBook 5.1 Topic... 656
DocBook Targetset Document Type (Framework)... 657
DITA Topics Document Type (Framework)... 658
DITA Map Document Type (Framework)... 659
XHTML Document Type (Framework)... 660
TEI P5 Document Type (Framework)... 661
Customization of TEI Frameworks Using the Compiled Sources... 662
TEI ODD Document Type (Framework)... 662
jTEI Document Type (Framework)... 663
JATS Document Type (Framework)... 664
EPUB Document Type (Framework)... 665
Other Supported Document Types... 667
Chapter 10. Publishing... 669
Transformation Scenarios... 669
Built-in Transformation Scenarios... 670
Creating New Transformation Scenarios... 691

Editing a Transformation Scenario....762
Duplicating a Transformation Scenario....764
Applying Associated Transformation Scenarios....764
Configure Transformation Scenario(s) Dialog Box....765
Batch Transformations....769
Sharing Transformation Scenarios....770
Transformation Scenarios View....770
WebHelp Output Customization....774
WebHelp Responsive Output for DITA....774
WebHelp Classic Output for DocBook....914
DITA to PDF Output Customization....951
CSS-based PDF Customization....951
XSL-FO to PDF Customization....1107
DocBook to PDF Output Customization....1117
Chapter 11. Working with XPath Expressions....1119
XPath Builder View....1119
XPath Expression Results View....1122
XPath and XML Catalogs....1124
XPath Prefix Mapping....1124
Chapter 12. Working with Archives....1126
Browsing Archives....1126
Working with Archive Files....1128
Creating an Archive....1130
Editing and Saving Files Inside an Archive....1130
Migrating Archives to DITA or TEI....1130
Chapter 13. Databases and SharePoint....1131
Working with Databases....1131
Data Source Explorer View....1131
Table Explorer View....1133
Database Connection Support....1135
WebDAV Connections....1181
SQL Execution Support....1183
XQuery and Databases....1186

Integration with Microsoft SharePoint....1193
How to Configure a SharePoint Connection....1194
SharePoint Contextual Menu Actions....1194
Chapter 14. Importing Data....1198
Import from Text Files....1198
Import from MS Excel Files....1200
Import Data from MS Excel 2007 or Newer....1202
Import Database Data as an XML Document....1203
Import from HTML Files....1206
Import Content Dynamically....1206
Chapter 15. XSLT/XQuery Debugging....1210
Debugger Layout....1211
Control Toolbar....1212
Debugging Information Views....1215
Multiple Output Documents in XSLT 2.0 and XSLT 3.0....1227
Steps in a Typical Debugging Process....1227
Identify the XSLT / XQuery Expression that Generated Particular Output....1228
Using Breakpoints....1229
Performance Profiling of XSLT Stylesheets and XQuery Documents....1230
Invocation Tree View....1232
Hotspots View....1233
Debugging XSLT that Call Java Extensions....1235
Debugging Java Extensions....1235
Supported Processors for XSLT / XQuery Debugging....1236
Chapter 16. Extension Points for the Oxygen Eclipse Plugin....1237
Chapter 17. Tools....1238
XML Refactoring....1238
Built-in Refactoring Operations....1241
Custom Refactoring Operations....1249
Storing and Sharing Refactoring Operations....1262
Localizing XML Refactoring Operations....1263
Generate Sample XML Files....1264
Generate/Convert Schema....1269

Convert DB Structure to XML Schema ... 1272
Compile XSL Stylesheet for Saxon ... 1272
JSON Tools ... 1274
JSON to XML ... 1274
XML to JSON ... 1277
XSD to JSON Schema ... 1280
Generate Sample JSON Files ... 1283
Generate JSON Schema ... 1284
Generate Documentation ... 1285
XML Schema Documentation ... 1285
XSLT Stylesheet Documentation ... 1289
XQuery Documentation ... 1292
WSDL Documentation ... 1293
Canonicalize ... 1296
Sign ... 1298
Verify Signature ... 1300
WSDL SOAP Analyzer ... 1300
Testing Remote WSDL Files ... 1302
XML Schema Regular Expressions Builder ... 1302
Chapter 18. Common Problems ... 1305
Performance Problems and Solutions ... 1305
Out of Memory on External Processes ... 1305
Performance Issues with Large Documents ... 1305
Misc Problems and Solutions ... 1306
Application Reports Errors When Starting After Installing a New Version ... 1306
Application Takes Several Minutes to Start ... 1306
Compatibility Issue Between Java and Certain Graphics Card Drivers ... 1307
Damaged File Associations on OS X ... 1307
Details to Submit in a Request for Technical Support Using the Online Form ... 1308
DITA Map Transformation Fails (Cannot Connect to External Location) ... 1309
DITA Map WebHelp Transformation Fails (Duplicate Topic References Found) ... 1309
DITA-OT Transformation Takes a Long Time to Process ... 1310
DITA PDF Transformation Fails ... 1310

DITA PDF Processing Common Errors....1311
DITA to CHM Transformation Fails - Cannot Open File....1312
DITA to CHM Transformation Fails - Compilation Failed....1313
Error: 'Address Family Not Supported by Protocol Family; Connect'....1313
Error After Switching Oxygen Products in Eclipse....1314
Hunspell Spell Checker is Unusable on Your Platform Error....1314
High Resolution Scaling Issues....1315
High Resolution Scaling Issues on Linux....1315
Images Appear Stretched Out in the PDF Output....1316
Increasing the Memory for the Ant Process....1317
Mac Touch Bar Function Keys Do Not Work....1317
Mac OS X Keyboard no Longer Works After Inserting Accented Characters....1317
Machine Signature Mismatch Error....1318
MSXML 4.0 Transformation Issues....1319
Navigation to a Web Page is Canceled when Viewing CHM on a Network Drive....1319
References Outside the Main DITA Map Folder....1319
Syntax Highlights Not Available in Eclipse Plugin....1320
TocJS Transformation Does not Generate All Files for a Tree-Like TOC....1321
Windows Service Reports Incorrect Function Error When Starting (TCP Server)....1321
Windows Service Reports Process Terminated Unexpectedly Error (TCP Server)....1321
XSLT Debugger Is Very Slow....1322

Chapter 19. Glossary....1323

Active Cell....1323
Anchor....1323
Apache Ant....1323
Block Element....1323
Bookmap....1323
Canonicalize....1323
Content Completion Assistant....1324
Dockable....1324
Document Type Association....1324
DITA Map....1324
DITA-OT-DIR....1325

Foldable Element....1325
Framework....1325
IDML....1325
Inline Element....1325
Java Archive....1326
Key Space....1326
Keystore....1326
Master File....1326
Named User....1326
Perspective....1327
Plugin....1327
Pretty-Print....1327
QName....1328
Quick Assist....1328
Quick Fix....1328
Root Map....1328
WebHelp Output Directory....1329
Working Set....1329
XML Catalog....1329
Index....a
Copyright....am

1.

Introduction

Welcome to the User Manual of Oxygen XML Developer Eclipse plugin 22.1.

Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse 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 Eclipse 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.

2.
Getting Started

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 Eclipse plugin would be to install the software. For detailed information about that process, see the Installation chapter *(on page 10)*.

After installation, when you launch Oxygen XML Developer Eclipse 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.

What is Oxygen XML Developer Eclipse plugin

Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin, see the Oxygen Website.

Getting Familiar with the Interface

Oxygen XML Developer Eclipse plugin includes several *perspectives (on page 1327)* and editing modes *(on page 168)* to help you accomplish a wide range of tasks. Each *perspective* and editing mode also includes a large variety of helper views, menu actions, toolbars, and contextual menu functions.

Regardless of the *perspective (on page 1327)* or editing mode *(on page 168)* that you are working with, the default layout consists of the following areas:

Menus

Menu-driven access to all the features and functions available in Oxygen XML Developer Eclipse plugin. Most of the menus are common for all types of documents, but Oxygen XML Developer Eclipse 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 Eclipse plugin includes a large variety of *dockable (on page 1324)* 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 *(on page 168)* 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 *(on page 168)* also includes a variety of contextual menu actions and other features to help streamline your editing tasks. Each file that has been opened has a tab at the top of the editing pane.

Perspectives

Oxygen XML Developer Eclipse plugin includes several different *perspectives (on page 161)* 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 Eclipse plugin for the first time. Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin

Configuring Oxygen XML Developer Eclipse plugin

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

See the Configuring Oxygen section *(on page 36)* for details on the various ways that you can configure the application and its features.

Video Tutorials

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

Go to the Oxygen XML Developer Eclipse plugin Videos Page to see the list of video tutorials and webinars.

Oxygen XML Developer Eclipse plugin Documentation

The Oxygen XML Developer Eclipse 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 *(on page 168)*** - Provides information about the **Text** editor.
- **XML Schema Diagram Editor** *(on page 169)* - Provides information about the schema design mode.
- **Editing Specific Document Types Chapter *(on page 222)*** - Includes information about editing numerous different types of documents.

- **Publishing Chapter** *(on page 669)* - Provides information about the various ways that you can publish content.
- **Importing Data Chapter** ***(on page 1198)*** - Provides information about importing data from text files, MS Excel files, database data, and HTML files.
- **Tools Chapter** *(on page 1238)* - Details about the various built-in tools that are available in Oxygen XML Developer Eclipse plugin.

Sample Documents

Your installation of Oxygen XML Developer Eclipse 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 Eclipse plugin. You will find files and folders for various types of documents, including the following:

- **Sample project file (*sample.xpr*)** - 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 *(on page 192)*.
- **Sample files (*personal.xml, etc.)*** - 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 Eclipse plugin:

- See the Oxygen XML Developer Eclipse plugin Blog Site for a large variety of current and archived blogs regarding numerous features, requests, and instructional topics.
- Take advantage of the Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin, see the Tools section *(on page 1238)*.
- 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 Eclipse plugin *frameworks (on page 1325)* and *plugins (on page 1327)*, and to integrate Eclipse plugins.
- For a list of new features that were implemented in the latest version of Oxygen XML Developer Eclipse plugin, see the What's New Section of the Website

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 Eclipse 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 *(on page 192)* 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 *(on page 192)*.

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 *(on page 192)* will display the existing files and subdirectories. If your files are scattered among 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 *(on page 192)* 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 192)*.

Related Information:

Using Projects to Group Documents *(on page 191)*

Getting Help

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

- The Oxygen XML Developer Eclipse plugin Support Section of the Website
- The Oxygen XML Developer Eclipse plugin Forum
- The Oxygen XML Developer Eclipse plugin Video Tutorials
- The Common Problems and Solutions Section of the User Manual *(on page 1305)*
- The Online Technical Support Form

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

Help Menu

The Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse 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 (on page 1327)* with your report.
- **Frameworks** - You can choose whether or not to include details about your installed *frameworks (on page 1325)* 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 that will have the text content randomized. You can then save the resources and send them to the *Oxygen* support team without fear of compromising sensitive or private data. For more information, see Randomize XML Text Content *(on page 8)*.

⚠ **Warning:** Before using this action, it is highly recommended that you copy the XML resources to be processed, save them in a separate folder, and then process this operation on the copies instead of the original files. Otherwise, you may lose your original content.

About Eclipse

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

Related Information:

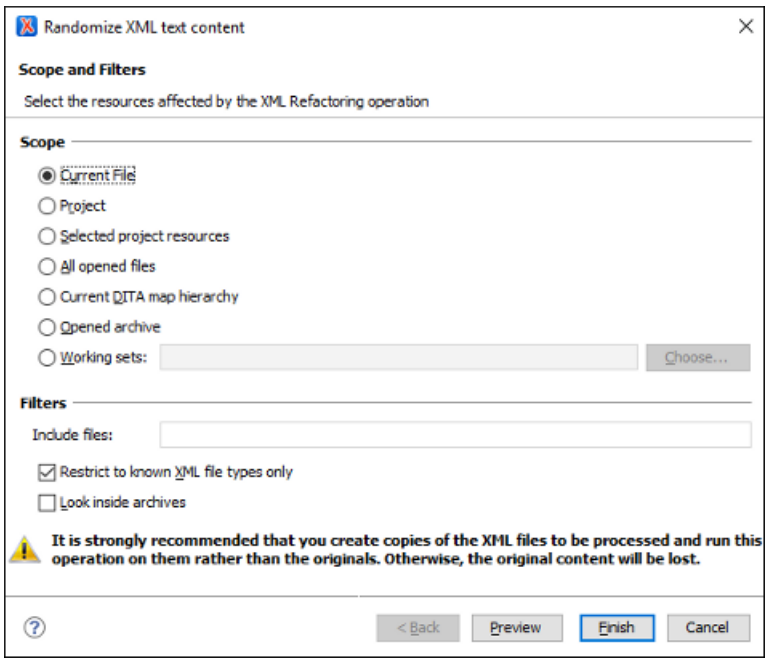
Details to Submit in a Request for Technical Support Using the Online Form *(on page 1308)*

Randomize XML Text Content

Oxygen XML Developer Eclipse plugin includes an action that randomizes the text content of an XML document. This action is available in the **Help > Support Tools** menu. It is helpful if you need to send XML 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 that will have the text content 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, it is highly recommended that you copy the XML resources to be processed, save them in a separate folder, and then perform this operation on the copies instead of the original files. Otherwise, you may lose your original content.

Figure 1. Randomize XML Text Content Dialog Box



The **Randomize XML Text Content** dialog box includes the following options:

Scope

Allows you to select the set of files whose text content will be randomized by the operation. You can select from predefined resource sets (such as the current file, your whole project, the current *DITA map (on page 1324)* hierarchy for DITA projects, etc.) or you can define your own set of resources by creating a *working set (on page 1329)*.

Filters

This 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 selected, only resources with a known XML file type will be affected by the operation.

3.
Installation

Oxygen XML Developer Eclipse 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 Eclipse plugin on your system or server. This section also includes information about registering, transferring, or releasing licenses, upgrading, installing *add-ons*, and uninstalling.

Choosing How Oxygen XML Developer Eclipse plugin Runs

You can install Oxygen XML Developer Eclipse plugin to run in several ways:

- As a desktop application (running standalone or as an Eclipse plugin) on Windows, Linux, or Mac OS X.
- As a desktop application (running standalone or as an Eclipse plugin) on a Unix or Linux server or on Windows Terminal Server.

Choosing an Installer

You also have a choice of several different installers:

- The native installer for your platform (Windows, Linux, or Mac OS X).
- On Windows and Linux, the native installer can also run in unattended mode.
- The Update Site installer *(on page 11)*.
- The Zip archive installer *(on page 12)*.

Choosing a License Option

You must obtain and register a license key *(on page 13)* to run Oxygen XML Developer Eclipse plugin.

You can choose from two types of licenses:

- A named-user 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 *(on page 33)* Oxygen XML Developer Eclipse plugin, transfer a license *(on page 18)*, or uninstall *(on page 35)* Oxygen XML Developer Eclipse plugin.

Getting help with installation

If you need help, email support at: support@oxygenxml.com.

Installing Oxygen as an Eclipse Plugin

System Requirements

Operating Systems

- Windows Vista, Windows 7, Windows 8, Windows 10, Windows Server 2008, Windows Server 2012
- OS X version 10.11 64-bit or later
- Any Unix/Linux distribution with an available Java SE Runtime Environment version 1.8 from Oracle

CPU

- Minimum - 1 GHz processor
- 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

Java 1.8 from Oracle.

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

Eclipse

The following Eclipse versions are officially supported: 3.6-3.8, 4.2-4.15.

Update Site Method

To install the Eclipse plugin using the *Update Site* method, follow this procedure:

1. Start Eclipse.
2. Go to **Help > Install New Software > Available Software**.
3. Click **Add** in the **Available Software** dialog box.
4. Enter `https://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 Eclipse 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 Eclipse plugin is installed correctly by creating a new XML Project. Go to **File > New > Other** and choose **Oxygen XML Developer Eclipse plugin > XML Project**.
10. When prompted for a license key, enter the license information received in the registration email.

Note: If you already have a native version of Oxygen XML Developer Eclipse 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.

Zip Archive Method

To install the Eclipse plugin using the *Zip Archive* method, follow this procedure:

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

Note: If you already have a native version of Oxygen XML Developer Eclipse 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 Marketplace

It is also possible to install Oxygen XML Developer Eclipse plugin from the Eclipse Marketplace. Simply search for **Oxygen** and use the **Install** button that has instructions when you hover over the button.

Site-Wide Deployment

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

Creating custom default options

You can create a custom set of default options *(on page 145)* for Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin project files *(on page 191)* are used to configure a project. You can create and deploy default project files *(on page 191)* for your projects so that your users will have a preconfigured project file to begin work with.

Using floating licenses

If you have a number of people using Oxygen XML Developer Eclipse plugin on a part-time basis or in different time zones, you can use a floating license *(on page 15)* so that multiple people can share a license.

Licensing

Oxygen XML Developer Eclipse plugin is not free software. To activate and use Oxygen XML Developer Eclipse 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 Eclipse plugin:

1. A **Named-User License** *(on page 14)* may be used by a single *Named User (on page 1326)* 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** *(on page 15)* may be used by any user on any machine. However, the total number of copies of Oxygen XML Developer Eclipse 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 Eclipse plugin (for example, Standalone and Eclipse Plugin) at the same time on the same computer, consumes a single floating license.
3. A **Subscription** license *(on page 14)* 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 Key

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

- You can purchase one or more licenses from the Oxygen XML Developer Eclipse plugin website at https://www.oxygenxml.com/buy.html or through one of the authorized resellers. A license key will be sent to you by email.

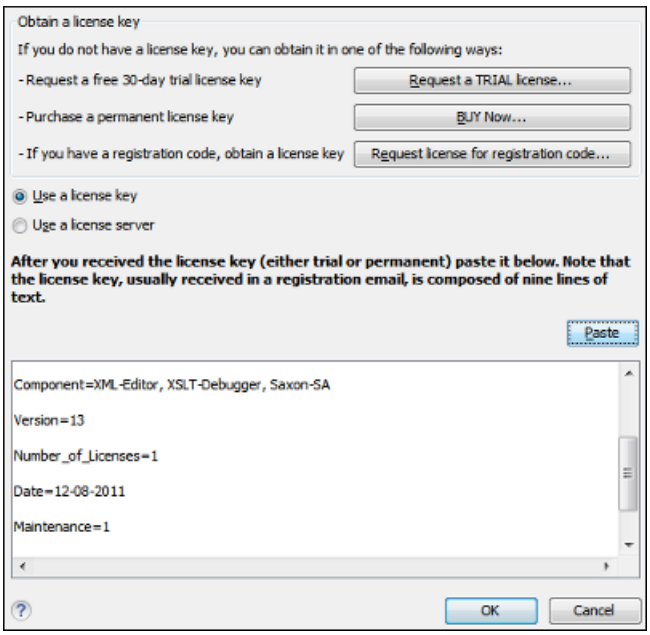
- If your company or organization has already purchased licenses, contact your license administrator to obtain a license key or configuration details to connect to a license server.
- If you purchased a subscription and you received a registration code, you can use it to obtain a license key from https://www.oxygenxml.com/registerCode.html. 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 Eclipse 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 (on page 1326)*, follow these steps:

1. Purchase a license from the Oxygen XML Developer Eclipse 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 Eclipse plugin.
   If this is a new installation of Oxygen XML Developer Eclipse 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 Eclipse plugin** and click the **Register** button.

   Figure 2. License Registration Dialog Box

4. Select **Use a license key** as the licensing method.

   Note: If your license key has 20 or more licenses, you must use a license server *(on page 18)* instead.
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. Click **OK**.

Related Information:

Oxygen XML Developer Eclipse plugin End-User License Agreement

Floating Licenses

The floating license type is commonly used by organizations that have a large number of infrequent users who do not need to use Oxygen XML Developer Eclipse plugin all at the same time. Instead of each user having their own individual license key to use at their discretion, there are a pool of licenses that are used one at a time by various users upon request.

To use floating licenses, a license server is required and the license key needs to be activated. Your system administrator will most likely be responsible for setting up the license server *(on page 18)*. Then you will need to request a floating license from the server *(on page 15)*. This process is designed to help you comply with the *Oxygen End-User License Agreement (EULA)*. This means that the license key will be locked to a particular license server deployment, and the same license key cannot be used with any other license server.

For information about releasing and returning a floating license to the pool for other users, see Releasing a Floating License *(on page 16)*.

For information about reserving (or locking) a floating license so that it does not get returned to the pool, see Reserving a Floating License *(on page 17)*.

Requesting a Floating License from an HTTP License Server

How to Request a Floating License

To request a floating license from an HTTP license server, follow this procedure:

1. Contact your server administrator to make sure the license server has already been set up and get network address and login details for the license server.
2. Start the Eclipse platform.
3. Open the **Preferences** dialog box *(on page 36)* and click the **Register** button.

   Step Result: 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.

Result: If a floating license is available, it is registered in Oxygen XML Developer Eclipse plugin. To display the license details, open the **Preferences** dialog box *(on page 36)*. If a floating license is not available, you will get a message listing the users currently using floating licenses.

How to Register Floating Licenses for Additional Users

If you are an administrator and you want to register floating licenses for multiple users without having to open Oxygen XML Developer Eclipse plugin on each machine to manually configure the registration details one by one, you can use the following procedure:

1. Reset the registration details in Oxygen XML Developer Eclipse plugin:
    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 *(on page 15)*.
3. Copy the `license.xml` file from the `preferences` directory *(on page 37)* and place it in the installation folder on each machine to be registered.

Related Information:

Setting up an HTTP License Server (Floating or Named-User Licenses) *(on page 19)*

Requesting a Floating License from a TCP License Server (Deprecated)

Use this procedure if your company uses an Oxygen XML Developer Eclipse 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 *(on page 36)* 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 Eclipse plugin. To display the license details, open the **Preferences** dialog box *(on page 36)*. If a floating license is not available, you will get a message listing the users currently using floating licenses.

Related Information:

Setting up TCP Floating License Server (Deprecated) in 32-bit Windows *(on page 27)*

Setting up TCP Floating License Server (Deprecated) All-Platforms *(on page 29)*

Releasing 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.
- The Oxygen XML Developer Eclipse plugin will consume one license from the server's pool of licenses if at least one *Oxygen* editor window is opened (not necessarily focused). In other words, if a user wants to release a license, all *Oxygen* editor windows must be closed.
- You exit the application running on your machine, and no other copies of Oxygen XML Developer Eclipse plugin running on your machine are using your floating license.
- You register a *Named User (on page 1326)* license with your copy of Oxygen XML Developer Eclipse plugin, and no other copies of Oxygen XML Developer Eclipse plugin running on your machine are using your floating license.
- Your computer idles for more than 2 hours.
- Your system administrator manually revokes the license *(on page 23)*.

Tip: To prevent your floating license from being released, you can use the **Lock floating license** action available in the **Preferences** dialog box (go to **Window** (**Eclipse** on Mac OSX) and choose **Preferences > Oxygen XML Developer Eclipse plugin**. You can use the same action to unlock the license. Note that your system administrator can also unlock your license *(on page 24)*.

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

1. Open the **Preferences** dialog box *(on page 36)* 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 (on page 1326)* 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 Eclipse plugin.

Reserving a Floating License

There are instances where you might want to reserve or lock a floating license. For example, you could lock a floating license if you want to use your floating license offline while traveling.

To reserve/lock a floating license, follow these steps:

1. Select **Lock floating license** from the **Preferences** dialog box (go to **Window** (**Eclipse** on Mac OSX) and choose **Preferences > Oxygen XML Developer Eclipse plugin**.
2. Click **OK**.
   Your floating license is now locked. You can use the same action to unlock the license or you can contact your system administrator to unlock it.

Transferring a License Key

If you want to transfer your Oxygen XML Developer Eclipse 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 page 13)* on the new computer in the normal way.

To unregister a license, prior to transferring it, follow this procedure:

1. Open the **Preferences** dialog box *(on page 36)* and click **Register**.
   The license registration dialog box is displayed.
2. Make sure the **Use a license key** option is selected.
3. The license key field should be empty (this is normal). If it is not empty, delete any text in the field.
4. Click the **Remove** button at the bottom-right corner of the dialog box.
   A confirmation message is displayed asking if you want to remove your license key.
5. Select between:
   - **Yes** - Removes your license key from your user account on the current computer.
   - **No** - Falls back to your previous license key, if applicable.

License Servers

Installing a License Server to Manage Licenses

If you are using floating licenses or a large number of user-based licenses (20 or more) for Oxygen XML Developer Eclipse plugin, you must set up an Oxygen XML Developer Eclipse plugin license server. A license server can be installed as one of the following:

- An HTTP server *(on page 19)*. This is the recommended method.
- A TCP server *(on page 27)* (deprecated).

Note: Oxygen XML Developer Eclipse plugin version 17 or higher requires a license server version 17 or higher. License servers version 20.1 or higher can be used with any version of a floating or named-user license key.

Activating License Keys

To help you comply with the Oxygen XML Developer Eclipse plugin EULA (terms of licensing), all floating or named-user 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 Eclipse 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. If you have multiple license keys for the same Oxygen XML Developer Eclipse 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, contact the Oxygen support team and ask for a new license key.

Setting up an HTTP License Server (Floating or Named-User Licenses)

The HTTP License Server is available in several distributions, tailored for covering a variety of deployment configurations:

- **Windows installer** *(on page 19)* - Easy-to-use Windows installation wizard. Requires elevated permissions to run it.
- **All-platform distribution** *(on page 20)* - Script-based deployment that does not require elevated permissions to run it. Provides scripts for Windows, Mac, and Linux.
- **Web Archive (WAR) distribution** *(on page 20)* - Provides more flexibility in your deployment configuration, but it requires an existing HTTP server (such as Apache Tomcat).

Installing the HTTP License Server Installer Distribution for Windows

1. Download the HTTP license server installer from the HTTP License Server 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 Eclipse plugin license server administrative interface.
    b. **Standard user credentials** - Used by an Oxygen XML Developer Eclipse plugin application to connect to the license server.

    Note: It is possible to configure the credentials for users after the installation. Once the installation is complete, there will be a `password.properties` file located in the license server's installation directory. You can edit this file and follow the instructions at the top to assign *admin*, *manager*, or *user* roles to each user. There is also a `readme.txt` file in the same directory that has more information.
4. You can choose to change the default 8080 port the server will run on. If you need to change the port after the installation, you can do it by editing the following *vmoptions* file: `oXygen HTTP License Server\Windows Service\oXygenHTTPLicenseServer.vmoptions`.
5. Optionally, you can choose to install the server as a Windows service. In this case, you can choose the name of the Windows service.

Tip: In case you run into issues, the license server log file is located in: `[Installation_Directory]\temp\logs\oXygenLicenseServlet.log`.

Installing the HTTP License Server All-Platform Distribution

1. **[Prerequisite]** Java 8 or later must be installed.
2. Download the HTTP license server all-platform archive from the HTTP License Server website.
3. Unpack the archive.
4. 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`). You can also change the port by editing the following *vmoptions* file: `oXygen HTTP License Server\Windows Service \oXygenHTTPLicenseServer.vmoptions`.
5. On the first run, you will be prompted to set two sets of credentials:
   a. **Administrator credentials** - Used for accessing the Oxygen XML Developer Eclipse plugin license server administrative interface.
   b. **Standard user credentials** - Used by an Oxygen XML Developer Eclipse plugin application to connect to the license server.

      Note: It is possible to configure the credentials for users after the installation. Once the installation is complete, there will be a `password.properties` file located in the license server's installation directory. You can edit this file and follow the instructions at the top to assign *admin*, *manager*, or *user* roles to each user. There is also a `readme.txt` file in the same directory that has more information.

      Tip: 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.

      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.

Installing the HTTP License Server WAR Distribution

1. Make sure that you have Java Servlet Container installed on the machine you have selected to be the license server. Apache Tomcat 5.5 or higher is recommended (available at http://tomcat.apache.org).

Important: By default, the license server stores the statistics database and other data in the Java Servlet Container's temporary directory. If you are not using Apache Tomcat, this directory may be deleted when the server is stopped or restarted. However, you can set the `oxygen.license.server.work.dir` system property to specify a different path for the directory where the database is stored.

2. Download the HTTP license server **Web Archive** (**.war**) from the HTTP License Server website.
3. Configure three user roles in your installation of the Java Servlet Container (such as Apache Tomcat):
    a. One user with the role *user*, used by an Oxygen XML Developer Eclipse plugin application to connect to the license server. In the subsequent example, this user name is **John**.
    b. Another user with the role *admin*, used for accessing the HTTP License Server administrative interface and the management interface. In the subsequent example, this user name is **Mary**.
    c. Another user with the role *manager*, used for accessing the HTTP License Server main page and the statistics pages. In the subsequent example, this user name is **Henry**.

    For example, in Apache Tomcat, 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 three 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"/>
  <user username="John" password="user_pass" roles="user"/>
  <user username="Mary" password="admin_pass" roles="admin"/>
  <user username="Henry" password="admin_pass" roles="manager"/>
</tomcat-users>
```

4. Deploy the WAR file.

    For example, in Apache Tomcat, go to the Web Application Manager page and log in with the user you configured with the *admin* 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 HTTP License Server administration page. By default, the address of this page is `http://<server-address>/oXygenLicenseServlet`. In Apache Tomcat, you can also open this 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 the example above).
6. **Activate the license key**. This process involves binding your license key to your license server deployment. The browser used in the activation process needs to have Internet access.

Note: If you cannot access the internet during the deployment, you can manually activate the license key *(on page 22)*.

Once the process is completed you cannot activate the license on another license server. Follow these steps to activate the license:

a. Paste your license key into the form and click **Register/Activate**.

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

b. Click **Register/Activate**.

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

7. The application's log file location is specified by the `log4j.appender.R2.File` property from the `WEB-INF/lib/log4j.properties` configuration file.

   For example, in Apache Tomcat, the configuration file is located at `TomcatInstallDir/webapps/oXygenLicenseServlet/WEB-INF/lib/log4j.properties` and the default log file location is `TomcatInstallDir/logs/oxygenLicenseServlet.log`.

Installing Multiple Instances of the Oxygen License Servlet on a Tomcat Web Server

For cases where your organization has multiple sets of licenses (for example, an integrator with multiple clients might host a different license server for each client), follow this procedure to install multiple instances of the *Oxygen License Servlet* on a Tomcat web server:

1. Rename the license server WAR file according to your needs. For example, you could use the customer name and a number (e.g. *client23415*).
2. Go to your Tomcat license server manager (e.g. `http://my.tomcatserver.com:port/manager/`) and enter your credentials.
3. Scroll to **WAR file to deploy** and press **Browse** button.
4. Locate the WAR file from step 1 and press the **Open** button.
5. Press the **Deploy** button.
6. Check that the newly deployed license server is running (it must be in the **Applications** table).

Manual License Activation Procedure (For Users Whose License Server is Offline)

If your license server is not connected to the internet (therefore, the license cannot be activated automatically during the installation), you can manually activate the license by following these steps:

1. Access the HTTP license server management page in a web browser.
2. Copy the license server machine signature code.
3. Go to the activation page at: http://www.oxygenxml.com/activation/.
4. Enter or paste the license server machine signature code and the license key, then click **Activate**.

   Step Result: The activated license key is displayed on-screen.
5. Copy the activated license key and paste it in the license registration page of the HTTP server.

Backup License Server Information

If you want to use a backup license server, the setup instructions are the same as the procedures for a main license server, but it will require its own separate license key. You do not need to purchase the separate license if it is to be used for a backup server, but you will need to request a separate license key by contacting the Oxygen support team.

Related Information:

Troubleshooting: Machine Signature Mismatch Errors *(on page 31)*

License Server Management and Statistics Pages

A system administrator can manage and access information about the license server at: `http://hostName:port/oXygenLicenseServlet`.

This page provides access to several statistics reports and management tasks. It also shows the current status of the server and provides additional instructions for using the license server with Oxygen XML Developer Eclipse plugin.

This page includes the following links for accessing statistics or managing tasks:

- **Current Allocated Licenses** - Opens the Allocated License Report page *(on page 23)*.
- **Usage Statistics** - Available only for floating licenses. Opens the License Usage Statistics page *(on page 24)*.
- **View License Key** - Use this link to view details about the license key.
- **Replace/Remove License Key** - Use this link if you need to replace or remove the current license key *(on page 26)*.
- **Configuration** - Opens a page where you can configure notification settings and specify whether or not users are allowed to lock licenses. This page can be used for setting up the mail server used for sending emails whenever license requests from users are rejected.
- **Users management** - Available only for named-user licenses. Opens a page where you can manage the list of users who are entitled to use the license key.

Allocated License Report Page

This report page provides a system administrator the ability to revoke or unlock current running instances of licenses and includes the following information:

- **License load** - A graphical indicator that shows how many licenses are available.
- **License server status** - General information about the license server status, such as start time, license counts, rejected and acknowledged requests, average usage time, license refresh and timeout intervals, location of the license key, and the server version.
- **Current running instances** - Lists all currently acknowledged users, including user name, date and time when the license was granted, IP and MAC address of the computer where Oxygen XML Developer Eclipse plugin runs, and lock status.

- **Revoke** - A system administrator can click on the ✕ **Revoke** icon next to a user name to release that particular license and return it to the pool.
- **Unlock** - If a user has locked their license, the system administrator can also unlock it from this page.

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

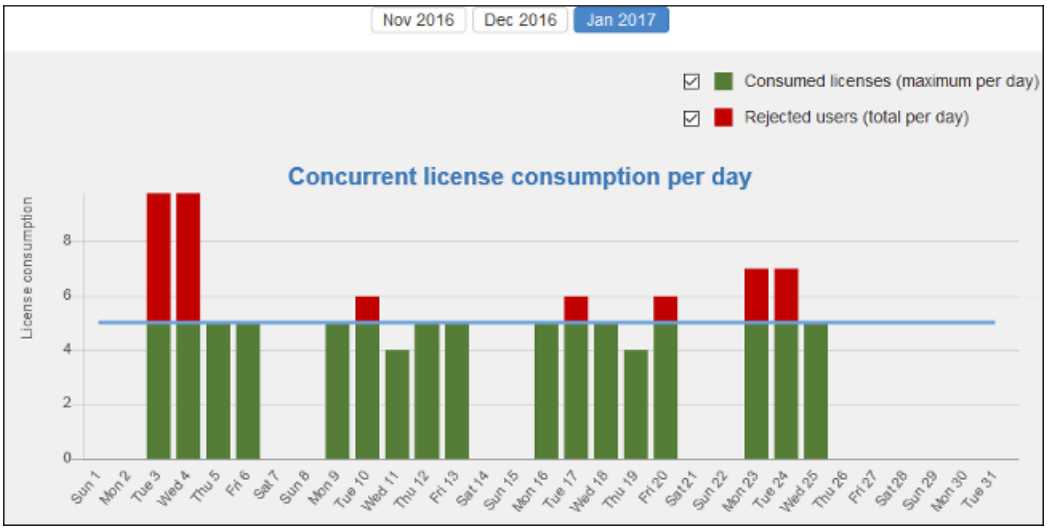
License Usage Statistics Page (Floating License Only)

This report page provides some usage statistics for the floating licenses. It is helpful for determining the number of licenses that are needed and monitoring times when licenses are consumed. It includes the following information:

- **Maximum number of concurrent licenses** - Shows the maximum number of floating licenses that can be consumed at any given time.
- **Concurrent license consumption per day** - A chart that shows the peak number of licenses that were consumed and the total number of users that were rejected, on a daily basis. This chart can be used to detect the amount of concurrent licenses that are needed to avoid having rejected users.

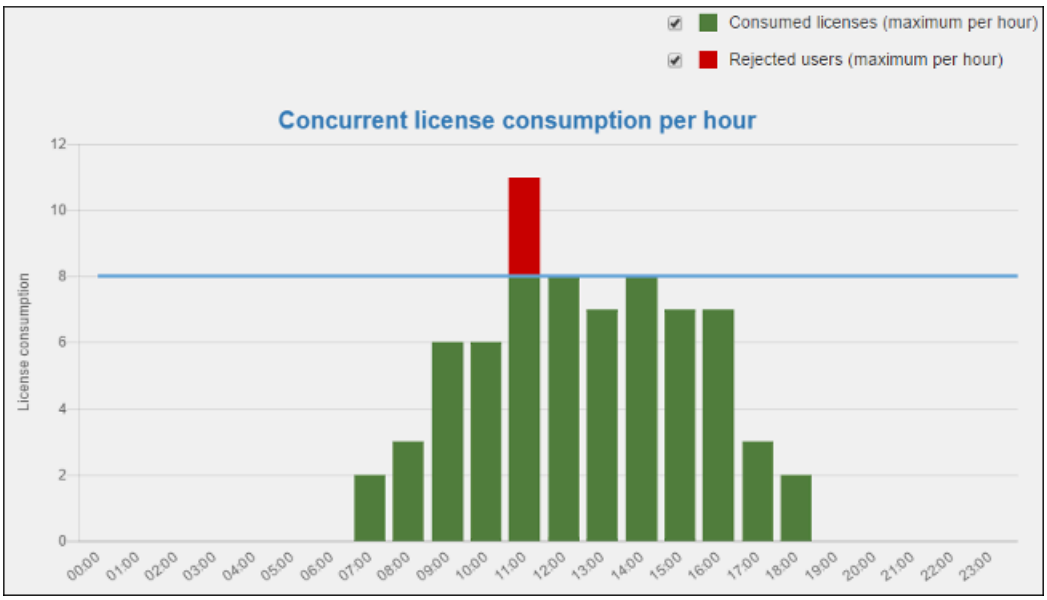
Tip: You can click on any bar to see the license consumption per hour for that particular day.

Figure 3. Concurrent License Consumption per Day Chart



- **Concurrent license consumption per hour** - A chart that shows the peak number of licenses that were consumed per hour throughout that particular month. This is useful for identifying the time of day when the most licenses were consumed.

Figure 4. Concurrent License Consumption per Hour Chart



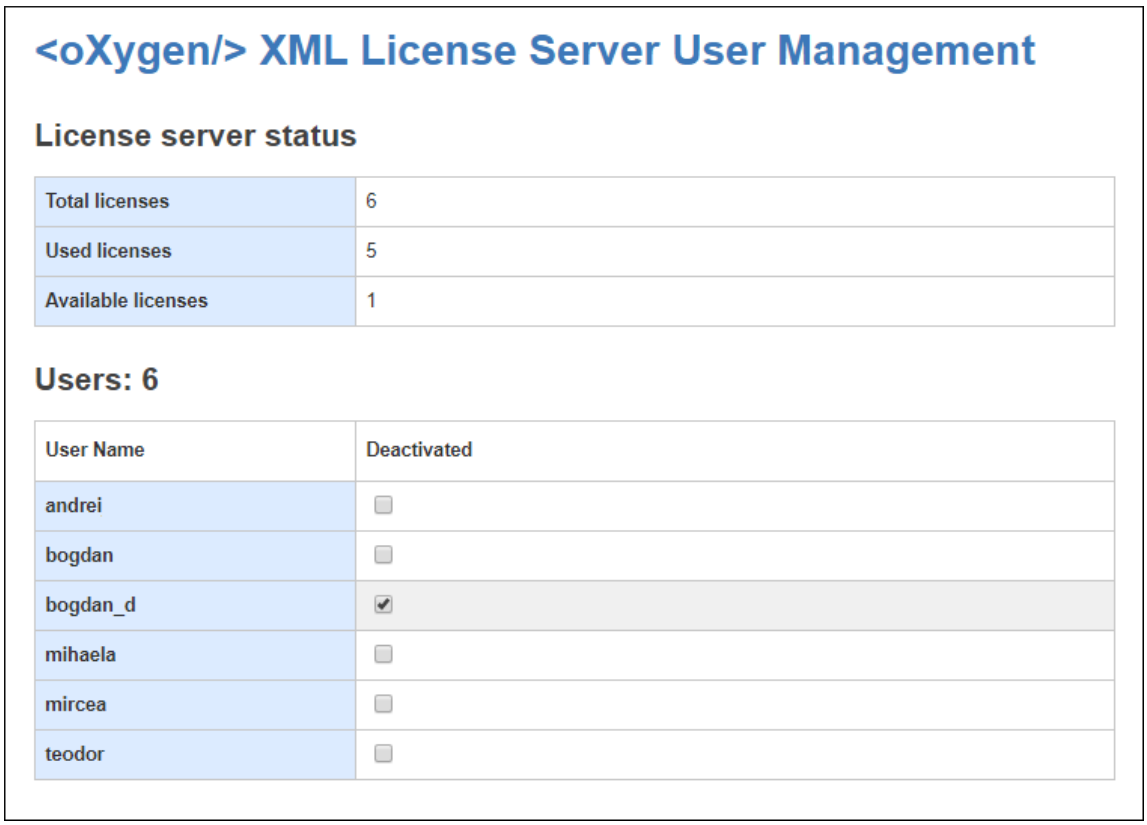
Users Management Page (Named-User License Only)

When using a named-user license key, the license server allocates available licenses on a *first come first served* basis until the maximum number is reached. Further users attempting to get a license key will be rejected.

This page provides access to the list of registered users and allows the server admin to:

- Revoke a user's right to use a license.
- Reactivate a previously deactivated user.

Figure 5. Users List Management Page



Replacing or Removing a License Key in an HTTP License Server

The following procedure assumes that your Oxygen XML Developer Eclipse plugin HTTP license server contains a previously activated license key *(on page 21)* and provides instructions for replacing it with another one or removing it completely.

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 *(on page 19)*.

Replacing a License Key

To replace a license key that is activated on your HTTP 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 using your **admin** credentials.
2. Click the **Replace/Remove license key** link. This will open a page that contains details about the license currently in use.
3. Click the **Replace** button to begin the replacement procedure.
4. Paste the new license key in the displayed form, then click **Register/Activate**. The browser used in the process needs to have Internet access.

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

Note: If you cannot access the online activation form, you can manually activate the license key *(on page 22)*.

5. Click **Register/Activate**.

Result: 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.

Removing a License Key

To remove a license key that is activated on your HTTP license server, follow these steps:

1. Access the license server by following the link provided by the Tomcat Web Application Manager page using your **admin** credentials.
2. Click the **Replace/Remove license key** link. This will open a page that contains details about the license currently in use.
3. Click the **Remove** button to begin the license deletion procedure.
4. Click the **Remove** button in the confirmation page.

Important: The removal process is irreversible. Once the process is complete, you cannot restore the license key.

Upgrading Your HTTP License Server

The goal of the following procedure is to help you minimize the downtime when you upgrade the HTTP License Server to its latest version.

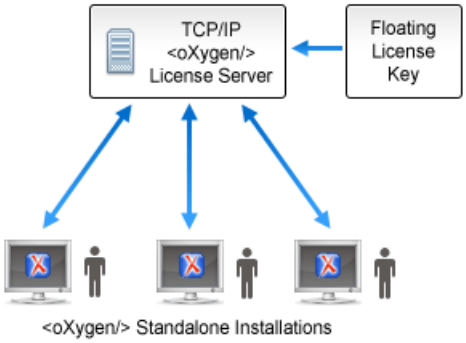
Follow this procedure:

1. Access the license server by following the link provided by the Tomcat Web Application Manager page. If prompted for authentication, use the **admin** 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 **admin** role, and *Undeploy* the license server.
4. Download the Web Archive (WAR) distribution of 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** user.
7. Paste the license key into the form and register it.

Setting up TCP Floating License Server (Deprecated) in 32-bit Windows

This section contains information about setting up the TCP floating license server as a Windows process.

Figure 6. TCP Floating License Server (Process in Windows)



Installation Steps

1. Download the license server installation kit for Windows from the Oxygen XML Developer Eclipse 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.

   Tip: 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.

   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 Eclipse 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. If you have multiple license keys for the same Oxygen XML Developer Eclipse 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, contact the Oxygen support team and ask for a new license key.

Replacing Floating License Key in TCP Floating License Server (Deprecated)

The following procedure assumes that your Oxygen XML Developer Eclipse plugin TCP floating license server contains a previously activated license key *(on page 21)* 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 *(on page 19)*.

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 *(on page 33)*.
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.

Upgrading Your TCP Floating License Server (Deprecated)

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

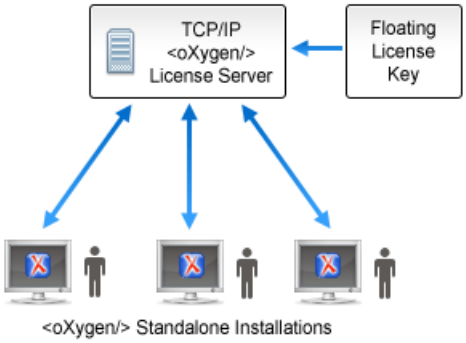
Follow this procedure:

1. Go to the Oxygen XML Developer Eclipse plugin website and download the latest floating license server.
2. Run the installation kit.
3. Leave the default **Update the existing installation** option selected. 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.

Setting up TCP Floating License Server (Deprecated) All-Platforms

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 7. 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 Eclipse 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 Eclipse plugin instances. The default value is **12346**.

   Example: 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.

Replacing Floating License Key in TCP Floating License Server (Deprecated)

The following procedure assumes that your Oxygen XML Developer Eclipse plugin TCP floating license server contains a previously activated license key *(on page 21)* 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 *(on page 19)*.

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.

Upgrading Your TCP Floating License Server (Deprecated)

The goal of the following procedure is to help you minimize the downtime generated when you upgrade the Oxygen XML Developer Eclipse 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 the example above, it is `pathToLicenseDir`) and the port number (in the example above, it 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 Eclipse 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.

Common Problems: License Server Errors

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

Machine Signature Mismatch Error

Problem

I receive an error indicating that *the current license was already activated on a License Server* or that the *License Server's Machine Signature does not match*.

During the license activation process, the license key becomes bound to a particular license server deployment. This means that a code that uniquely identifies your license server deployment (called *Machine Signature*) is sent to the Oxygen XML Developer Eclipse plugin servers, which in turn will sign the license key. The *Machine Signature* is computed from the list of network interfaces of the machine where you deployed the license.

When starting the license server, if you receive an error stating that your server's Machine Signature does not match, there are several possible causes:

Possible Cause 1

The license key was moved to a new machine that hosts your license server.

Solution

Revert to your previous configuration.

Possible Cause 2

A new network interface was changed, added, or activated in the machine that hosts your license server.

Note: A specific example of when this could happen is if the Bluetooth or the WiFi module is activated/deactivated.

Solution

If reverting is not possible, contact the Oxygen support team.

Possible Cause 3

The license server was restarted from a different location as the previous restart. For example, some server configurations will have the Apache Tomcat server installed in a versioned folder (`/usr/local/apache-tomcat-V.V.V`) with a symbolic link to the typical folder (`/usr/local/tomcat`). The server can be restarted from either location, but it is recommended to always restart from the typical folder (`/usr/local/tomcat`) and always restart from the same location.

Solution

The server simply needs to always be restarted from the same location.

Windows Service Reports Incorrect Function Error When Starting (TCP Server)

Problem

My TCP floating license server reports an "Incorrect Function" error message when starting the Windows service.

Cause

This usually appears because the Windows service launcher cannot locate a Java virtual machine on your system.

Solution

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 Process Terminated Unexpectedly Error (TCP Server)

Problem

My TCP floating license server reports a "Process Terminated Unexpectedly" error message for the Windows service.

Cause

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).

Solution

Re-deploy your license key. For more information, see the Setting up a Floating License Server section *(on page 27)*.

Upgrading

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

Upgrading Oxygen XML Developer Eclipse plugin on Windows/Linux

What is Preserved During an Upgrade?

When you install a new version of Oxygen XML Developer Eclipse plugin, some data is preserved and some is overwritten. If there is a previous version of Oxygen XML Developer Eclipse 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 *framework (on page 1325)* files, XSLT stylesheets, *XML Catalogs (on page 1329)*, 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 (on page 1325)* that were stored outside the installation directory (as configured in **Document type associations > Locations** *(on page 51)*) 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 *framework (on page 1325)* files, XSLT stylesheets, *XML Catalogs (on page 1329)*, 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 (on page 1325)* that were stored outside the installation directory (as configured in **Document type associations > Locations** *(on page 51)*) will be preserved and will be found by the new installation.

How to Upgrade Oxygen XML Developer Eclipse plugin on Windows or Linux

1. Uninstall the current version of Oxygen XML Developer Eclipse plugin *(on page 35)*.
2. Download and install the new version according to the instructions for your platform and the type of installer you selected.
3. Restart Oxygen XML Developer Eclipse plugin.
4. 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, you will need to enter a new license for the new version into the registration dialog box that is displayed when the plugin is started.

Upgrading Oxygen XML Developer Eclipse plugin on OS X

What is Preserved During an Upgrade?

When you install a new version of Oxygen XML Developer Eclipse plugin, first you need to remove or rename the old installation directory. By renaming the directory, it can coexist with the new installation and the following data will be preserved:

- All the files from the old install directory will be preserved, including any modification in *framework (on page 1325)* files, XSLT stylesheets, *XML Catalogs (on page 1329)*, 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 (on page 1325)* that were stored outside the installation directory (as configured in **Document type associations > Locations** *(on page 51)*) will be preserved and will be found by the new installation.

How to Upgrade Oxygen XML Developer Eclipse plugin on OS X

1. Uninstall the current version of Oxygen XML Developer Eclipse plugin *(on page 35)* or rename the installation directory (for example, `Oxygen XML Editor.old`).
2. Download and install the new version in an empty folder according to the instructions for your platform and the type of installer you selected.
3. Restart Oxygen XML Developer Eclipse plugin.

4. 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, you will need to enter a new license for the new version into the registration dialog box that is displayed when the plugin is started.

Uninstalling

How to Uninstall Oxygen XML Developer Eclipse plugin

⚠ **CAUTION:** The following procedure will remove Oxygen XML Developer Eclipse 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 Eclipse plugin from the list of plugins.
3. Choose **Uninstall**.
4. Accept the Eclipse restart.
5. If you want to remove the user preferences:
    - **Windows** - Remove the directory: `%APPDATA%\com.oxygenxml.developer` (usually `%APPDATA%` has the value: `[user-home-dir]\Application Data`). Note that this directory is hidden.
    - **OS X** - Remove the directory: `Library/Preferences/com.oxygenxml.developer` of the user home folder.
    - **On Linux**, remove the directory: `.com.oxygenxml.developer` from the user home directory.

4.

Configuring Oxygen XML Developer Eclipse plugin

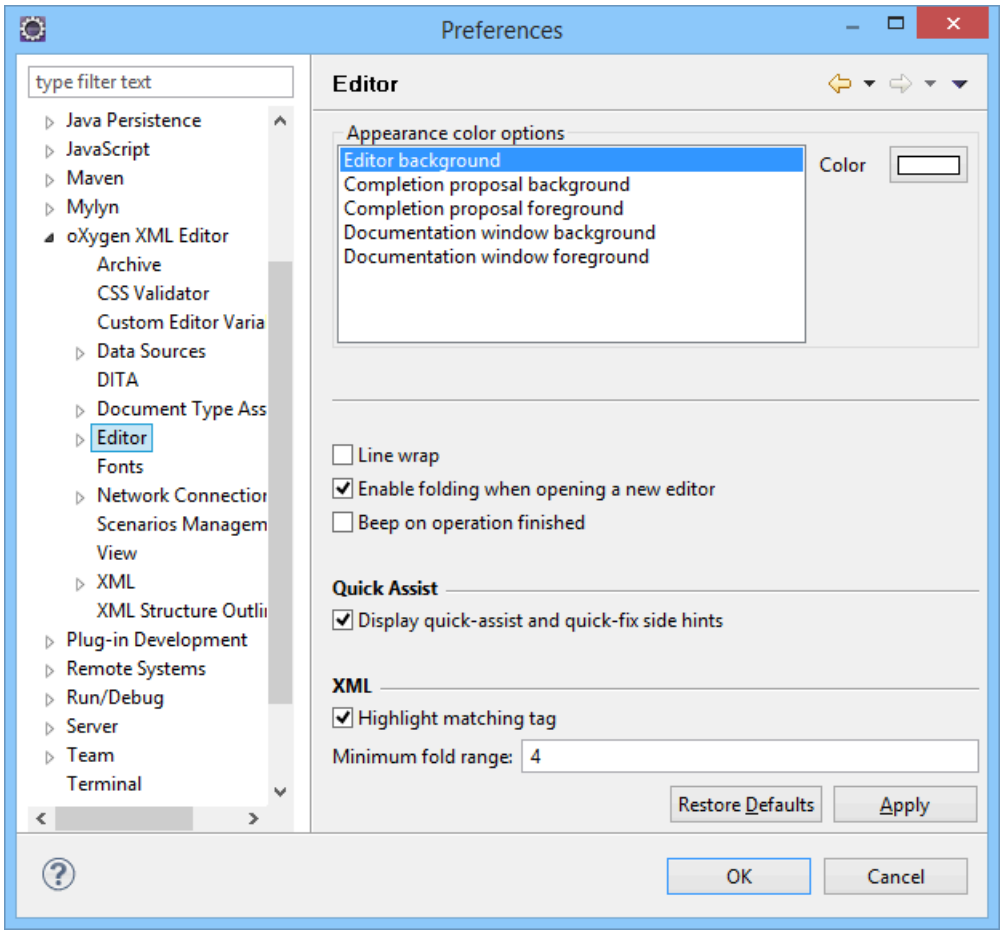
This chapter presents all the user preferences and options that allow you to configure various features and aspects of the application itself. It also includes information about storing and sharing options, importing and exporting options or scenarios, customizing system properties, setting startup parameters, and the editor variables *(on page 148)* that are available for customizing user-defined commands.

Preferences

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

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

Figure 8. Eclipse Preferences Dialog Box

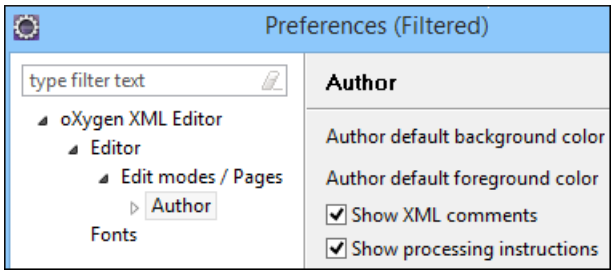


Click the ? icon or press **F1** for help on any preferences page.

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 9. 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.developer`
- **Mac OS X** - `[user_home_directory]/Library/Preferences/com.oxygenxml.developer`
- **Linux/Unix** - `[user_home_directory]/.com.oxygenxml.developer`

Oxygen XML Developer Eclipse plugin License

To configure the license options, open the **Preferences** dialog box *(on page 36)*. This preferences page presents the details of the license key that enables the Oxygen XML Developer Eclipse plugin plugin, such as registration name, category and number of purchased licenses, encrypted signature of the license key. Clicking the **Register** button opens the Oxygen XML Developer Eclipse plugin **License** dialog box that allows you to insert a new license key.

Archive Preferences

To configure *Archive* options, open the **Preferences** dialog box *(on page 36)* 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.

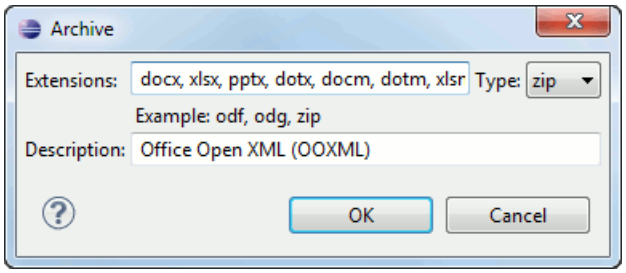
Show archive backup dialog box

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 Eclipse 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 10. Edit Archive Extension Mappings



Important: You have to restart Oxygen XML Developer Eclipse 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 *(on page 36)* and go to **CSS Validator**.

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

- **Profile** - Selects one of the available validation profiles: **CSS 1**, **CSS 2**, **CSS 2.1**, **CSS 3**, **CSS 3 + SVG**, **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 *(on page 522)* 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. The following vendor extensions are specified as ignored by default: **-ro-*** (PDFreactor), **-ah-*** (Antenna House), **prince-*** (Prince). As wildcards you can use:
  - ***** to match any string.
  - **?** to match any character.
- **Recognize browser CSS extensions (also applies to content completion)** - If selected, Oxygen XML Developer Eclipse plugin recognizes browser-specific CSS properties (no validation is performed). The *Content Completion Assistant (on page 1324)* 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*.

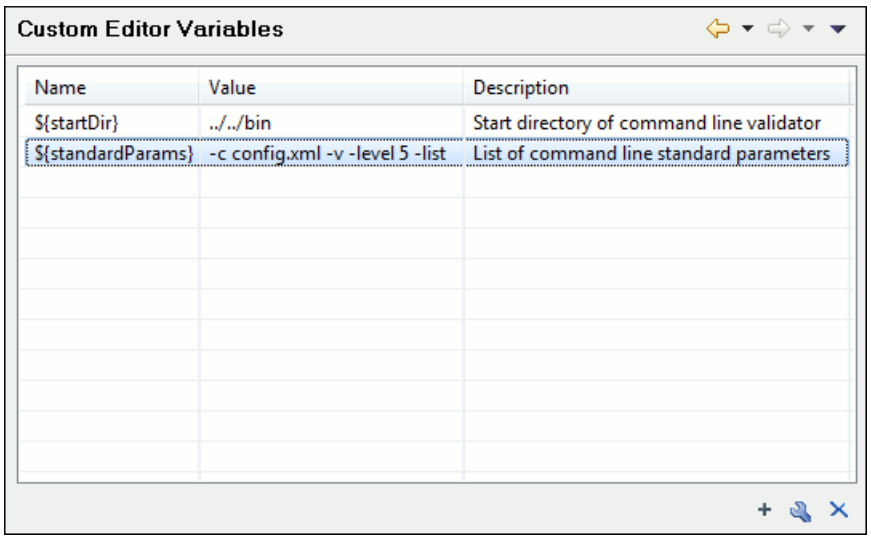
Custom Editor Variables Preferences

An editor variable *(on page 148)* 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 *(on page 148)*.

Custom editor variables are created and configured in the **Custom Editor Variables** preferences page. To access this page, open the **Preferences** dialog box *(on page 36)* 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**. The 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.

Figure 11. Custom Editor Variables Table



Data Sources Preferences

To configure the **Data Sources** preferences, open the **Preferences** dialog box *(on page 36)* 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 45)*.

Connection Wizards Section

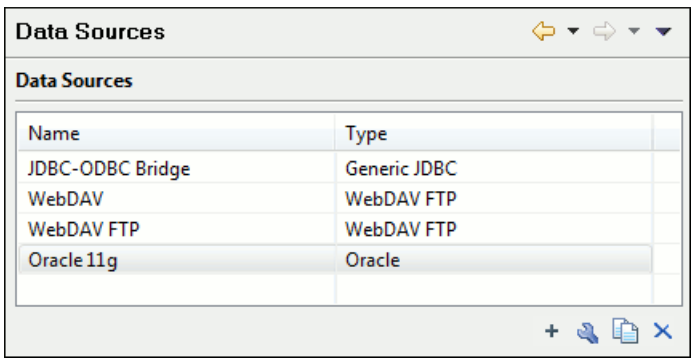
Create eXist-db XML connection

Click this link to open the dedicated **Create eXist-db XML connection** dialog box *(on page 1160)* that provides a quick way to create an eXist connection.

Data Sources Section

This section allows you to add and configure data sources.

Figure 12. Data Sources Preferences Panel

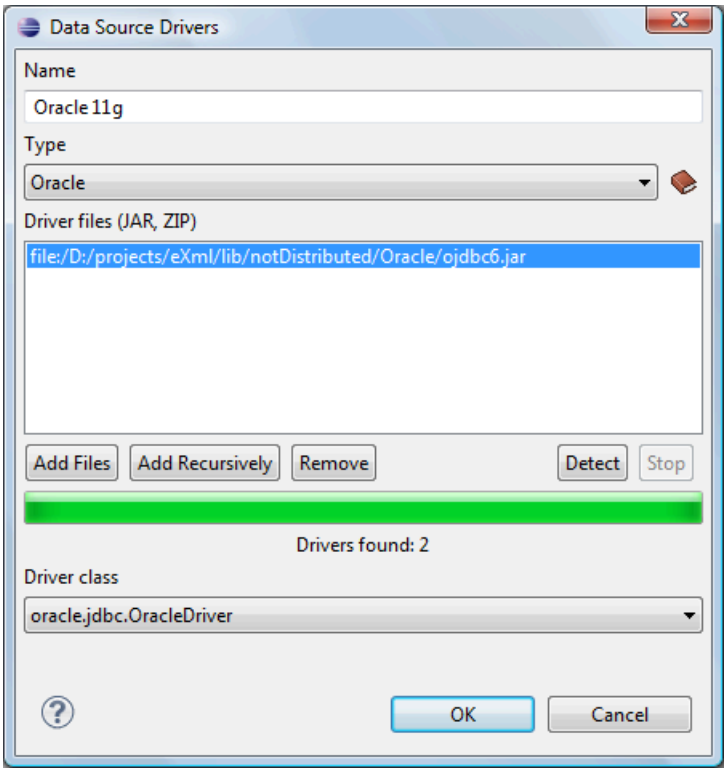


The following buttons are available at the bottom of the **Data Sources** panel:

- **New**

Opens the **Data Sources Drivers** dialog box that allows you to configure a new database driver.

Figure 13. 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 *(on page 45)* 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 *(on page 45)* that are necessary for accessing databases in Oxygen XML Developer Eclipse 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.

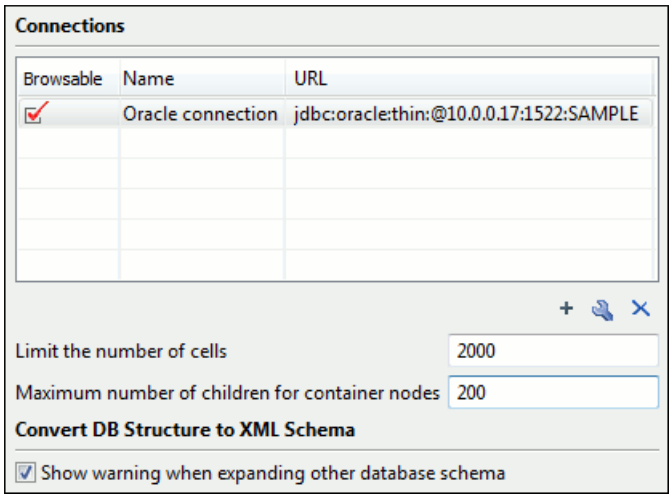
Delete

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 14. Connections Preferences Panel

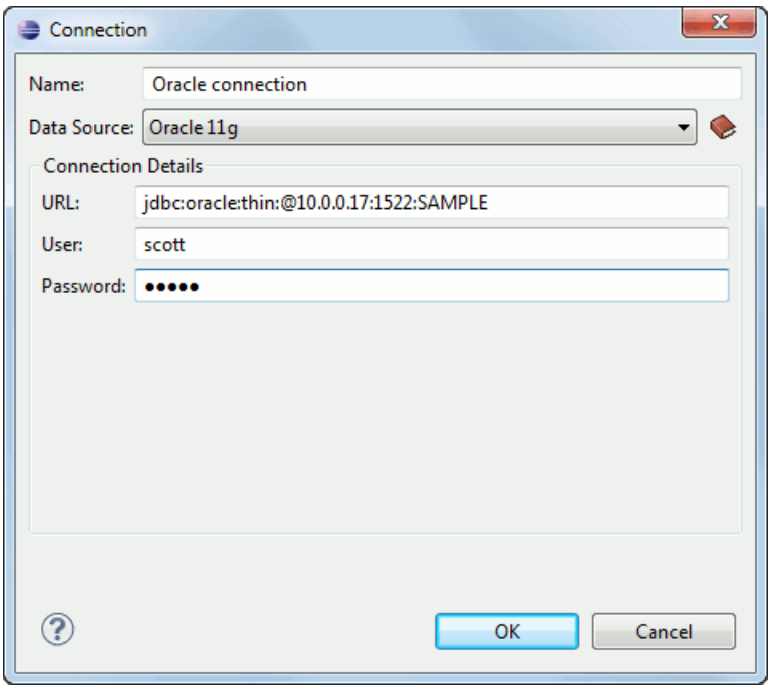


The following buttons and options are available at the bottom of the **Connections** panel:

New

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

Figure 15. 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 *(on page 1133)* 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 *(on page 1133)*, 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 *(on page 1131)* 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. You can navigate to other contained resources from the same container by using the *Up* and *Down* buttons in the **Data Source Explorer** view *(on page 1131)*. 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 *(on page 1203)* and the **Select database table** section of the **Convert DB Structure to XML Schema** dialog box *(on page 478)*.

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 *(on page 1131)*. To open this preferences page, open the **Preferences** dialog box *(on page 36)* 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 Eclipse plugin:

- **Berkeley DB XML database** - Copy the *jar* files from the Berkeley database install directory into the Oxygen XML Developer Eclipse plugin install directory as described in the procedure for configuring a Berkeley DB data source *(on page 1154)*.
- **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 Eclipse plugin for configuring a DB2 data source *(on page 1137)*.
- **eXist database** - Copy the *jar* files from the eXist database install directory to the Oxygen XML Developer Eclipse plugin install directory as described in the procedure for configuring an eXist data source *(on page 1160)*.
- **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.
- **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.

DIFF Preferences

To access the **DIFF** preferences page, go to **Window** (**Eclipse** on Mac OSX) and choose **Preferences > Oxygen XML Developer Eclipse plugin > DIFF**. This preferences page includes the following sections and options:

Enable file comparison in Author mode

When selected, the visual comparison mode is available when comparing XML files. If not selected, the built-in Eclipse text mode comparison will always be used.

Two-Way Diff section

Default algorithm

The default algorithm used for comparing two files. The following options are available:

- **Auto** - Selects the most appropriate algorithm, based on the compared content and its size (selected by default).
- **XML Fast** - Comparison that works well on large files or fragments, but it is less precise than **XML Accurate**.
- **XML Accurate** - Comparison that is more precise than **XML Fast**, at the expense of speed. It compares two XML files or fragments looking for identical XML nodes.

Algorithm strength

Controls the amount of resources allocated to the application to perform the comparison. The algorithm stops searching more differences when reaches the maximum allowed resources. A dialog box is displayed when this limit is reached and partial results are displayed. Four settings are available: **Low**, **Medium** (default), **High** and **Very High**.

Three-Way Diff section

Default algorithm

The default algorithm used for performing a three-way comparison. The following options are available:

- **Auto** - Selects the most appropriate algorithm, based on the compared content and its size (selected by default).
- **XML Fast** - Comparison that works well on large files or fragments, but it is less precise than **XML Accurate**.
- **XML Accurate** - Comparison that is more precise than **XML Fast**, at the expense of speed. It compares two XML files or fragments looking for identical XML nodes.

Algorithm strength

Controls the amount of resources allocated to the application to perform the comparison. The algorithm stops searching more differences when reaches the maximum allowed resources. A dialog box is displayed when this limit is reached and partial results are displayed. Four settings are available: **Low**, **Medium** (default), **High** and **Very High**.

Show pseudo conflicts

Specifies whether or not the file comparison displays pseudo-conflicts. A pseudo-conflict occurs when two users make the same change (for example, when they both add or remove the same line of code).

XML Diff section

In this section, you can choose to ignore **Namespaces**, **Prefixes**, **Namespace declarations**, and the **Attribute order**.

Appearance Preferences

To configure the appearance options for the visual file comparison tool, go to **Window** (**Eclipse** on Mac OSX) and choose **Preferences > Oxygen XML Developer Eclipse plugin > DIFF > Appearance**. This preferences page includes the following sections and options:

Incoming color

Specifies the color used on the vertical bar for incoming changes.

Outgoing color

Specifies the color used on the vertical bar for outgoing changes.

Conflict color

Specifies the color used on the vertical bar for conflicts between the compared files.

DITA Preferences

To access the DITA Preferences page, open the **Preferences** dialog box *(on page 36)* 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 Eclipse plugin installation) to be used for validating and publishing DITA content. You can select from the following:

Built-in DITA-OT 3.x

Oxygen XML Developer Eclipse plugin comes bundled with DITA-OT 3.4.1. By default, all defined DITA transformation/validation scenarios will run with this version. This also provides access to DITA 1.3 features (including document templates). The default DITA-OT 3.4.1 directory is: *[OXYGEN_INSTALL_DIR]*`/frameworks/dita/`*`DITA-OT3.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** *(on page 148)* button, or the **Browse** button.

Important: Using a custom DITA Open Toolkit may disable certain features in the application. Examples of features that may not work properly:

- If the custom DITA-OT is missing certain publishing plugins, default transformation scenarios such as DITA Map WebHelp Responsive or DITA Map PDF - based on HTML5 & CSS may no longer work properly.
- Validation of Markdown documents using Schematron may not work because it is based on a certain DITA Open Toolkit plugin.
- The DITA framework (defined in the **Preferences > Document Type Associations** page) will use the XML catalogs specified in the DITA-OT configured in the **Preferences > DITA** page to perform the validation of all DITA topic types. If this DITA-OT is different from the one that comes bundled with the Oxygen XML Developer Eclipse plugin default distribution, you might encounter validation-related issues.

CAUTION: Oxygen XML Developer Eclipse plugin support engineers cannot offer support and troubleshooting assistance for a custom DITA-OT. If you discover any issues or inconsistent behavior while using a custom DITA-OT, you should revert to the default built-in DITA-OT.

DITA New Topics Preferences

To access the DITA New Topics preferences page, open the **Preferences** dialog box *(on page 36)* and go to **DITA > New Topics**. This preferences page includes the following options:

Use the title to generate the file name

This option (and its sub-options) pertain to the rules that will be used to generate file names in the . Select this option to use the text entered in the **Title** field to automatically generate a file name (the generated name can be seen in the **Save as** field). By default, the generated name will replace spaces with underscores (_), all illegal characters will be removed, and all upper case characters changed to lower case, but you can use the sub-options to change this.

Replace spaces with

If selected, the file name generation mechanism will replace spaces in the title with the character entered in this option.

Lower case only

If selected, the file name generation mechanism will only use lower case letters.

Use camel case

If selected, the file name generation mechanism will convert the title to a file name using the *camel case* convention where the first word starts with a lower case letter and all subsequent words begin with upper case (for example, `myFileName`).

Upper case first letter

Select this option if you want the file name generation mechanism to convert the title to a file name using the *camel case* convention but with an upper case letter for the first word (for example, `MyFileName`).

Use the file name as the value of the root ID attribute

If selected, when creating a new topic, the file name (as seen in the **Save as** field but without the file extension) will be used as the value of the root `@id` attribute for the new topic.

Use the file name as the value of the "keys" attribute for topic references

If selected, when inserting a topic reference into a map, the file name will be used as the value of the `@keys` attribute for the new `<topicref>`.

Note: This option also has an effect on image references. When inserting a reference to an image in a DITA map and this option is selected, a `<keydef>` element is created if it is allowed by the schema. If it is not allowed (or this option is deselected), a specific topic reference element is created with the value of the `@processing-role` attribute set to **resource-only**.

DITA Publishing Preferences

To access the DITA Publishing preferences page, open the **Preferences** dialog box *(on page 36)* and go to **DITA > Publishing**. You can also open this page by clicking the **Configure Publishing Templates Gallery** link in the **Templates** tab of the transformation scenario dialog box for **WebHelp Responsive** transformations.

You can use this preferences page to specify additional directories where custom publishing templates are stored. The templates stored in these directories will appear in the preview pane in the **Templates** tab of the transformation scenario dialog box, along with all the built-in templates.

DITA Logging Preferences

To access the DITA Logging preferences page, open the **Preferences** dialog box *(on page 36)* and go to **DITA > Logging**. This preferences page includes the following sections and options:

Show console output

Allows you to specify when to display the console output log in the message panel at the bottom of the editor. The following options are available:

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

Show the following types of messages in a new tab

This section allows you to specify which types of messages will be displayed in separate tabs in the message panel at the bottom of the editor if a DITA transformation results in errors or warnings. You can choose whether or not to display the following types of messages:

- DITA-OT errors

- DITA-OT warnings
- DITA-OT info
- FOP errors
- FOP warnings
- FOP info
- XSLT problems

Document Type Association Preferences

Oxygen XML Developer Eclipse plugin uses *document type associations (on page 1324)* to associate a document type *(on page 646)* with a set of functionality provided by a *framework (on page 1325)*. To configure the **Document Type Association** options, open the **Preferences** dialog box *(on page 36)* 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 *frameworks (on page 1325)* (*document type associations (on page 1324)*), sorted by priority and alphabetically. Each edited document type has a set of association rules *(on page 53)* (used by the application to detect the proper document type association to use for an open XML document).

New

Opens a **Document type** configuration dialog box *(on page 52)* that allows you to add a new *framework*.

Edit

Opens a **Document type** configuration dialog box *(on page 52)* that allows you to edit an existing *framework*.

Note: If you try to edit an existing *framework* when you do not have write permissions to its storage location, a dialog box will be shown asking if you want to extend it.

Duplicate

Opens a **Document type** configuration dialog box *(on page 52)* that allows you to duplicate the configuration of an existing *framework*. 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 as the **Extend** action does.

Extend

Opens a **Document type** configuration dialog box *(on page 52)* that allows you to extend an existing *framework*. 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 *framework* (document type).

Enable DTD/XML Schema processing in document type detection

When this option is selected (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 Eclipse plugin to correctly match the DITA customization.

Only for local DTDs/XML Schemas

When this option is selected (default value), only the local DTDs / XML Schemas will be processed.

Enable DTD/XML Schema caching

When this option is selected (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 Eclipse plugin allows you to change the location where *frameworks (on page 1325)* (document types) 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 (*framework*) can be loaded from the following locations:

- **Internal preferences** - The document type configuration is stored in the application **Internal** preferences *(on page 53)*.
- **Additional *framework* 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.

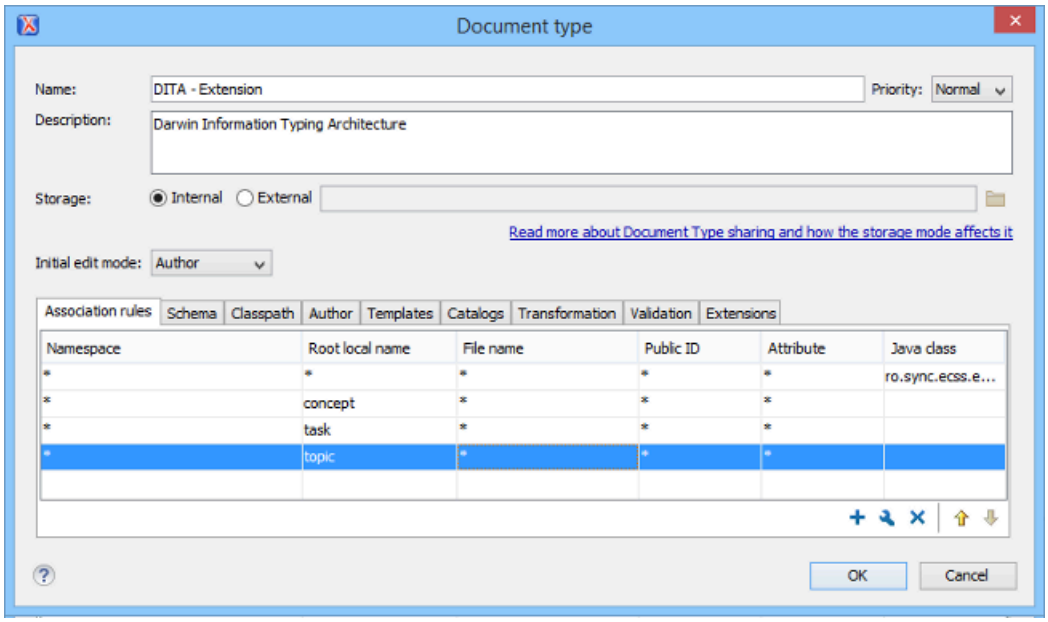
All loaded document type configurations are first sorted by priority, then by document type name and then by load location (in the exact order specified above). When an XML document is opened, the application chooses the first document type configuration from the sorted list that matches the specific 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 *framework (on page 1325)* (document type). It is displayed when you use the **New**, **Edit**, **Duplicate**, or **Extend** buttons in the **Document Type Association** preferences page *(on page 50)* (open the **Preferences** dialog box *(on page 36)* and go to **Document Type Association**).

Figure 16. Document Type Configuration Dialog Box



The configuration dialog box includes the following fields and sections:

Name

The name of the *framework*. This will be displayed as its name in the **Document Type** column in the **Document Type Association** preferences page *(on page 50)*.

Priority

Depending on the priority level, Oxygen XML Developer Eclipse plugin establishes the order that the existing *frameworks* 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 for *frameworks* you want to be evaluated first.

Note: The built-in document types are set to Low priority by default. *Frameworks* that have the same priority are sorted alphabetically.

Description

The document type description displayed as a tooltip in the **Document Type Association** preferences page *(on page 50)*.

Storage

The location where the framework is saved. If you select the **External** storage option, the framework is saved in a specified file with a mandatory extension (located in a subdirectory of your current framework directory. If you select the **Internal** storage option, the framework configuration data is saved in the Oxygen XML Developer Eclipse plugin internal options file.

Initial edit mode

Sets the default edit mode when you open a document for the first time: **Editor specific**, **Text**, **Grid** and **Design** (available only for the W3C XML Schema editor). If the **Editor specific** option is selected, the initial editing mode is determined based upon the editor type. You can find the mapping between editors and edit modes in the **Edit modes** preferences page. *(on page 85)* You can impose an initial mode for opening files that match the association rules of the document type. For example, if the files are usually edited in the **Author** mode you can set it in the **Initial edit mode** combo box.

Note: You can also customize the initial mode for a document type in the **Edit modes** preferences page. Open the **Preferences** dialog box *(on page 36)* and go to **Editor > Edit modes**.

Configuration Tabs

The bottom section of the dialog box includes various tabs where you can configure numerous options for the *framework*.

Association Rules Tab

To open the **Association Rules** tab of the **Document type** configuration dialog box, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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.

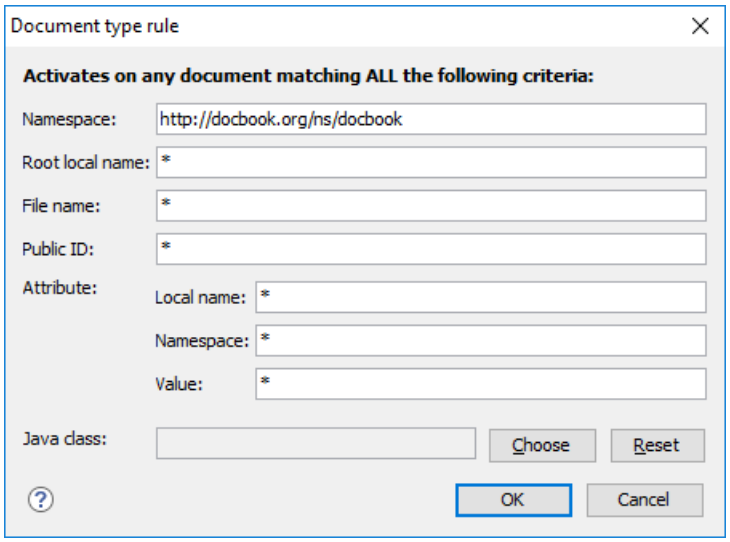
Move Down

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

By combining multiple association rules you can instruct Oxygen XML Developer Eclipse plugin to identify the type of a document. Oxygen XML Developer Eclipse plugin identifies the type of a document when the document matches at least one of the *association rules*. This tab gives 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 create a new association rule, click the **New** button at the bottom of the **Association Rules** tab, or to edit an existing rule, click the **Edit** button.

Figure 17. Document Type Rule Dialog Box



The **Document type rule** dialog box includes the following fields and options:

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.

Attribute Local name

Specifies the local name of the attributes for the root element (* *(any)* by default).

Attribute Namespace

Specifies the namespace of the attributes for the root element (* *(any)* by default).

Attribute Value

Specifies the value of the attributes for the root element (* *(any)* by default).

Java class

Presents the name of the Java class that is used to determine if a document matches the rule. This Java class should implement the `ro.sync.ecss.extensions.api.DocumentTypeCustomRuleMatcher` interface.

Tip: You can use wildcards (*?* and *) or editor variables *(on page 148)* in the **Document Type Rule** dialog box, and you can enter multiple values by separating them with a comma.

Schema Tab

In the **Schema** tab, you can specify a default schema for Oxygen XML Developer Eclipse plugin to use 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 *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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** *(on page 148)* button, or the browsing actions 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 *(on page 153)* 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 (on page 1326)* libraries that hold implementations for API extensions, implementations for custom **Author** mode operations, various resources (such as stylesheets), and *framework (on page 1325)* translation files. Oxygen XML Developer Eclipse 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 *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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** *(on page 148)* button, or the browsing actions 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** *(on page 148)* button, or the browsing actions 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 76)*

Author Tab *(on page 56)*

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 (on page 1325)*-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 *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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 Eclipse 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 *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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** *(on page 148)* button, or the browsing actions 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 selected 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 of the **Document Type Configuration** dialog box contains a sortable table with all the **Author** mode actions that are configured for the specific *framework (on page 1325)*. Each action has a unique ID, a name, a description, and a shortcut key.

To open the **Actions** subtab, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, select your framework, use the **Duplicate** or **Extend** button to create an extension of the framework (or the **Edit** button for an already extended framework), click on the **Author** tab, and then the **Actions** subtab.

The following features are available in this subtab:

Export existing actions ( )

It is possible to export existing actions to use them in other frameworks. Each exported action is extracted from the framework configuration file and exported as an individual XML file.

To export actions, the **Storage** option *(on page 53)* in the top part of the **Document Type Configuration** dialog box must be set to **External** and the external location must be a subdirectory of your current framework directory.

The **Export** action is found by right-clicking an action or a selection of multiple actions (the **Export** button is also located below the table of actions). If you choose to export a single action, a resulting dialog box will allow you to select the destination path for the new XML file that contains the configuration details of the action. If you export multiple actions, they will automatically be saved as individual XML files inside a newly created folder (it will have ***_externalAuthorActions*** at the end of the folder name) inside your current framework directory.

Result: Exported actions will display the icon in the first column in the table.

Important: The newly created files for the exported actions will not appear on disk until you click **OK** several times to confirm your changes and exit the **Preferences** dialog box.

Tip: If you want to create a new XML file for an action, there is a document template called **Author Actions** in the **New from templates** wizard *(on page 177)* to help you get started.

Note: You can add or edit the action files outside of Oxygen XML Developer Eclipse plugin, but you will need to restart the application each time to reload the changes.

Open in editor ( )

For exported actions, there is a **Open in editor** action in the contextual menu that will open the XML file for that action in the main editor.

Create a new action ( )

Use the **New** button (located underneath the table of actions) to open the **Action** dialog box *(on page 59)* where you can configure a new action.

Duplicate an existing action ( )

Use the **Duplicate** action (found in the contextual menu and underneath the table of actions) to duplicate the selected action.

Edit an existing action ( )

Use the **Edit** button (found in the contextual menu and underneath the table of actions) to open the **Action** dialog box *(on page 59)* where you can edit the selected action.

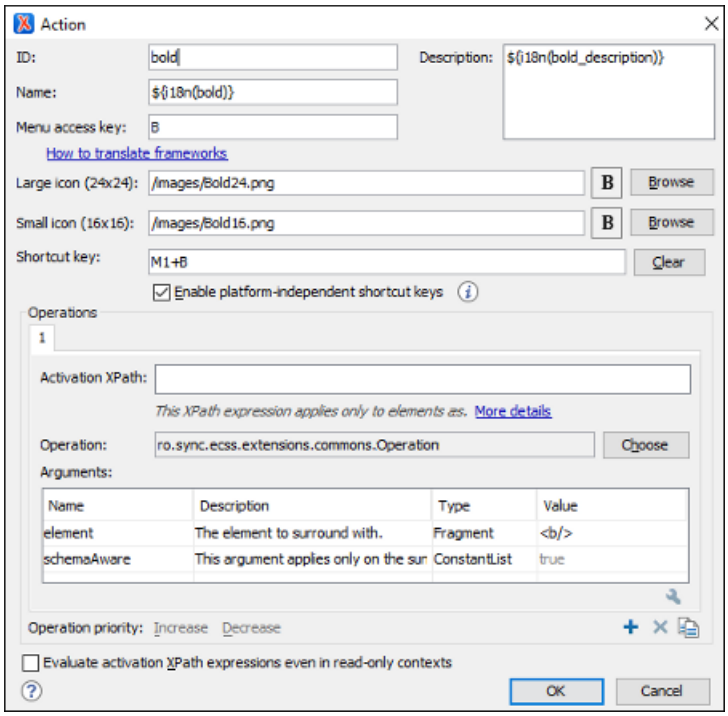
Delete an existing action ( )

Use the **Delete** button (found in the contextual menu and underneath the table of actions) to delete the selected action.

Author Action Dialog Box

To edit an existing document type action or create a new one, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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.

Figure 18. 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 *(on page 153)* 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 *(on page 153)* 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 Eclipse 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 *(on page 153)* 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 Eclipse 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 selected, 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 selected 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: Controlling Which Author Operations Gets Executed Through XPath Expressions *(on page 62)*.

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: Controlling Which Author Operations Gets Executed Through XPath Expressions *(on page 62)*.

Operation

Specifies the invoked operation that can be a default operation or a custom 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 selected, the action can be invoked even when the cursor is placed in a read-only location.

Controlling Which Author Operations Gets Executed Through XPath Expressions

An **Author** mode action can have multiple operation modes, each one invoking an Author operation with certain configured parameters. Each operation mode has an XPath 2.0 expression for activating it.

For each operation mode of an action, the application will check if the XPath expression is fulfilled (when it returns a non-empty node set or a **true** result). Only the first operation whose XPath operation is fulfilled will be executed.

The following special XPath extension functions are provided:

- *oxy:allows-child-element() (on page 62)* - 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() (on page 64)* - Use this function to check whether or not an element is a valid global element for the current *framework (on page 1325)*, according to the associated schema.
- *oxy:current-selected-element() (on page 65)* - Use this function to get the currently selected element.
- *oxy:selected-elements() (on page 65)* - Use this function to get the selected elements.
- *oxy:is-required-element() (on page 65)* - Use this function to check if the element returned by the given XPath expression is required (based on the rules declared in the schema).
- *oxy:platform() (on page 66)* - Use this function to get the current platform in cases where you want to enable or disable an action depending on the platform. Possible values include: *eclipse*, *standalone* and *webapp*.

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 previously 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 to 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 (on page 1325)*, 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 previously 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 to 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.

Example: *oxy:current-selected-element* Function

```
oxy:current-selected-element()[self::p]/b
```

This example returns the `<b>` elements that are children of the currently selected `<p>` element.

oxy:selected-elements() Function

This function returns the selected elements from **Author** mode.

Example: *oxy:selected-elements* Function

```
oxy:selected-elements()[self::para][@audience="novice"]
```

This example would activate an action when at least one of the selected elements is a `<para>` element with the `@novice` attribute defined.

oxy:is-required-element() Function

This function checks if the element returned by the given XPath expression is required (based on the rules declared in the schema). It has only one argument, an XPath expression, and the XPath expression must be written in such a way that it returns a single element.

Example: *oxy:is-required-element* Function

```
oxy:is-required-element(.)
```

This example would check to see if the current element is required by the schema.

oxy:is-editable-element() Function

This function checks if the element returned by the given XPath expression is editable (content can be inserted in it), meaning both that the entire XML file is editable and that the current context where the element is placed is editable. For example, if the element is inside an *xi:included* section, it is not editable.

It only has one argument, an XPath expression, and the XPath expression must be written in such a way that it returns a single element.

Example: *oxy:is-editable-element* Function

```
oxy:is-editable-element(ancestor-or-self::table)
```

This example would return *true* if the cursor is placed inside a table and it is editable or *false* if it is not editable.

oxy:platform() Function

This function returns the current platform. You can use this if you want to enable or disable an action depending on the platform. The possible values are: **standalone**, **eclipse**, or **webapp**.

Example: *oxy:platform* Function

```
oxy:platform()="standalone"
```

This example would keep the action activated for the *standalone* distribution of Oxygen XML Developer Eclipse plugin, but disable it for the *Eclipse* and *Web Author* distributions.

Menu Subtab

In the **Menu** subtab, you can configure which actions will appear in the *framework (on page 1325)*-specific menu. The subtab is divided into two sections: **Available actions** and **Current actions**.

To open the **Menu** subtab, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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 Eclipse 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

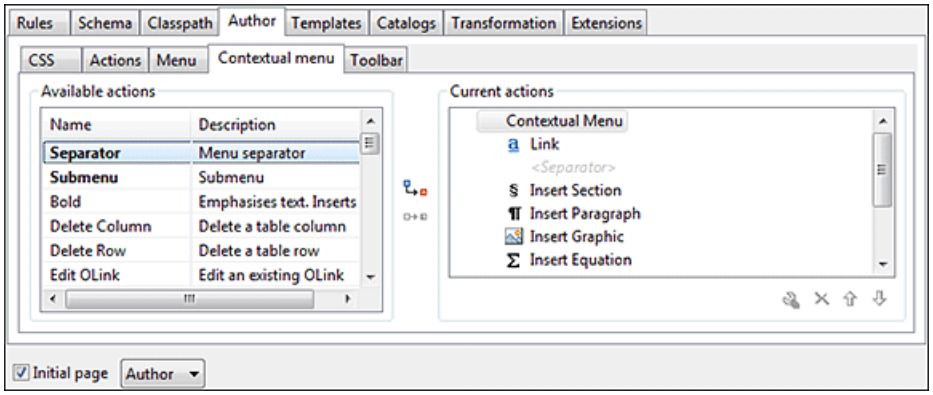
Moves an item down.

Contextual Menu Subtab

In the **Contextual menu** subtab you configure what *framework (on page 1325)*-specific action the *Content Completion Assistant (on page 1324)* proposes. The subtab is divided into two sections: **Available actions** and **Current actions**.

To open the **Contextual Menu** subtab, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, click on the **Author** tab, and then the **Contextual Menu** subtab.

Figure 19. 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.

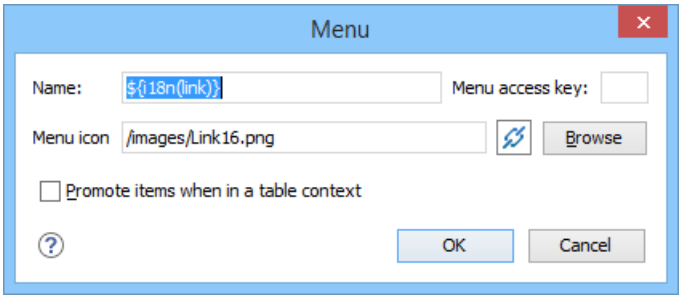
Add as child

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 20. 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 *(on page 153)* 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 Eclipse plugin uses for the container (submenu).

Promote items when in a table context

If this option is selected, 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 (on page 1325)*-specific action the Oxygen XML Developer Eclipse plugin toolbar holds. The subtab is divided into two sections: **Available actions** and **Current actions**.

To open the **Toolbar** subtab, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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 Eclipse 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 (on page 1325)*-specific the *Content Completion Assistant (on page 1324)* proposes. The subtab is divided into two sections: **Available actions** and **Current actions**.

To open the **Content Completion** subtab, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, click on the **Author** tab, and then the **Content Completion** subtab.

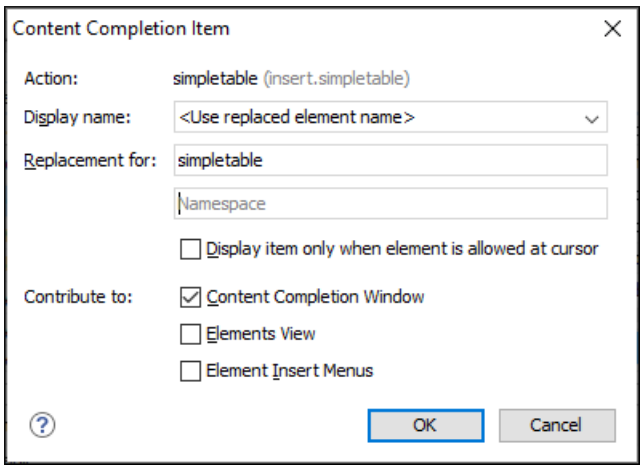
Available and Current Actions

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 that belongs to the edited *framework*.

To add the selected available action as a sibling of the currently selected action in the **Current actions** section, use the **Add as sibling** button. To add it as a child of the currently selected action, use the **Add as child** button. To edit an existing action, select it and use the **Edit** button. To remove an existing action, use the **Remove** button. You can also move items up and down the list using the **Move Up** or **Move Down** buttons.

Adding an action (or editing an existing one) opens the **Content Completion Item** dialog box.

Figure 21. Content Completion Item Dialog Box



Use this dialog box to configure the action:

Action

Displays the name of the selected action.

Display name

You can use the drop-down menu to choose between displaying the action name or the replaced element name, or you can enter another name to be displayed.

Replacement for

Use this section to specify an element to be replaced by the configured action. The first field is the name of the element name to be replaced. You can use the next field to specify the namespace for the element. Also, if the **Display item only when element is allowed at cursor** option is selected, the configured item will not be displayed in any of the UI components selected in the **Contribute to** section unless the associated schema allows the action at the current location in the document.

Contribute to

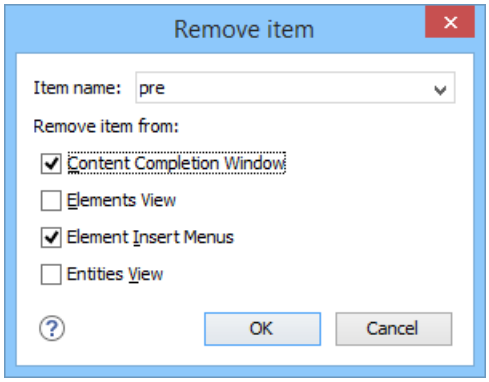
Use this section to specify where to display the configured item in the interface:

- **Content Completion Window** - The configured item will appear in the *Content Completion Assistant (on page 1324)*.
- **Elements View** - The configured item will appear in the **Elements** view *(on page 247)*.
- **Element Insert Menus** - The configured item will 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 *(on page 240)*).

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 22. 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> [elementName]** - Filters split entries for elements that have the form **Split elementName** or **New elementName**.
- **<SPLIT>** - Filters split entries for all elements.
- **<ENTER>** - Filters **Insert New Line** entries that appear in elements where whitespace is significant.

Note: When the user tries to insert an element in an invalid position (for example, using the content completion assistant), the editor will attempt to make the insertion valid. This may mean finding an alternate position for the insertion or splitting the element at the current position. If a **<SPLIT>** entry is added for an element, the editor will never split that element.

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 (on page 1324)*.
- **Elements View** - The element will not appear in the **Elements** view *(on page 247)*.
- **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 *(on page 240)*).
- **Entities View** - The element will not appear in the **Entities** view *(on page 247)*.

Related Information:

Templates Tab

The **Templates** tab specifies a list of directories where new document templates are located for this particular framework. These directories, along with the document templates that are saved inside them, will appear in the **New from templates** wizard *(on page 177)* inside the **Framework templates** category according to your framework and the directory path you specify in this tab.

To open the **Templates** tab of the **Document type** configuration dialog box, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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 a directory that contains document templates for this framework. You can specify the path by using the text field, its history drop-down, the **Insert Editor Variables** *(on page 148)* button, or the browsing actions 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 a directory that contains document templates for this framework. You can specify the path by using the text field, its history drop-down, the **Insert Editor Variables** *(on page 148)* button, or the browsing actions 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 directory from the list.

Move Up

Moves the selected template directory up one spot in the list.

Move Down

Moves the selected template directory down one spot in the list.

Related Information:

Creating New Document Templates *(on page 178)*

Customizing Document Templates *(on page 179)*

Sharing Custom Document Templates *(on page 182)*

Catalogs Tab

The **Catalogs** tab specifies a list of *XML Catalogs (on page 312)*, specifically for the edited *framework (on page 1325)*, that are added to list of catalogs that Oxygen XML Developer Eclipse plugin uses to resolve resources.

To open the **Catalogs** tab of the **Document type** configuration dialog box, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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** *(on page 148)* button, or the browsing actions 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 (on page 1325)* you are editing. These transformation scenarios are presented in the **Configure Transformation Scenarios** dialog box *(on page 765)* 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 *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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 *(on page 765)*.

New

Opens the **New scenario** dialog box allowing you to create a new transformation scenario for the particular document type *(on page 691)*.

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 *(on page 764)*.

Edit

Opens the **Edit scenario** dialog box allowing you to edit the properties of the currently selected transformation scenario *(on page 762)*.

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 (on page 1325)* 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*, along with any Schematron schema defined in the default scenarios for that particular *framework*, are used for the validation. These take precedence over other types of validation units defined in the default scenarios for the particular *framework*. For more information on *master files*, see Master Files Support *(on page 200)* or Working with Modular XML Files in the Master Files Context *(on page 315)*.

To open the **Validation** tab of the **Document type** configuration dialog box, open the **Preferences** dialog box *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, 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 *(on page 288)*.

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 *(on page 293)*.

Edit

Opens the **Edit scenario** dialog box allowing you to edit the properties of the currently selected validation scenario *(on page 293)*.

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 *(on page 36)*, go to **Document Type Association**, use the **New**, **Edit**, **Duplicate**, or **Extend** button *(on page 50)*, and click on the **Extensions** tab.

Libraries containing the implementations must be present in the *classpath (on page 55)* 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 Eclipse plugin offers the possibility to configure the appearance of various components and features of the main editor. To access these options, open the **Preferences** dialog box *(on page 36)* 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 (on page 1324)*.

Completion proposal foreground

Allows you to set the color of the text in the *Content Completion Assistant (on page 1324)*.

Documentation window background

Allows you to set the background color of the documentation of elements suggested by the *Content Completion Assistant (on page 1324)*.

Documentation window foreground

Allows you to set the color of the text for the documentation of elements suggested by the *Content Completion Assistant (on page 1324)*.

Line wrap

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

Enable folding when opening a new editor

If selected (default value), the vertical stripe that holds the folding markers *(on page 231)* is displayed in **Text** mode.

Beep on operation finished

Oxygen XML Developer Eclipse 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 (on page 1328)* icon ( ) and *Quick Fix (on page 1328)* 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 Eclipse plugin highlights the corresponding member of the pair.

Tip: You can configure the colors and how various types of highlights are shown from the Eclipse **Annotations** preferences page (**Window ('Eclipse' on Mac OSX) > Preferences > General > Editors > Text Editors > Annotations**).

Minimum fold range

You can specify the minimum number of lines in a block that will have the *folding (on page 1325)* support become active. If you modify this value, the change takes effect next time you open the editor.

Content Completion Preferences

Oxygen XML Developer Eclipse plugin provides a *Content Completion Assistant (on page 1324)* 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 *(on page 36)* 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 selected, Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin inserts both start and end tags. The following additional options are available with regard 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 Eclipse plugin inserts the required elements specified in the DTD, XML Schema, or RELAX NG schema that is associated with the edited XML document *(on page 303)*.
  - **Add optional content** - If selected, Oxygen XML Developer Eclipse plugin inserts the optional elements specified in the DTD, XML Schema, or RELAX NG schema.
  - **Add first Choice particle** - If selected, Oxygen XML Developer Eclipse plugin inserts the first **choice** particle specified in the DTD, XML Schema, or RELAX NG schema.

Case sensitive search

When selected, 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 selected, Oxygen XML Developer Eclipse 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 *(on page 78)* is not selected.

Show all entities

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

Insert the fixed attributes

If selected, Oxygen XML Developer Eclipse 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 selected, Oxygen XML Developer Eclipse 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 selected, Oxygen XML Developer Eclipse plugin learns the attribute values used in a document.

Learn on open document

Oxygen XML Developer Eclipse 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 selected, Oxygen XML Developer Eclipse 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: For the words to be learned, they need to be separated by space characters.

Activation delay of the proposals window (ms)

Delay in milliseconds from the 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 Eclipse plugin can display these annotations when offering content completion suggestions. To configure the **Annotations** preferences, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Content Completion > Annotations**.

The following options are available:

Show annotations in Content Completion Assistant

If selected, Oxygen XML Developer Eclipse plugin displays the schema annotations of an element, attribute, or attribute value currently selected in the *Content Completion Assistant (on page 1324)* proposals list.

Show annotations in tooltip

If selected, Oxygen XML Developer Eclipse 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 *(on page 247)*. If not selected, tooltips are disabled in all modes.

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. If this option is not selected, 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 Eclipse 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.

If this option is selected, Oxygen XML Developer Eclipse 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.

If not selected, all comments, regardless of their type, are considered annotations and displayed as such.

Use all Relax NG annotations as documentation

If 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 (on page 1324)* window and in the **Model** view *(on page 245)*. 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 237)*

XPath Preferences

Oxygen XML Developer Eclipse plugin provides content-completion support for XPath expressions. To configure the options for the content completion in XPath expressions, open the **Preferences** dialog box *(on page 36)* 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 *(on page 370)* 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 *(on page 374)* section for more information.

XSD Preferences

Oxygen XML Developer Eclipse plugin provides content completion assistance when you are writing XML Schema (XSD). To configure XSD preferences, open the **Preferences** dialog box *(on page 36)* 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 (on page 1324)* 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 Eclipse plugin can suggest elements from these vocabularies. To configure the XSLT content completion options, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Content Completion > XSLT**.

The following options are available:

Include elements declared in the schema section

This section includes options with regard 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 Eclipse plugin analyzes the namespaces declared in the root element to find an appropriate schema.

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

- **None** - The *Content Completion Assistant (on page 1324)* suggests only XSLT elements.
- **HTML** - The *Content Completion Assistant (on page 1324)* includes HTML elements, including HTML5 elements (such as `<video>`, `<canvas>`, etc.).
- **Formatting objects** - The *Content Completion Assistant (on page 1324)* 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:

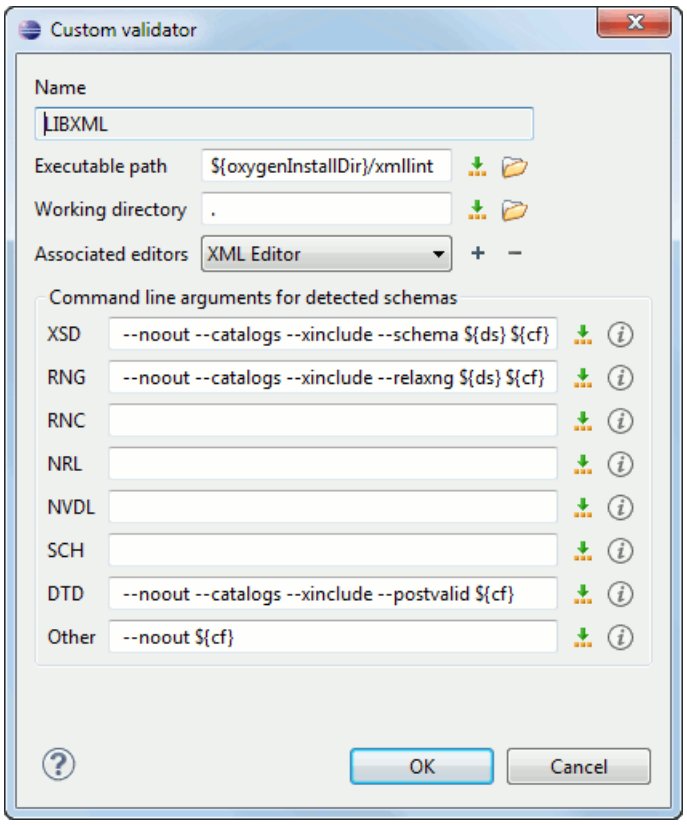
- **Built-in schema** - Uses the built-in schema for documentation.
- **Custom schema** - Allows you to specify a custom schema for documentation. The supported schema types are XSD, RNG, RNC, DTD, and NVDL.

Custom Validation Engines Preferences

As the name implies, the **Custom Validation Engines** preferences page displays the list of custom validation engines that can be associated to a particular editor and used for validating documents. To access this page, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Document Validation > Custom Validation Engines**.

If you want to add a new custom validation tool or edit the properties of an existing one, you can use the **Custom Validator** dialog box displayed by pressing the **New** or **Edit** button.

Figure 23. 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** *(on page 148)* 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** *(on page 148)* 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 (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 *(on page 148)*:

- **${cf}** - Current file as file path, that is the absolute file path of the currently edited document.
- **${currentFileURL}** - Current file as URL, that is the absolute file path of the currently 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 148)*

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.
- **Note:** Increasing the value of the **-Xss** parameter affects all the threads of the application.

Document Validation Preferences

To configure document validation options, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Document Validation**. 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 *(on page 279)* and manual validation *(on page 279)*.

Clear validation markers on close

If this option is selected, all the error markers added in the **Problems** view for that document are removed when the Oxygen XML Developer Eclipse plugin 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 Eclipse 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 Eclipse plugin lets you configure which edit mode *(on page 168)* 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 *(on page 36)* and go to **Editor > Edit Modes** .

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

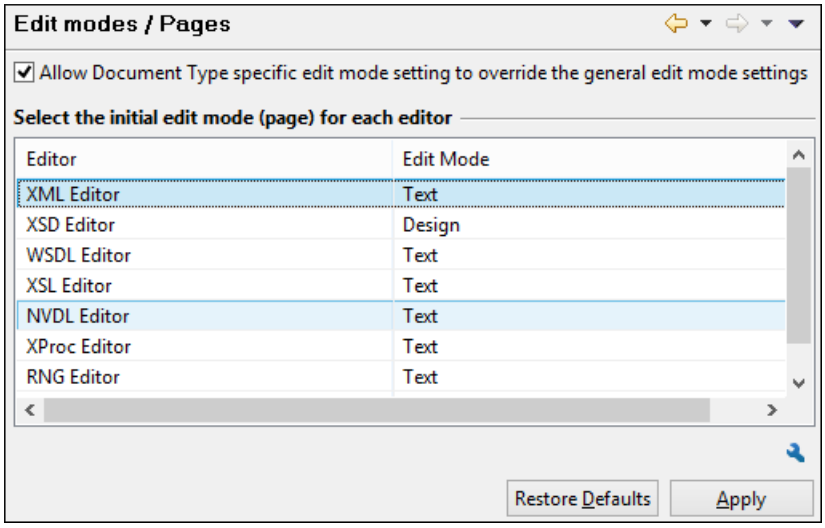
If selected, the initial edit mode setting set in the **Document Type** configuration dialog box *(on page 52)* overrides the general edit mode setting from the table below.

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** *(on page 168)*
- **Grid** *(on page 168)*
- **Design** (available only for the XSD editor).

Figure 24. Edit Modes Preferences Page



Grid Preferences

Oxygen XML Developer Eclipse plugin provides a **Grid** view *(on page 168)* of an XML document. To configure the **Grid** options, open the **Preferences** dialog box *(on page 36)* 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 *(on page 251)* 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 (on page 1323)* of the grid. 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 (on page 1323)*.

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 Eclipse plugin provides a graphical schema design editor *(on page 169)* to make editing XML Schema easier. To configure the **Schema Design** options, open the **Preferences** dialog box *(on page 36)* 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 Eclipse 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 Eclipse 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 Eclipse plugin asks if you want to open the imported schema file.

Properties Preferences

Oxygen XML Developer Eclipse plugin lets you control which properties to display for XML Schema components in the XML Schema **Design** view *(on page 169)*. To configure the schema design properties displayed, open the **Preferences** dialog box *(on page 36)* 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 Eclipse plugin provides a diagram view as part of the **Text** mode editor. To configure the **Diagram** preferences, open the **Preferences** dialog box *(on page 36)* 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 schema using a diagram instead of text, use the schema **Design** view *(on page 169)*.

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** *(on page 243)*, **Model** *(on page 245)*, **Elements** *(on page 247)*, **Outline** *(on page 533)*).

Show Relax NG diagram

Displays the Relax NG schema diagram in the split-screen views (**Full Model View** *(on page 526)* and **Logical Model View** *(on page 527)*).

Enable NVDL diagram and related views

Enables the NVDL schema diagram and synchronization with the related views (**Attributes** *(on page 243)*, **Model** *(on page 245)*, **Elements** *(on page 247)*, **Outline** *(on page 547)*).

Show NVDL diagram

Displays the NVDL schema diagram in the split-screen views (**Full Model View** *(on page 543)* and **Logical Model View** *(on page 544)*).

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 *(on page 87)* 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 *(on page 669)*.

To configure the **Format** options, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Format**.

The following options are available:

Detect indent on open

If selected, Oxygen XML Developer Eclipse plugin detects how a document is indented when it is opened. Oxygen XML Developer Eclipse 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 Eclipse 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** *(on page 90)* 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 of your documents 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** *(on page 89)* option.

Indent size

The meaning of this setting depends on the following:

- If the **Detect indent on open** option *(on page 89)* 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 Eclipse plugin uses the number of detected SPACE characters.
- If the **Indent with tabs** option *(on page 90)* 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 255)*.

For information about when this setting is used, see Where Indent Size and Line Width Settings are Used in Oxygen XML Developer Eclipse plugin *(on page 91)*.

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 Eclipse 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 Eclipse 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 *(on page 91)*.

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 Eclipse plugin will be used for storing the undo states. If this option is selected, Oxygen XML Developer Eclipse 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 Eclipse 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.

For more information about the formatting options offered by Oxygen XML Developer Eclipse plugin, watch our video demonstration:

https://www.youtube.com/embed/1plmdN0Cfso

CSS Preferences

Oxygen XML Developer Eclipse plugin can format and indent your CSS files. To configure the **CSS** formatting options, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Format > CSS**.

The following options control how your CSS files are formatted and indented:

Class body on new line

If selected, the *class* body (including the curly brackets) is placed on a new line. This option is not selected by default.

Indent class content

When selected (default state), the *class* content is indented.

Add space before the value of a CSS property

When selected (default state), whitespaces are added between the `:` (colon) and the value of a style property.

Add new line between classes

If selected, an empty line is added between two classes. This option is not selected by default.

Preserve empty lines

When selected (default state), the empty lines from the CSS content are preserved.

Allow formatting embedded CSS

When selected (default state), CSS content that is embedded in XML is also formatted when the XML content is formatted.

JavaScript Preferences

To configure the **JavaScript** format options, open the **Preferences** dialog box *(on page 36)* 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 selected by default.
- **Allow formatting embedded JavaScript** - Applied only to XHTML documents, this option allows Oxygen XML Developer Eclipse plugin to format embedded JavaScript code, taking precedence over the Schema-aware format and indent *(on page 95)* option. This option is selected by default.

XML Preferences

To configure the **XML** Formatting options, open the **Preferences** dialog box *(on page 36)* 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 selected, the **Break long attributes** option *(on page 93)* 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 selected:

```
<root> text <parent>
    <child/>
  </parent>
</root>
```

Indent inline elements not selected:

```
<root> text <parent> <child/> </parent> </root>
```

Expand empty elements

If not selected (default), the **Format and Indent** operation results in an empty XML element being serialized in a compact form (`<a atr1="v1"/>`). If selected, the same operation results in empty XML elements being serialized in expanded form (for example, `<a atr1="v1"></a>`).

Note: When using the **Format and Indent** operation in **Text** mode, if the **Schema-aware format and indent** option *(on page 95)* is enabled, Oxygen XML

Developer Eclipse plugin will use information from the associated schema and avoid expanding tags for elements that are defined as *empty* in the schema.

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 that will have the **Format and Indent** operation preserve the whitespaces (such as blanks, tabs, and newlines).

Default space

List of elements that will have the content 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 that will have line breaks 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 regard 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 *(on page 94)*) are ignored by the **Format and Indent** operation. When this option is selected 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, deselect this option.

Whitespaces Preferences

When Oxygen XML Developer Eclipse plugin formats and indents XML documents, a whitespace normalization process is applied, thus replacing whitespace sequences with single space characters. Oxygen XML Developer Eclipse 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 *(on page 36)* and go to **Editor > Format > XML > Whitespaces**.

This table lists the Unicode whitespace characters. Select 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 *(on page 36)* and go to **Editor > Format > XPath**.

The following option is available:

Format XPath code embedded in XSLT, XSD and Schematron files

If selected, 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 *(on page 36)* 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 *(on page 258)* feature activated. To configure these options, open the **Preferences** dialog box *(on page 36)* 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 *(on page 258)* feature in XML files.
- **XSLT files** - Activates the Highlight Component Occurrences *(on page 385)* feature in XSLT files.
- **XML Schema files** - Activates the Highlight Component Occurrences *(on page 458)* feature in XSD files.
- **WSDL files** - Activates the Highlight Component Occurrences *(on page 458)* feature in WSDL files.
- **RNG files** - Activates the highlight component occurrences feature in RNG files.
- **Schematron files** - Activates the Highlight Component Occurrences *(on page 601)* feature in Schematron files.

Open/Save Preferences

Oxygen XML Developer Eclipse plugin lets you control how files are opened and saved. To configure the options for opening and saving documents, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Open/Save**.

The following options are available:

Open section

Format document when longest line exceeds

Oxygen XML Developer Eclipse 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

Check errors on save

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

Save all files before transformation or validation

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

Performance section

Clear undo buffer on save

If selected, Oxygen XML Developer Eclipse 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 Eclipse plugin provides support for spell checking in the **Text** *(on page 168)* editing mode. To configure the **Spell Check** options, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Spell Check**.

The following options are available:

Automatic spell check

This option is not selected by default. When selected, Oxygen XML Developer Eclipse 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 automatically be spell checked. File types such as CSS and DTD are excluded by default since automatic spell checking is not usually helpful in these types of files.

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 *(on page 205)*.

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 *(on page 98)* 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 detected capitalization errors.

Note: This option will not have any effect on words stored in dictionaries, term lists, and the list of learned words because they are not handled as case-sensitive.

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 *(on page 36)* 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 Eclipse plugin uses and to choose where to save new learned words.

The following options are valid when Oxygen XML Developer Eclipse 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 Eclipse 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 regarding 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 Eclipse plugin User Guide that explains how to add more dictionaries and term lists *(on page 208)*.

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.

Note: Words stored in dictionaries, term lists, and the list of learned words are not handled as case-sensitive. Therefore, you do not need to include both uppercase and lowercase versions of the words.

Related Information:

Adding Custom Spell Check Dictionaries *(on page 208)*

Adding Custom Spell Check Term Lists *(on page 210)*

Syntax Highlight Preferences

Oxygen XML Developer Eclipse 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 *(on page 36)* 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 Eclipse 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 *(on page 36)* 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, select 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 *(on page 238)* are code fragments that can be inserted at the current editing position. Oxygen XML Developer Eclipse plugin includes a set of built-in templates for CSS, Schematron, XSL, XQuery, JSON, HTML, and XML Schema document types. You can also define your own code templates *(on page 239)* for any type of file and share them with your colleagues *(on page 240)* using the template export and import functions.

To configure **Code Templates**, open the **Preferences** dialog box *(on page 36)* 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** - [Optional] 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 (on page 1324)*. 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** - [Optional] If you want to assign a shortcut key that can be used to insert the code template, place the cursor in the **Shortcut key** field and press the desired key combination on your keyboard. Use the **Clear** button if you make a mistake. If the **Enable platform-independent shortcut keys** checkbox is selected, 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. An editor variable *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button.

Edit

Opens the **Code template** dialog box and allows you to edit an existing code template. You can edit the following fields:

- **Description** - [Optional] 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 (on page 1324)*. HTML markup can be used for better rendering.
- **Shortcut key** - [Optional] If you want to assign a shortcut key that can be used to insert the code template, place the cursor in the **Shortcut key** field and press the desired key combination on your keyboard. Use the **Clear** button if you make a mistake. If the **Enable platform-independent shortcut keys** checkbox is selected, 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. An editor variable *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button.

Duplicate

Creates a duplicate of the currently selected code template.

Delete

Deletes the currently selected code template. This action is not available 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 *(on page 148)* 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 currently selected text content in the currently edited document. This variable can be used in a code template, in **Author** mode operations, or in a **selection** *plugin*.
- **${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value', @id)}** - To prompt for values at runtime, use the *ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value')* editor variable. You can set the following parameters:
  - **'message'** - The displayed message. Note the quotes that enclose the message.
  - **'default-value'** - Optional parameter. Provides a default value.
  - **@id** - Optional parameter. Used for identifying the variable to reuse the answer using the **${answer(@id)}** editor variable.
  - **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 *relative_url* 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:** *${ask('message', url, 'default_value')}*
	Description: Input is considered a URL. Oxygen XML Developer Eclipse plugin checks that the provided URL is valid.
	Example: ▪ *${ask('Input URL', url)}* - The displayed dialog box has the name *Input URL*. The expected input type is URL. ▪ *${ask('Input URL', url, 'http://www.example.com')}* - The displayed dialog box has the name *Input URL*. The expected input type is URL. The input field displays the default value *http://www.example.com*.
password	**Format**: *${ask('message', password, 'default')}*
	Description: The input is hidden with bullet characters.
	Example: ▪ *${ask('Input password', password)}* - The displayed dialog box has the name *'Input password'* and the input is hidden with bullet symbols. ▪ *${ask('Input password', password, 'abcd')}* - The displayed dialog box has the name *'Input password'* and the input hidden

Parameter	
	with bullet symbols. The input field already contains the default **abcd** value.
generic	**Format**: *${ask('message', generic, 'default')}*
	Description: The input is considered to be generic text that requires no special handling.
	Example: ▪ *${ask('Hello world!')}* - The dialog box has a *Hello world!* message displayed. ▪ *${ask('Hello world!', generic, 'Hello again!')}* - The dialog box has a *Hello world!* message displayed and the value displayed in the input box is *'Hello again!'*.
relative_url	**Format:** *${ask('message', relative_url, 'default')}*
	Description: Input is considered a URL. Oxygen XML Developer Eclipse plugin tries to make the URL relative to that of the document you are editing. **Note:** If the *$ask* editor variable is expanded in content that is not yet saved (such as an *untitled* file, whose path cannot be determined), then Oxygen XML Developer Eclipse plugin will transform it into an absolute URL.
	Example: *${ask('File location', relative_url, 'C:/example.txt')}* - The dialog box has the name *'File location'*. The URL inserted in the input box is made relative to the currently edited document location.
combobox	**Format:** *${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}*
	Description: Displays a dialog box that offers a drop-down menu. The drop-down menu is populated with the given *rendered_value* values. Choosing such a value will return its associated value (*real_value*). **Note:** The *'default'* parameter specifies the default-selected value and can match either a key or a value.
	Example: ▪ *${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}* - The dialog box has the name *'Operating System'*. The drop-down menu displays the three given operating systems. The associated value will be returned based upon your selection.

Parameter	
	Note: In this example, the default value is indicated by the *osx* key. However, the same result could be obtained if the default value is indicated by *Mac OS X*, as in the following example: *${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'Mac OS X')}* ▪ *${ask('Mobile OS', combobox, ('win':'Windows Mobile';'ios':'iOS';'and':'Android'), 'Android')}*
editable_combobox	**Format:** *${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}*
	Description: Displays a dialog box that offers a drop-down menu with editable elements. The drop-down menu is populated with the given *rendered_value* values. Choosing such a value will return its associated real value (*real_value*) or the value inserted when you edit a list entry. **Note:** The *'default'* parameter specifies the default-selected value and can match either a key or a value.
	Example: *${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}* - The dialog box has the name *'Operating System'*. The drop-down menu displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.
radio	**Format:** *${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}*
	Description: Displays a dialog box that offers a series of radio buttons. Each radio button displays a *'rendered_value* and will return an associated *real_value*. **Note:** The *'default'* parameter specifies the default-selected value and can match either a key or a value.
	Example: *${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}* - The dialog box has the name *'Operating System'*. The radio button group allows you to choose between the three operating systems.

Parameter	
	Note: In this example, `Mac OS X` is the default-selected value and if selected, it would return *osx* for the output.

- **${timeStamp}** - The timestamp, which is the current time in Unix format. For example, it can be used to save transformation results in multiple output files on each transformation.
- **${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 the extension and parent folder. The current file is the one currently open and selected.
- **${cfne}** - Current file name with extension. The current file is the one currently open and selected.
- **${cf}** - Current file as file path, that is the absolute file path of the currently edited document.
- **${cfd}** - Current file folder as file path, that is the path of the currently edited document up to the name of the parent folder.
- **${frameworksDir}** - The path (as file path) of the `frameworks` directory.
- **${pd}** - The file path to the folder that contains the current project file (`.xpr`).
- **${oxygenInstallDir}** - Oxygen XML Developer Eclipse plugin installation folder as file path.
- **${homeDir}** - The path (as file path) of the user home folder.
- **${pn}** - Current project name.
- **${env(VAR_NAME)}** - Value of the *VAR_NAME* environment variable. The environment variables are managed by the operating system. If you are looking for Java System Properties, use the **${system(var.name)}** editor variable.
- **${system(var.name)}** - Value of the *var.name* Java System Property. The Java system properties can be specified in the command-line arguments of the Java runtime as *-Dvar.name=var.value*. If you are looking for operating system environment variables, use the **${env(VAR_NAME)}** editor variable instead.
- **${date(pattern)}** - Current date. The allowed patterns are equivalent to the ones in the Java SimpleDateFormat class. **Example:** *yyyy-MM-dd*.

  Note: This editor variable supports both the **xs:date** and **xs:datetime** parameters. For details about **xs:date**, go to: http://www.w3.org/TR/xmlschema-2/#date. For details about **xs:datetime**, go to: http://www.w3.org/TR/xmlschema-2/#dateTime.

Related Information:

Code Templates *(on page 238)*

Document Templates Preferences

Oxygen XML Developer Eclipse plugin provides a variety of built-in document templates that make it easier to create new documents in various formats. The list of available templates is presented in the **New Document**

wizard *(on page 171)* when you create a new document (**New** toolbar button or **File > New > New from Templates**).

You can also create your own templates *(on page 178)* and share them with others. You can store your custom document templates in the existing `templates` folder in the Oxygen XML Developer Eclipse plugin installation directory or store them in a custom directory. If you store them in a custom directory, you need to use this **Document Templates** preferences page to add that directory to the list of template directories that Oxygen XML Developer Eclipse plugin makes available in the **New Document** wizard.

To add a template directory, follow these steps:

1. open the **Preferences** dialog box *(on page 36)* and go to **Editor > Templates > Document Templates**.
2. Use the **New** button to select a location of the new document template folder.
3. You can also use the **Edit** or **Delete** buttons to manage folders in the list, and you can alter the order that Oxygen XML Developer Eclipse plugin looks in these directories by using the **Up** and **Down** buttons.

Result: This will add the folder to the list in this preferences page and it will now appear in the **New from templates** wizard *(on page 177)* in a category based upon the folder path you specified.

Note: For DITA templates, they will also appear in the dialog box for creating new DITA topics, but if you customize the template *(on page 179)*, you need to set the `type` property to **dita** in the corresponding properties file.

Fonts Preferences

Oxygen XML Developer Eclipse 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 *(on page 36)* and go to **Fonts**.

The following options are available:

Text

This option allows you to choose the font used in **Text** mode. There are two options available:

- **Map to text font** - Uses the same font for the basic text editor as the one set in **General > Appearance > Colors and Fonts**.
- **Customize** - Allows you to configure various font-related options.

Schema

Allows you to configure various font-related options that will be used in:

- The **Design** mode of the XML Schema editor *(on page 406)*.
- 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.

Markdown Preferences

The **Markdown** preferences page makes it possible to validate Markdown documents with Schematron. To access the page, open the **Preferences** dialog box *(on page 36)* and go to **Markdown**. This preferences page includes the following options:

Validate converted HTML content

If selected, converted HTML content will be validated using the Schematron file specified in this option.

Validate converted DITA content

If selected, converted DITA content will be validated using the Schematron file specified in this option.

Note: It is also possible to create a Schematron association for Markdown documents by adding a catalog mapping *(on page 312)* for one of the following URIs:

- **http://www.oxygenxml.com/schematron/validation/markdown-as-html**
- **http://www.oxygenxml.com/schematron/validation/markdown-as-dita**

The catalog mapping is a fallback in case the validation is disabled in this preferences page or the path to the Schematron is empty. The associations configured in this preferences page take precedence.

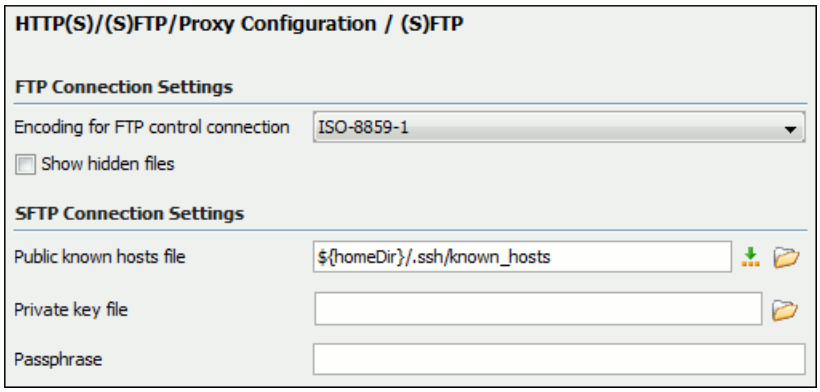
Network Connection Settings Preferences

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 *(on page 36)* and go to **Network Connection Settings > (S)FTP**. You can customize the following options:

Figure 25. (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 Eclipse 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 selected, 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 *(on page 36)* 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 Eclipse plugin. Any adjustment to this option requires a restart of the application.

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 selected, the HTTPS connections that Oxygen XML Developer Eclipse 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.

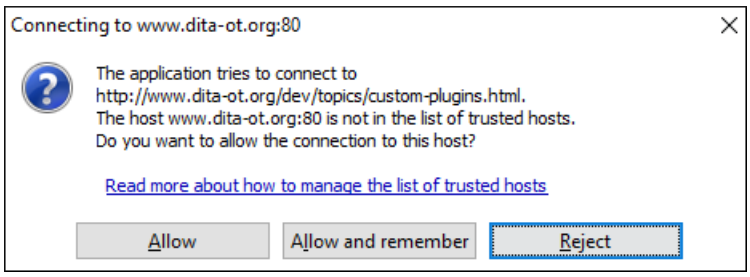
Lock WebDAV files on open

If selected, 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

Oxygen XML Developer Eclipse plugin comes with a built-in firewall that controls the access to external resources. Anytime the application detects a request to connect to a remote resource, it checks to see if the URL belongs to a domain that has been identified as trusted. If not, a confirmation dialog box will be displayed where you can choose whether to allow or reject access to the remote connection.

Figure 26. Trusted Hosts Confirmation Dialog Box



You can configure the list of trusted hosts using the **Trusted Hosts** preferences page. It contains a list of domains that have been identified as trusted. You can add or remove domains from the list and Oxygen XML Developer Eclipse plugin will allow connections to the listed hosts without requesting user confirmation.

Note: Connections defined in the **Data Sources** preferences page *(on page 40)* or accepted by add-ons are also considered trusted.

To add or remove domains, open the **Preferences** dialog box *(on page 36)* and go to **Network Connection Settings > Trusted Hosts**. The following options are available:

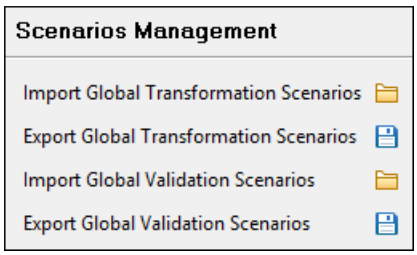
- **New** - Allows you to manually add a new entry to the list of trusted hosts.

  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 *(on page 36)* 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 27. Scenarios Management Preferences Panel



The actions available in this panel are as follows:

- **Import Global Transformation Scenarios** - Allows you to import all 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 validation 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 regarding certain views. To edit these options, open the **Preferences** dialog box *(on page 36)* and go to **View**.

The following options are available:

Console section

Enable Oxygen consoles

If selected, various messages will be contributed to the **Console** view *(on page 220)* when certain events are triggered (such as *schema detection*, *validation*, or *transformation* events).

Fixed width console

If selected, a line in the **Console** view *(on page 220)* will be hard wrapped after the specified maximum numbers of characters allowed on a line is reached.

Limit console output

If selected, the content of the **Console** view *(on page 220)* will be limited to a configurable number of characters.

Console buffer

If the **Limit console output** option is selected, this specifies the maximum number of characters that can be written in the **Console** view *(on page 220)*.

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 *(on page 36)* 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 selected, 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 selected, `null` values from a database column are imported as empty elements.

Escape XML content

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

Add annotations for generated XML Schema

If selected, 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

Table 1. Pattern Letters (continued)

Letter	Date or Time Component	Presentation	Examples
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am / pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am / pm (0-11)	Number	0
h	Hour in am / pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	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)

PDF Output Preferences

The **PDF Output** preferences page simply includes links to sub-pages for configuring PDF output options.

FO Processors Preferences

Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse 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 *(on page 716)* and XML with XSLT transformation scenarios *(on page 691)*.

To configure the options for the FO Processors, open the **Preferences** dialog box *(on page 36)* and go to **XML > PDF Output > FO Processors**. This preferences page includes the following options:

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 Eclipse plugin to use the built-in Apache FO processor. To see the version of the built-in XSL-FO processor for your installation, go to **Help > About > Libraries** and search for *Apache FOP*.

Use other Apache FOP

Instructs Oxygen XML Developer Eclipse 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** *(on page 148)* 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 Eclipse 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** *(on page 148)* 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 *(on page 747)*.

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).

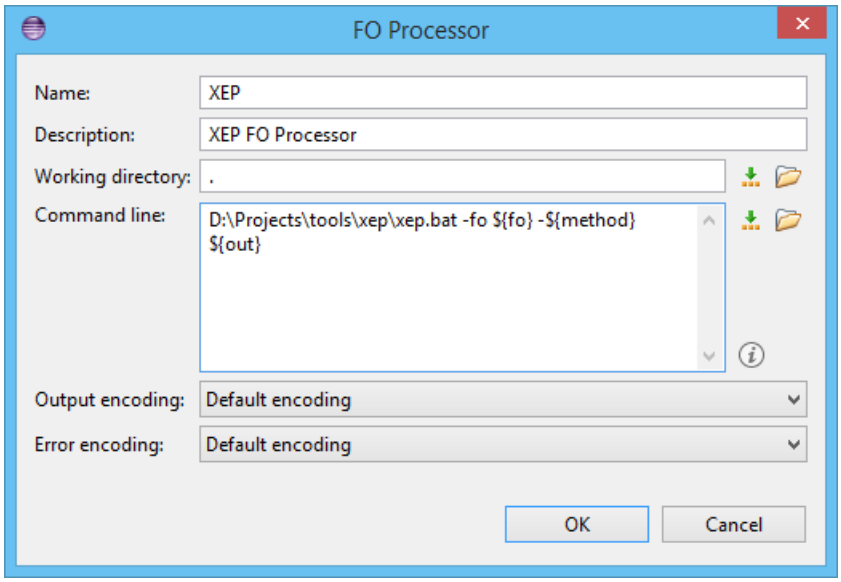
Add 'Antenna House' FO processor (executable file is needed)

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 *(on page 117)*, 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. Click 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 28. 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 are stored. You can specify the path by using the text field, the **Insert Editor Variables** *(on page 148)* button, or the **Browse** button. You can use one of the following editor variables *(on page 148)*:

- **${homeDir}** - The path to the user home directory.
- **${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 Eclipse 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** *(on page 148)* button, or the **Browse** button. You can use one of the following editor variables *(on page 148)*:

- **${method}** - The FOP transformation method: **pdf**, **ps**, or **txt**.
- **${fo}** - The input FO file.
- **${out}** - The output file.
- **${pd}** - The project directory.
- **${frameworksDir}** - The path of the `frameworks` subdirectory of the Oxygen XML Developer Eclipse plugin installation directory.
- **${oxygenInstallDir}** - The Oxygen XML Developer Eclipse 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 *(on page 248)* at the bottom of the Oxygen XML Developer Eclipse plugin window.

Error Encoding

The encoding of the FO processor error stream that is displayed in a **Results** panel *(on page 248)* at the bottom of the Oxygen XML Developer Eclipse plugin window.

CSS-based Processors Preferences

Oxygen XML Developer Eclipse plugin includes a built-in **XML to PDF transformation with CSS** scenario type for generating PDF output using a CSS-based processor.

To configure the options for the CSS-based processors, open the **Preferences** dialog box *(on page 36)* and go to **XML > PDF Output > CSS-based Processors**. This preferences page includes the following options:

Oxygen PDF Chemistry Section

Auto-detect

If selected, the directory of the **Chemistry** processor will be automatically detected. This is based on the system's PATH environmental variable. If none is detected, it will use the path of the built-in distribution.

Custom installation directory

Use this option to select an external directory of a custom installation of the **Chemistry** processor.

Memory available to the processor (MB)

Specifies the maximum amount of memory that is available for the transformation. If your transformations fail with an Out of Memory error (**OutOfMemoryError**), you can use this option to select a bigger value for the amount of memory reserved for the process.

Generates PDF/UA-1 output

Use this option to produce output that conforms with the PDF/UA-1 accessibility standards.

Note: This mode has some special requirements. For example, all fonts have to be embedded and the title of documents must be marked using the metadata. For more information, see Oxygen PDF Chemistry User Guide: Fully Accessible PDF (PDF/UA1).

Show console output

Allows you to specify when to display the console output log in the message panel at the bottom of the editor. The following options are available:

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

Sample XML Files Generator Preferences

The **Generate Sample XML Files** tool *(on page 461)* (available on the **XML Tools** menu) allows you to generate XML instance documents based on an 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 *(on page 36)* and go to **XML > Sample XML Files Generator**.

The following options are available:

Generate optional elements

When selected, all elements are generated, including the optional ones (having the `minOccurs` attribute set to 0 in the schema).

Generate optional attributes

When selected, 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 on the 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 **Sample XML Files Generator** preferences page). Note that type restrictions are ignored when this option is selected. 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 on the data type descriptor of the particular element or attribute.

  Important: If all of the following are true, the **Generate Sample XML Files** tool 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 selected, 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 selected, 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 selected, 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 461)*

XML Catalog Preferences

To configure options that pertain to *XML Catalogs (on page 1329)*, open the **Preferences** dialog box *(on page 36)* 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 that has both a public identifier and a system identifier 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, 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 selected, Oxygen XML Developer Eclipse 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 not selected, the system IDs are used to resolve the schema location.

Use default catalog

If this option is selected and Oxygen XML Developer Eclipse 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 Eclipse plugin Determines which Catalog to Use *(on page 314)*.

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** *(on page 148)* button, or the browsing actions 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** *(on page 148)* button, or the browsing actions 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 *(on page 279)* so that the changes take full effect.

You can also add or configure catalogs at *framework* level from the **Catalogs** tab *(on page 73)* in the **Document Type** configuration dialog box *(on page 52)*.

Related Information:

Controlling the Catalog Resolver

Working with XML Catalogs *(on page 312)*

XML Parser Preferences

To configure the **XML Parser** options, open the **Preferences** dialog box *(on page 36)* 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 open XML files that reference the same schemas (grammars) such as DTD, XML Schema, or RelaxNG.

Enable system parameter entity expansion in other entity definitions

This security setting controls the expansion of the DTD system parameter entities (the ones that are loaded from disk or from remote sources). This option is off by default, to protect against XXE attacks. If you enable it, make sure the XML files you are opening or processing with the application come from a trusted source.

Ignore the DTD for validation if a schema is specified

Forces validation against a referenced schema (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 an 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 selected, the XInclude support in Oxygen XML Developer Eclipse 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 infoset 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 deselect 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 included parent has a different [language] property. If the addition of `@xml:lang` is not allowed by your application, you can deselect this option.

DTD post-validation

Select 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 deselect 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 regarding Relax NG, open the **Preferences** dialog box *(on page 36)* 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: Selecting 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.

Ignore "data-" attributes in XHTML

This option is selected by default, which means that when XHTML documents are validated with an RNG schema, any **data-** attributes detected in the document will not be taken into account by the validation engine.

Schematron Preferences

To configure options regarding Schematron, open the **Preferences** dialog box *(on page 36)* 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 select 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 associated XML Schema to expand default attribute values

When selected (default value), if the validated XML document has an XML Schema associated that contains default values for attributes defined in the XML content, the Schematron will be able to match on those default attributes.

Use Saxon EE (schema aware) for xslt2/xslt3 query language binding

When selected, Saxon EE is used for `xslt2`/`xslt3` query binding. If this option is not selected, Saxon PE is used.

Enable Schematron Quick Fixes (SQF) support

Allows you to enable or disable the support for *Quick Fixes (on page 1328)* in Schematron files. This option is selected 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 *(on page 159)* 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 is 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 is defined within 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 regarding XML Schema, open the **Preferences** dialog box *(on page 36)* 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 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 *(on page 171)*.

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 (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 XML Schema components from the same namespace are used to compose the validation model. If this option is not selected, only the first XML Schema file that is encountered in the XML Schema import tree is taken into account.

Enable full XPath 2.0 for alternative types

When selected (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 *(on page 36)* and go to **XML > XML Refactoring**. The following option is available in this preferences page:

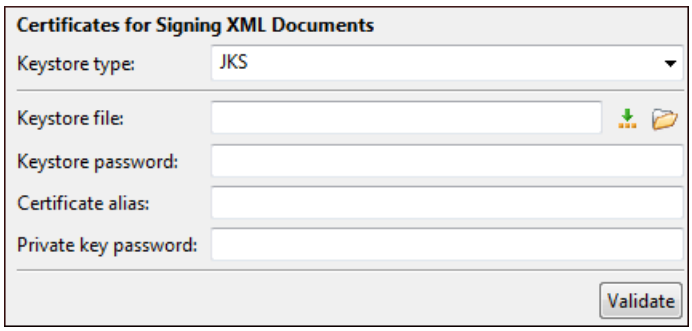
Load additional refactoring operations from

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** *(on page 148)* button, or the **Browse** button. Oxygen XML Developer Eclipse plugin looks for XML refactoring operations recursively in the specified folder, so they can be stored in descendant folders.

XML Signing Certificates Preferences

Oxygen XML Developer Eclipse plugin provides two types of *keystores (on page 1326)* 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 *(on page 36)* and go to **XML > XML Signing Certificates**. You can customize the following parameters of a *keystore*:

Figure 29. Certificates Preferences Panel



- **Keystore type** - The type of *keystore (on page 1326)* that Oxygen XML Developer Eclipse 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 (on page 1326)*.
- **Private key password** - The private key password of the certificate (required only for JKS *keystores (on page 1326)*).
- **Validate** - Click this button to verify the configured *keystore (on page 1326)* and the validity of the certificate.

XProc Preferences

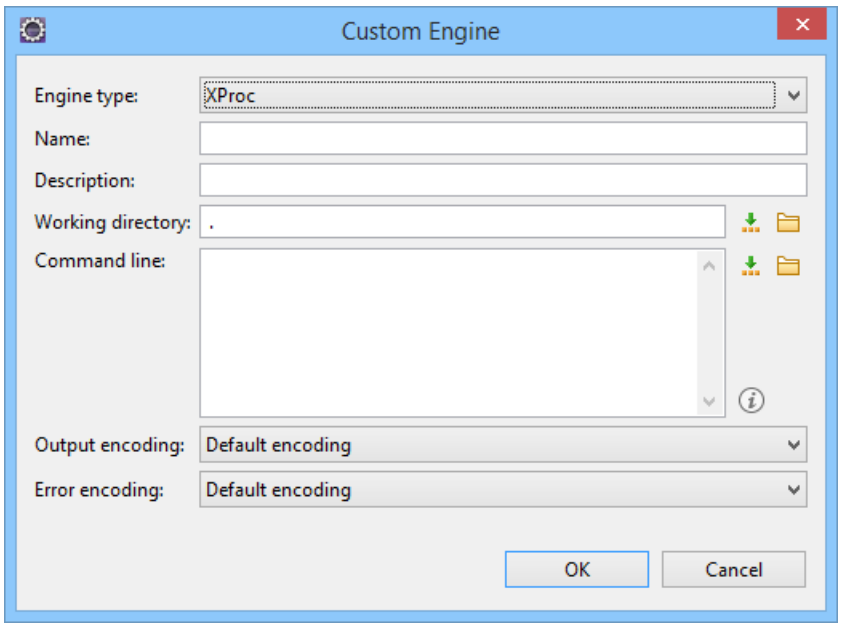
Oxygen XML Developer Eclipse plugin includes a bundled version of the *Calabash* XProc engine that can be used for XProc transformations and validation, but you also have several ways to integrate other external XProc engines.

If the external engine is Java-based, or it has validation support, or it can receive parameters or ports passed from the transformation, you need to integrate the external XProc engine using a plugin extension procedure *(on page 753)*.

If you do not need the engine to be used for automatic validation or pass parameters/ports and it is not Java-based, you can add an external XProc engine by using the **XProc** preferences page. Open the **Preferences** dialog box *(on page 36)* and go to **XML > XProc**.

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 30. Creating an XProc external engine



The following options can be configure in this custom engine configuration 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** *(on page 148)* 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** *(on page 148)* 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 Calabash processor by using the `calabash.config` file. This file is located in *`[OXYGEN_INSTALL_DIR]`*`\lib\xproc\calabash\lib`. If that file does not exist, you have to create it.

The **Show XProc messages** option at the bottom of the **XProc** preferences page can be selected if you want all messages emitted by the XProc processor during a transformation to be presented in dedicated XProc **Results** view *(on page 248)*.

XSLT-XQuery Preferences

To configure options regarding XSLT and XQuery processors, open the **Preferences** dialog box *(on page 36)* and go to **XML > XSLT-XQuery**. This panel contains only the most generic options for working with

XSLT or 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.

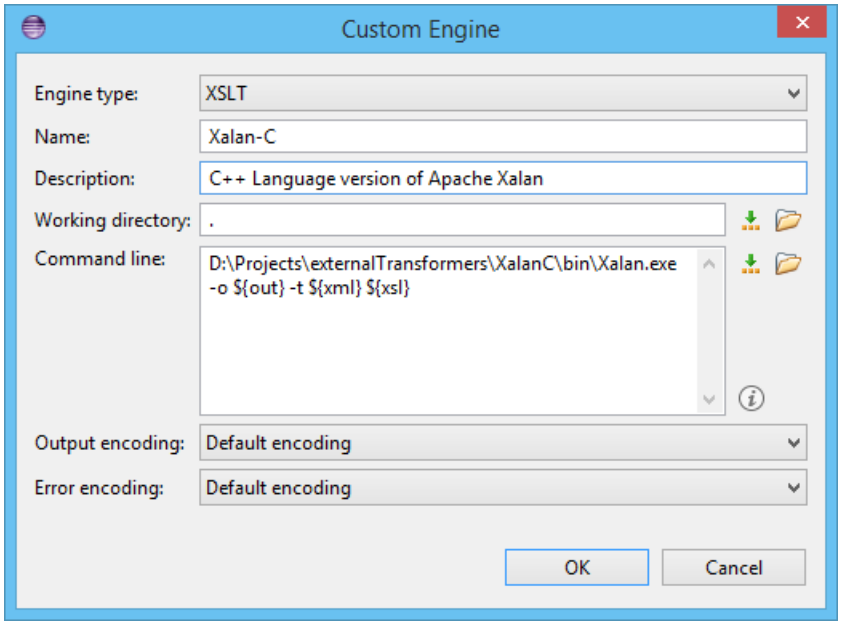
Custom Engines Preferences

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

To configure the **Custom Engines** preferences, open the **Preferences** dialog box *(on page 36)* and go to **XML > XSLT-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 31. 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 *(on page 148)* 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 Eclipse plugin install directory.

Command line

The command line that must be executed by Oxygen XML Developer Eclipse plugin to perform a transformation with the engine. The following editor variables *(on page 148)* 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 *(on page 36)* and go to **XML > XSLT-XQuery > Debugger**.

The following options are available:

Show xsl:result-document output

If selected, the debugger presents the output of `<xsl:result-document>` instructions into the debugger output view.

Infinite loop detection

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

Enable Saxon optimizations

This option is not selected by default and this means that the optimization for the debugging process is suppressed. This is helpful when reducing the compiling time is important, optimization conflicts with debugging, or optimization causes extension functions with side-effects to behave unpredictably.

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 Eclipse plugin allocates to the evaluation of XPath expressions while debugging.

Profiler Preferences

This section explains the settings available for the XSLT/XQuery Profiler. To access and modify these settings, open the **Preferences** dialog box *(on page 36)* and go to **XML > XSLT-XQuery > Profiler** (see Debugger Preferences *(on page 131)*).

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 1233)*.

Ignore invocation less than

Invocations are ignored below this specified amount (in microseconds). For more information, see Invocation Tree View *(on page 1232)*.

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 *(on page 36)* and go to **XML > XSLT-XQuery > XPath**.

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

Unescape XPath expression

If selected, the entities of an XPath expression that you type in the **XPath/XQuery Builder** *(on page 1119)* are unescaped during their execution. For example, the expression:

```
//varlistentry[starts-with(@os,'&#x73;')]
```

is equivalent to:

```
//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 Eclipse 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 Eclipse 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 Eclipse 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 *(on page 36)* and go to **XML > XSLT-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 Eclipse 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 *(on page 712)*) 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 *(on page 496)* 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 (on page 1327)*).

Note: This option is ignored if you choose **Present as a sequence** *(on page 712)* (lazy extract data from a database) from the associated transformation scenario.

Create structure indicating the type nodes

If selected, Oxygen XML Developer Eclipse 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** *(on page 712)* (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 *(on page 36)* and go to **XML > XSLT-XQuery > XQuery > Saxon-HE/PE/EE**.

Oxygen XML Developer Eclipse plugin allows you to configure the following XQuery options for the Saxon 9.9.1.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE):

Use a configuration file ("-config")

Sets a Saxon 9.9.1.5 configuration file that is used for XQuery transformation and validation scenarios.

Enable Optimizations ("-opt")

This option is selected by default, which means that optimization is enabled. If not selected, the optimization is suppressed, which is helpful 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.

Recoverable errors ("-warnings")

Specifies 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")

Specifies 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.

The following option is available for Saxon 9.9.1.5 Professional Edition (PE) and Enterprise Edition (EE) only:

Allow calls on extension functions ("-ext")

If selected, calls on external functions are allowed. Selecting 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 9.9.1.5 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 allows for parsing the source documents with strict schema-validation enabled.
- **Lax schema validation ("lax")** - If an XML Schema is provided, this mode allows for 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. Selecting 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.

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 selected, backup versions for any XML files updated with an XQuery Update are generated. This option is available when the **Enable XQuery update** option is selected.

Saxon HE/PE/EE Advanced Preferences

To configure **Saxon HE/PE/EE Advanced** preferences, open the **Preferences** dialog box *(on page 36)* and go to **XML > XSLT-XQuery > XQuery > Saxon-HE/PE/EE > Advanced**.

The advanced XQuery options that can be configured for the Saxon 9.9.1.5 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 9.9.1.5 transformer (the -r option when run from the command line). The class name must be fully specified and the corresponding *JAR* or class extension must be configured from

the dialog box for configuring the XQuery extension *(on page 695)* 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 (on page 312)*.

- **Collection URI Resolver class name** - Allows you to specify a custom implementation for the Collection URI resolver used by the XQuery Saxon 9.9.1.5 transformer (the -cr option when run from the command line). The class name must be fully specified and the corresponding *JAR* or class extension must be configured from the dialog box for configuring the XQuery extension *(on page 695)* for the particular transformation scenario.

XSLT Preferences

To configure the **XSLT** options, open the **Preferences** dialog box *(on page 36)* and go to **XML > XSLT-XQuery > XSLT**.

The XSLT preferences page allows you to customize options for the default XSLT validation engines. You can also specify the engine directly in a validation scenario *(on page 293)*.

Note: If no specific engine is specified in the validation scenario and the XSLT file has a transformation scenario associated, Oxygen XML Developer Eclipse plugin will use the engine specified in the transformation scenario.

The following options are available in this page:

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 3.1) 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 *(on page 81)*, where you can configure the XSLT content completion options.

MSXML Preferences

To configure the MSXML options, open the **Preferences** dialog box *(on page 36)* and go to **XML > XSLT-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 selected, 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 selected, the resolution is disabled.

Strip non-significant whitespaces

If selected, strips non-significant white space from the input XML document during the load phase. Selecting this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.

Show time information

If selected, 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 *(on page 36)* and go to **XML > XSLT-XQuery > XSLT > MSXML.NET**.

The options in this preferences page for the MSXML.NET processor are as follows:

Enable XInclude processing

If selected, XInclude references will be resolved when MSXML.NET is used as the transformer in the XSLT transformation scenario *(on page 669)*.

Validate documents during parse phase

If selected, 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 selected, the resolution is disabled.

Strip non-significant whitespaces

If selected, strips non-significant white space from the input XML document during the load phase. Selecting this option can lower memory usage and improve transformation performance while, in most cases, creating equivalent output.

Show time information

If selected, 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 is selected, 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 *(on page 139)* 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 *(on page 140)*, 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 *plugin* 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 *(on page 36)* and go to **XML > XSLT-XQuery > XSLT > Saxon > Saxon-HE/PE/EE**.

Saxon-HE/PE/EE Options

Oxygen XML Developer Eclipse plugin allows you to configure the following XSLT options for the Saxon 9.9.1.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE):

Use a configuration file ("-config")

Select this option if you want to use a Saxon 9.9.1.5 configuration file that will be executed for the XSLT transformation and validation processes. You can specify the path to the configuration file by entering it in the **URL** field, or by using the **Insert Editor Variables** button, or using the browsing actions in the **Browse** drop-down list.

Debugger trace into XPath expressions (applies to debugging sessions)

Instructs the XSLT Debugger *(on page 1227)* to *step into* XPath expressions.

Enable Optimizations ("-opt")

This option is selected by default, which means that optimization is enabled. If not selected, the optimization is suppressed, which is helpful when reducing the compiling time is important, optimization conflicts with debugging, or optimization causes extension functions with side-effects to behave unpredictably.

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")

Specifies 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")

Specifies 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.

Saxon-PE/EE Options

The following options are available for Saxon 9.9.1.5 Professional Edition (PE) and Enterprise Edition (EE) only:

Register Saxon-JS extension functions and instructions

Registers the Saxon-CE extension functions and instructions when compiling a stylesheet using the Saxon 9.9.1.5 processors.

Note: Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Developer Eclipse 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 selected, 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.

Enable assertions ("-ea")

In XSLT 3.0, you can use the `<xsl:assert>` element to make assertions in the form of XPath expressions, causing a dynamic error if the assertion turns out to be false. If this option is selected, XSLT 3.0 `<xsl:assert>` instructions are enabled. If it is not selected (default), the assertions are ignored.

Saxon-EE Options

The options available specifically for Saxon 9.9.1.5 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 allows for parsing the source documents with strict schema-validation enabled.
- **Lax schema validation ("lax")** - If an XML Schema is provided, this mode allows for 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. Selecting 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.

Saxon-HE/PE/EE Advanced Preferences

To configure the **Saxon HE/PE/EE Advanced** preferences, open the **Preferences** dialog box *(on page 36)* and go to **XML > XSLT-XQuery > XSLT > Saxon > Saxon-HE/PE/EE > Advanced**.

You can configure the following advanced XSLT options for the Saxon 9.9.1.5 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 9.9.1.5 transformer (the -r option when run from the command line). The class name must be fully specified and the corresponding `jar` or `class` extension must be configured from the dialog box for configuring the XSLT extension *(on page 695)* 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 9.9.1.5 transformer (the -cr option when run from the command line). The class name must be fully specified and the corresponding `jar` or `class` extension must be configured from the dialog box for configuring the XSLT extension *(on page 695)* for the particular transformation scenario.

Saxon6 Preferences

To configure the **Saxon 6** options, open the **Preferences** dialog box *(on page 36)* and go to **XML > XSLT-XQuery > XSLT > Saxon > Saxon6**.

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 selected, external function calls are not allowed. Selecting 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, 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 *(on page 36)* and go to **XML > XSLT-XQuery > XSLT > XSLTProc**.

The following options are available in this preferences page:

- **Enable XInclude processing** - If selected, XInclude references will be resolved when XSLTProc is used as transformer in XSLT transformation scenarios *(on page 669)*.
- **Skip loading the document's DTD** - If selected, the DTD specified in the DOCTYPE declaration will not be loaded.
- **Do not apply default attributes from document's DTD** - If selected, 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 selected, 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 selected, the transformation will output detailed status messages about the transformation process in the **Warnings** view.
- **Show version of libxml and libxslt used** - If selected, Oxygen XML Developer Eclipse plugin will display in the **Warnings** view the version of the **libxml** and **libxslt** libraries invoked by XSLTProc.
- **Show time information** - If selected, the **Warnings** view will display the time necessary for running the transformation.
- **Show debug information** - If selected, the **Warnings** view will display debug information about what templates are matched, parameter values, and so on.
- **Show all documents loaded during processing** - If selected, Oxygen XML Developer Eclipse plugin will display in the **Warnings** view the URL of all the files loaded during transformation.
- **Show profile information** - If selected, Oxygen XML Developer Eclipse 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 selected, Oxygen XML Developer Eclipse 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 selected, 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 selected, 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 regarding the **Outline** view *(on page 240)*, open the **Preferences** dialog box *(on page 36)* 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 Eclipse 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 Eclipse plugin includes several distinct layers of option values.

The option layers are as follows (sorted from high priority to low):

- **Global Options** *(on page 36)*

  Allows individual users to personalize Oxygen XML Developer Eclipse plugin according to their specific needs.
- **Customized Default Options** *(on page 145)*

  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 Eclipse plugin when an end-user selects the **Restore defaults** *(on page 37)* or **Reset Global Options** *(on page 146)* actions.
- **Default Options**

  The predefined default values, tuned so that Oxygen XML Developer Eclipse 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 Eclipse plugin has an extensive set of options that you can configure. When Oxygen XML Developer Eclipse plugin in installed, these options are set to default values. You can provide a different set of default values for an installation using an XML *options file*.

Creating an XML *Options File*

To create an *options file*, follow these steps:

1. It is recommended that you use a fresh install for this procedure, to make sure that you do not copy personal or local preferences.
2. Open Oxygen XML Developer Eclipse plugin and open the **Preferences** dialog box *(on page 36)*.
3. Go through the options and set them to the desired defaults.
4. Go to back to the main preferences page and click **Export Global Options** to create an XML options file.

Configuring an Installation to Use Customized Default Options

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

The possible methods for using customized default options during an installation include:

- **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/plugins/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 Eclipse 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/plugins/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 Eclipse plugin. To open this page, open the **Preferences** dialog box *(on page 36)*. 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 *(on page 145)*. This action also resets the transformation and validation scenarios to the default scenarios and clears recently used document 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 select 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.

Import/Export Transformation or Validation Scenarios

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 (on page 1325)* option files, or exported scenario files). To access these import and export actions, open the **Preferences** dialog box *(on page 36)* 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 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 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 Eclipse plugin includes a variety of built-in editor variables. You can also create your own custom editor variables *(on page 155)* by using the **Custom Editor Variables** preferences page *(on page 39)*.

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

- **${activeConditionSet}** - Current active profiling condition set name. If there is no active condition set, the variable will be replaced with an *empty string*.
- **${af}** - The local file path of the ZIP archive that includes the currently edited document.
- **${afd}** - The local directory path of the ZIP archive that includes the currently edited document.
- **${afdu}** - The URL path of the directory of the ZIP archive that includes the currently edited document.
- **${afn}** - The file name (without parent directory and without file extension) of the zip archive that includes the currently edited file.
- **${afne}** - The file name (with file extension, for example `.zip` or `.epub`, but without parent directory) of the zip archive that includes the currently edited file.
- **${afu}** - The URL path of the ZIP archive that includes the currently edited document.
- **${answer(@id)}** - Used in conjunction with the **${ask}** editor variable. The **@id** parameter is required and identifies the answer from the **${ask}** editor variable with the same ID.

  Example:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd">
  ```

```
<topic id="topic_lcf_1c4_tdb">
    <title></title>
    <body>
        <data name="${ask('Set a data name', String, 'name', @name)}"></data>
        <p>The name is: ${answer(@name)}</p>
    </body>
</topic>
```

- **${ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default_value', @id)}** - To prompt for values at runtime, use the *ask('message', type, ('real_value1':'rendered_value1'; 'real_value2':'rendered_value2'; ...), 'default-value')* editor variable. You can set the following parameters:
  - **'message'** - The displayed message. Note the quotes that enclose the message.
  - **'default-value'** - Optional parameter. Provides a default value.
  - **@id** - Optional parameter. Used for identifying the variable to reuse the answer using the **${answer(@id)}** editor variable.
  - **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 *relative_url* 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:** *${ask('message', url, 'default_value')}*
	Description: Input is considered a URL. Oxygen XML Developer Eclipse plugin checks that the provided URL is valid.
	Example: ▪ *${ask('Input URL', url)}* - The displayed dialog box has the name *Input URL*. The expected input type is URL. ▪ *${ask('Input URL', url, 'http://www.example.com')}* - The displayed dialog box has the name *Input URL*. The expected input type is URL. The input field displays the default value *http://www.example.com*.
password	**Format**: *${ask('message', password, 'default')}*
	Description: The input is hidden with bullet characters.
	Example:

Parameter	
	▪ *${ask('Input password', password)}* - The displayed dialog box has the name *'Input password'* and the input is hidden with bullet symbols. ▪ *${ask('Input password', password, 'abcd')}* - The displayed dialog box has the name *'Input password'* and the input hidden with bullet symbols. The input field already contains the default **abcd** value.
generic	**Format**: *${ask('message', generic, 'default')}*
	Description: The input is considered to be generic text that requires no special handling.
	Example: ▪ *${ask('Hello world!')}* - The dialog box has a *Hello world!* message displayed. ▪ *${ask('Hello world!', generic, 'Hello again!')}* - The dialog box has a *Hello world!* message displayed and the value displayed in the input box is *'Hello again!'*.
relative_url	**Format:** *${ask('message', relative_url, 'default')}*
	Description: Input is considered a URL. Oxygen XML Developer Eclipse plugin tries to make the URL relative to that of the document you are editing. **Note:** If the *$ask* editor variable is expanded in content that is not yet saved (such as an *untitled* file, whose path cannot be determined), then Oxygen XML Developer Eclipse plugin will transform it into an absolute URL.
	Example: *${ask('File location', relative_url, 'C:/example.txt')}* - The dialog box has the name *'File location'*. The URL inserted in the input box is made relative to the currently edited document location.
combobox	**Format:** *${ask('message', combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}*
	Description: Displays a dialog box that offers a drop-down menu. The drop-down menu is populated with the given *rendered_value* values. Choosing such a value will return its associated value (*real_value*). **Note:** The *'default'* parameter specifies the default-selected value and can match either a key or a value.
	Example:

Parameter	
	▪ *${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}* - The dialog box has the name *'Operating System'*. The drop-down menu displays the three given operating systems. The associated value will be returned based upon your selection. **Note:** In this example, the default value is indicated by the *osx* key. However, the same result could be obtained if the default value is indicated by *Mac OS X*, as in the following example: *${ask('Operating System', combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'Mac OS X')}* ▪ *${ask('Mobile OS', combobox, ('win':'Windows Mobile';'ios':'iOS';'and':'Android'), 'Android')}*
editable_combobox	**Format:** *${ask('message', editable_combobox, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}*
	Description: Displays a dialog box that offers a drop-down menu with editable elements. The drop-down menu is populated with the given *rendered_value* values. Choosing such a value will return its associated real value (*real_value*) or the value inserted when you edit a list entry. **Note:** The *'default'* parameter specifies the default-selected value and can match either a key or a value.
	Example: *${ask('Operating System', editable_combobox, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')}* - The dialog box has the name *'Operating System'*. The drop-down menu displays the three given operating systems and also allows you to edit the entry. The associated value will be returned based upon your selection or the text you input.
radio	**Format:** *${ask('message', radio, ('real_value1':'rendered_value1';...;'real_valueN':'rendered_valueN'), 'default')}*
	Description: Displays a dialog box that offers a series of radio buttons. Each radio button displays a *'rendered_value* and will return an associated *real_value*. **Note:** The *'default'* parameter specifies the default-selected value and can match either a key or a value.
	Example:

Parameter	
	${ask('Operating System', radio, ('win':'Microsoft Windows';'osx':'Mac OS X';'lnx':'Linux/UNIX'), 'osx')} - The dialog box has the name *'Operating System'*. The radio button group allows you to choose between the three operating systems. **Note:** In this example, `Mac OS X` is the default-selected value and if selected, it would return *osx* for the output.

- **${caret}** - The position where the cursor is located. This variable can be used in a code template, in **Author** mode operations, or in a **selection** *plugin*.
- **${cf}** - Current file as file path, that is the absolute file path of the currently edited document.
- **${cfd}** - Current file folder as file path, that is the path of the currently edited document up to the name of the parent folder.
- **${cfdu}** - Current file folder as URL, that is the path of the currently edited document up to the name of the parent folder, represented as a URL.
- **${cfn}** - Current file name without the extension and parent folder. The current file is the one currently open and selected.
- **${cfne}** - Current file name with extension. The current file is the one currently open and selected.
- **${comma}** - Used to display a comma when the actual comma symbol would be considered part of some sort of instruction or delimiter.
- **${configured.ditaot.dir}** - The default directory of the DITA Open Toolkit distribution, as configured in the **DITA** preferences page *(on page 47)*.
- **${cp}** - Current page number. Used to display the current page number on each printed page in the **Editor / Print** Preferences page.
- **${currentFileURL}** - Current file as URL, that is the absolute file path of the currently 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.
- **${dbgXML}** - The local file path to the XML document that is currently selected in the Debugger source combo box (for tools started from the XSLT/XQuery Debugger).
- **${dbgXSL}** - The local file path to the XSL/XQuery document that is currently selected in the Debugger stylesheet combo box (for tools started from the XSLT/XQuery Debugger).
- **${dita.dir.url}** - A special local contextual editor variable that gets expanded only in the **Libraries** dialog box that is accessible from the **Advanced** tab of DITA transformation scenarios. The **Libraries** dialog box allows you to specify additional libraries (*JAR (on page 1326)* files or additional class paths) to be used by the transformer. This *${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* 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 Eclipse plugin employs the following algorithm when searching for a given *framework* name:
  - All *frameworks* are sorted, from high to low, according to their **Priority** *(on page 52)* setting from the **Document Type** configuration dialog box *(on page 52)*. Only *frameworks* that have the **Enabled** checkbox selected 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 Eclipse plugin options.
    - Additional *frameworks* added in the **Locations** preferences page *(on page 51)*.
    - *Frameworks* installed using the add-ons support.
    - *Frameworks* found in the main *framework* location *(on page 51)* (**Default** or **Custom**).
- **${frameworkDir}** - The path (as file path) of the current *framework* directory.
- **${frameworks}** - The path (as URL) of the `frameworks` directory.
- **${frameworksDir}** - The path (as file path) of the `frameworks` directory.
- **${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 Eclipse plugin installation folder as URL.
- **${oxygenInstallDir}** - Oxygen XML Developer Eclipse plugin installation folder as file path.
- **${pd}** - The file path to the folder that contains the current project file (`.xpr`).
- **${pdu}** - The URL path to the folder that contains the current project file (`.xpr`).
- **${pluginDir(pluginID)}** - Each plugin has an ID specified in its `plugin.xml` file. This editor variable expands to the file path of the folder that contains the `plugin.xml` file where that specific plugin ID is located.

- **${pluginDirURL(pluginID)}** - Each plugin has an ID specified in its `plugin.xml` file. This editor variable expands to the URL path of the folder that contains the `plugin.xml` file where that specific plugin ID is located.
- **${pn}** - Current project name.
- **${ps}** - Path separator, which is the separator that can be used on the current platform (Windows, OS X, Linux) between library files specified in the class path.
- **${rootMapDir}** - Will be expanded to the current root map parent directory file path.
- **${rootMapDirURL}** - Will be expanded to the current root map parent directory URL.
- **${rootMapFile}** - Will be expanded to the current root map file path.
- **${rootMapURL}** - Will be expanded to the current root map URL. For example, if in the main DITA Map you define a key with a certain value:

  ```
  <keydef keys="test">
    <topicmeta><keywords><keyword>ABC</keyword></keywords></topicmeta>
  </keydef>
  ```

  you can modify a DITA-OT publishing parameter to have the value: `${xpath_eval(doc('${rootMapURL}')//keydef[@keys='test']/keywords/keyword/text())}`. It will be expanded to the value of that specified key name.
- **${selection}** - The currently selected text content in the currently edited document. This variable can be used in a code template, in **Author** mode operations, or in a **selection** *plugin*.
- **${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=var.value*. If you are looking for operating system environment variables, use the **${env(VAR_NAME)}** editor variable instead.
- **${timeStamp}** - The timestamp, which 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.
- **${uuid}** - Universally unique identifier, a unique sequence of 32 hexadecimal digits generated by the Java UUID class.
- **${xmlCatalogFilesList}** - A list of file paths that point to all known XML catalog files, separated by semi-colons (;).
- **${xpath_eval(expression)}** - Evaluates an XPath 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 *(on page 148)* 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 *(on page 39)*. To create a custom editor variable, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* 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 148)*

Custom System Properties

A variety of Java system properties can be set in the application to influence its behavior.

com.oxygenxml.disable.http.protocol.handlers

- **Allowed Values:** `true` or `false`
- **Default Value:** `false`
- **Purpose:** By default, Oxygen XML Developer Eclipse plugin uses the open source Apache HTTP Client software for HTTP(S) connections. If set to `True`, the default Java Sun HTTP(S) will be used instead. You will also lose **WebDAV** support and possibly other related features.

com.oxygenxml.present.license.reminders

- **Allowed Values:** `true` or `false`
- **Default Value:** `true`
- **Purpose:** When set to `false`, Oxygen XML Developer Eclipse plugin will not display the messages that remind you to renew your Support and Maintenance Pack that covers your current license.

com.oxygenxml.enable.content.reference.caching

- **Allowed Values:** `true` or `false`
- **Default Value:** `true`
- **Purpose:** Enables content reference caching.

com.oxygenxml.eclipse.remove.grid.editing.mode

- **Allowed Values:** `true` or `false`
- **Default Value:** `false`
- **Purpose:** When set to `false`, Oxygen XML Developer Eclipse plugin does not show the Grid editing mode when opening an XML document.

com.oxygenxml.default.java.accessibility

- **Allowed Values:** `true` or `false`
- **Default Value:** `false`
- **Purpose:** System property that can be set to `true` to force the default detection of java accessibility. If *com.sun.java.accessibility.AccessBridge* cannot be loaded, Oxygen XML Developer Eclipse plugin forces the Java accessibility to be disabled.

com.oxygenxml.floating.license.timeout

- **Allowed Values:** An integer (minutes)
- **Default Value:** `120`
- **Purpose:** Stores the time interval (in minutes) before floating licenses are released in case of application's inactivity.

com.oxygenxml.language

- **Allowed Values:** Language code (for example, `en-us`)
- **Default Value:** N/A
- **Purpose:** Property that holds the language code set during installation.

com.oxygenxml.default.options

- **Allowed Values:** A URL-type relative or absolute path.
- **Default Value:** N/A
- **Purpose:** Provides the path to an XML file containing default application options. For more details, see Customizing Default Options *(on page 145)*.

com.oxygenxml.customOptionsDir

- **Allowed Values:** A file system absolute path pointing to a folder.
- **Default Value:** N/A

- **Purpose:** Sets a folder to be used by the application to load and save preference files. The default location where the options are saved varies according to the operating system. For more details, see Importing/Exporting/Resetting Global Options *(on page 146)*.

com.oxygenxml.ApplicationDataFolder **(Windows only)**

- **Allowed Values:** A file system absolute path pointing to a folder.
- **Default Value:** `%APPDATA%`
- **Purpose:** When the application runs on Windows, you can set this property to change the location where the application considers that the *APPDATA* folder is located.

com.oxygenxml.editor.frameworks.url

- **Allowed Values:** A URL-type absolute path.
- **Default Value:** `OXYGEN_DIR \frameworks`
- **Purpose:** Changes the folder where the application considers that the main *frameworks* are installed. It has the same effect as changing the custom *frameworks* directory value in the **Location** preferences page *(on page 51)*.

com.oxygenxml.MultipleInstances

- **Allowed Values:** `true` or `false`
- **Default Value:** `false`
- **Purpose:** If set to `true`, multiple instances of the application are allowed to be started.

com.oxygenxml.xep.location

- **Allowed Values:** A file system absolute path pointing to a folder.
- **Default Value:** N/A
- **Purpose:** Points to a folder where RenderX XEP is installed. Has the same effect as configuring XEP in the **FO Processors** preferences page *(on page 115)*.

com.oxygenxml.additional.classpath

- **Allowed Values:** A list of *JAR (on page 1326)*-type resources separated by a classpath separator.
- **Default Value:** N/A
- **Purpose:** An additional list of libraries to be used in the application's internal class loader in addition to the libraries specified in the `lib` folder.

com.oxygenxml.user.home **(Windows only)**

- **Allowed Values:** A file system absolute path pointing to a folder.
- **Default Value:** `USERPROFILE` folder

- **Purpose:** Overwrites the user home directory that was implicitly detected for the application.

com.oxygenxml.use.late.delegation.for.author.extensions

- **Allowed Values:** `true` or `false`
- **Default Value:** `true`
- **Purpose:** All Java extensions in a *framework* configuration are instantiated in a separate class loader. When **true**, the *JAR* libraries used in a certain document type will have priority to resolve classes before delegating to the parent class loader. When **false**, the parent class loader will take precedence.

com.oxygenxml.stack.size.validation.threads

- **Allowed Values:** The number of bytes used for validation threads.
- **Default Value:** `5*1024*1024`
- **Purpose:** Some parts of the application (validation, content completion) that use the Relax NG parser sometimes require a larger *Thread* stack size to parse complex schemas. The default value should be more than enough.

com.oxygenxml.jing.skip.validation.xhtml.data.attrs

- **Allowed Values:** `true` or `false`
- **Default Value:** `true`
- **Purpose:** By default, the Relax NG validation was configured to skip validation for XHTML attributes that start with "data-", which should be skipped from validation according to the XHTML 5 specification.

com.oxygenxml.report.problems.url

- **Allowed Values:** User-defined URL
- **Default Value:** N/A
- **Purpose:** The URL where a problem reported through the **Report Problem** dialog box is sent. The report is sent in XML format using the **report** parameter with the POST HTTP method.

com.oxygenxml.parallel.title.computing.threads

- **Allowed Values:** Integers
- **Default Value:** `4`
- **Purpose:** The number of parallel threads that will be used to compute referenced topic titles. Increasing this value reduces the amount of time it takes to compute topic titles in the **DITA Maps Manager** view.

com.oxygenxml.hidpi.scaling

- **Allowed Values:** Numerical values between 1 and 2 (1, 1.5, and 2 have been tested, and for example, 1.5 is for 150% scaling)
- **Default Value:** N/A
- **Purpose:** Used to override the HiDPI scaling detection to force a specific scaling setting. This is helpful if you encounter scaling detection issues in Windows or Linux.

com.oxygenxml.prefer.plugin.classloader.context.loader

- **Allowed Values:** `true` or `false`
- **Default Value:** `true`
- **Purpose:** Used to instruct the application to use the plugin class loader when there is code that loads content (usually Xerces code) using the thread's class loader. For instance, if you have a plugin that specifies a certain Xerces version and you want to load that version instead of the one from **Oxygen's** `lib` directory.

com.oxygenxml.classic.file.output.stream.save

- **Allowed Values:** `true` or `false`
- **Default Value:** `false`
- **Purpose:** When set to `true`, the files are saved using a Java classic file output stream, which destroys the NTFS alternate data streams set on the file. However, this might prevent data loss in the rare occasions when Oxygen XML Developer Eclipse plugin saves empty file content over shared network drives.

com.oxygenxml.disable.correct.over.ascii.chars

- **Allowed Values:** `true` or `false`
- **Default Value:** `false`
- **Purpose:** By default, Oxygen XML Developer Eclipse plugin will escape non-ASCII characters (encode them with their hexadecimal equivalent) within URL paths. If set to `true`, non-ASCII characters in URL paths will not be escaped. This is helpful if you are using a non-Latin alphabet (such as Arab, Japanese, Chinese) since the URL paths will remain unchanged and will be more readable.

Localizing of the User Interface

To localize the Oxygen XML Developer Eclipse plugin, you can use one of the following methods:

- **Localization through the update site:**

  Start Eclipse, go to **Help > Install New Software**. Click **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. Click 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 Eclipse 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 Eclipse 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 Eclipse plugin plugin installation directory, translate all the keys in the file and change its name to `plugin_<locale>.properties`.

5.

Perspectives

An Oxygen XML Developer Eclipse plugin *perspective (on page 1327)* is an interface layout geared towards a specific use. The Oxygen XML Developer Eclipse 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 Eclipse plugin. You can change the *perspective* by selecting the respective icon in the top-right corner of Oxygen XML Developer Eclipse plugin or by selecting the *perspective* from the **Window > Perspective > Open Perspective** menu.

Oxygen XML Perspective

The **Oxygen XML** *perspective (on page 1327)* is the most commonly used *perspective* and it is the default *perspective* when you start Oxygen XML Developer Eclipse 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 Eclipse plugin. Most of the menus are common for all types of documents. However, Oxygen XML Developer Eclipse plugin also includes some context-sensitive and *framework (on page 1325)*-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 Eclipse plugin includes a large variety of *dockable (on page 1324)* 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 toggle controls located on the top corner of the view (▭).

Some of the most helpful views in the **Oxygen XML** *perspective* include the following:

- **Navigator** view *(on page 192)* - Enables the definition of projects and logical management of the documents they contain.
- **Outline** view *(on page 240)* - 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** *(on page 248)* - Displays the messages generated as a result of user actions such as validations *(on page 277)*, transformation scenarios *(on page 669)*, spell checking in multiple files *(on page 214)*, search operations, and others. Each message is a link to the location related to the event that triggered the message.
- **Attributes** view *(on page 243)* - 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 *(on page 245)* - Presents the currently edited element structure model and additional documentation as defined in the schema.
- **Elements view** *(on page 247)* - Presents a list of all defined elements that you can insert at the current cursor position according to the document's schema.
- **Entities** view *(on page 247)* - Displays a list with all entities declared in the current document as well as built-in ones.
- **Transformation Scenarios** view *(on page 770)* - Displays a list with all currently configured transformation scenarios.
- **XPath/XQuery Builder** view *(on page 1119)* - Displays the results from running an XPath expression.
- **Text** view *(on page 702)* - Displays the text output that is produced in XSLT transformations.
- **Browser** view *(on page 702)* - 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 *(on page 220)* - Status information generated by the Schema detection, validation, and transformation threads.
- **WSDL SOAP Analyzer** view *(on page 519)* - 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:

Editing Documents *(on page 222)*

Editing Modes *(on page 168)*

Oxygen XSLT Debugger Perspective

The **XSLT Debugger** *perspective (on page 1327)* allows you to detect problems in an XSLT transformation by executing the process step by step in a controlled environment. 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 *(on page 131)* 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 *(on page 168)* 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 *(on page 1215)*

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 *(on page 1215)* 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).

For more information about the XSLT debugging capabilities in Oxygen XML Developer Eclipse plugin, watch our video demonstration:

https://www.youtube.com/embed/m9d8c4V-LJw

Related Information:

Debugging XSLT Stylesheets and XQuery Documents *(on page 1210)*

Oxygen XQuery Debugger Perspective *(on page 164)*

Oxygen XQuery Debugger Perspective

The **XQuery Debugger** *perspective (on page 1327)* 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 *(on page 131)* 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 *(on page 168)* 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 *(on page 1215)*

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 *(on page 1215)* 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).

For more information about the XQuery debugging capabilities in Oxygen XML Developer Eclipse plugin, watch our video demonstration:

https://www.youtube.com/embed/o5_M2kbyipU

Related Information:

Debugging XSLT Stylesheets and XQuery Documents *(on page 1210)*

Oxygen XSLT Debugger Perspective *(on page 162)*

Oxygen DB Perspective

The **Database** *perspective (on page 1327)* 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 Eclipse 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)
- 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 Eclipse plugin. For a detailed feature matrix that compares the Academic, Professional, and Enterprise editions of Oxygen XML Developer Eclipse plugin go to the Oxygen XML Developer Eclipse 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 *(on page 1133)*.
- 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 Eclipse 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 1135)*.

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 Eclipse 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 Eclipse 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 ***(on page 1131)***

Provides browsing support for the configured connections.

Table Explorer View ***(on page 1133)***

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

Related Information:

Working with Databases *(on page 1131)*

Data Source Explorer View *(on page 1131)*

Table Explorer View *(on page 1133)*

6.
Editing Modes

The main editing area in Oxygen XML Developer Eclipse 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 Eclipse plugin offers the following editing modes:

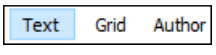
- **Text** *(on page 168)* - This mode presents the source of an XML document.
- **Grid** *(on page 168)* - This mode displays an XML document as a structured grid of nested tables.
- **Design** *(on page 169)* - This mode is found in the schema editor and represents the schema as a diagram.

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 *(on page 85)* is selected in the **Edit Modes** preferences page, then the edit mode specified in the **Document Type** configuration dialog box *(on page 52)* 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 *(on page 85)* is not selected, then the edit mode specified in the table in the **Edit Modes** preferences page *(on page 85)* is used when that particular type of document is initially opened.

Text Editing Mode

The **Text** mode editor in Oxygen XML Developer Eclipse 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 Eclipse plugin also includes specialized editing actions, a powerful *Content Completion Assistant (on page 234)*, and many other unique features.

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

For more information about working with XML documents in **Text** mode and all of the details about its features, see the **Editing XML Documents in Text Mode** section *(on page 222)*.

Related Information:

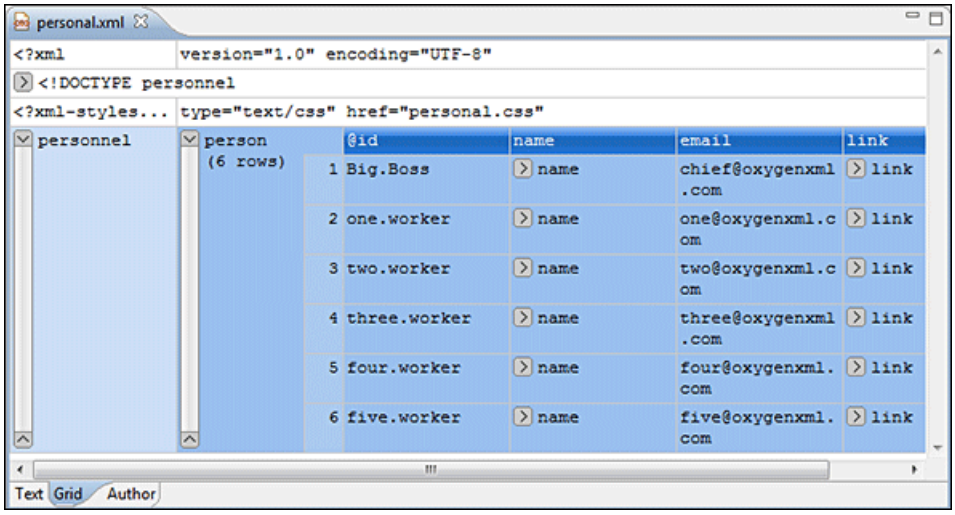
Editing XML Documents in Text Mode *(on page 222)*

Grid Editing Mode

The Oxygen XML Developer Eclipse 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 32. Grid Editing Mode



For more information about working with XML documents in **Grid** mode and all of the details about its features, see the **Editing XML Documents in Grid Mode** section *(on page 268)*.

Related Information:

Editing XML Documents in Grid Mode *(on page 268)*

Design Editing Mode (XML Schema Diagram Editor)

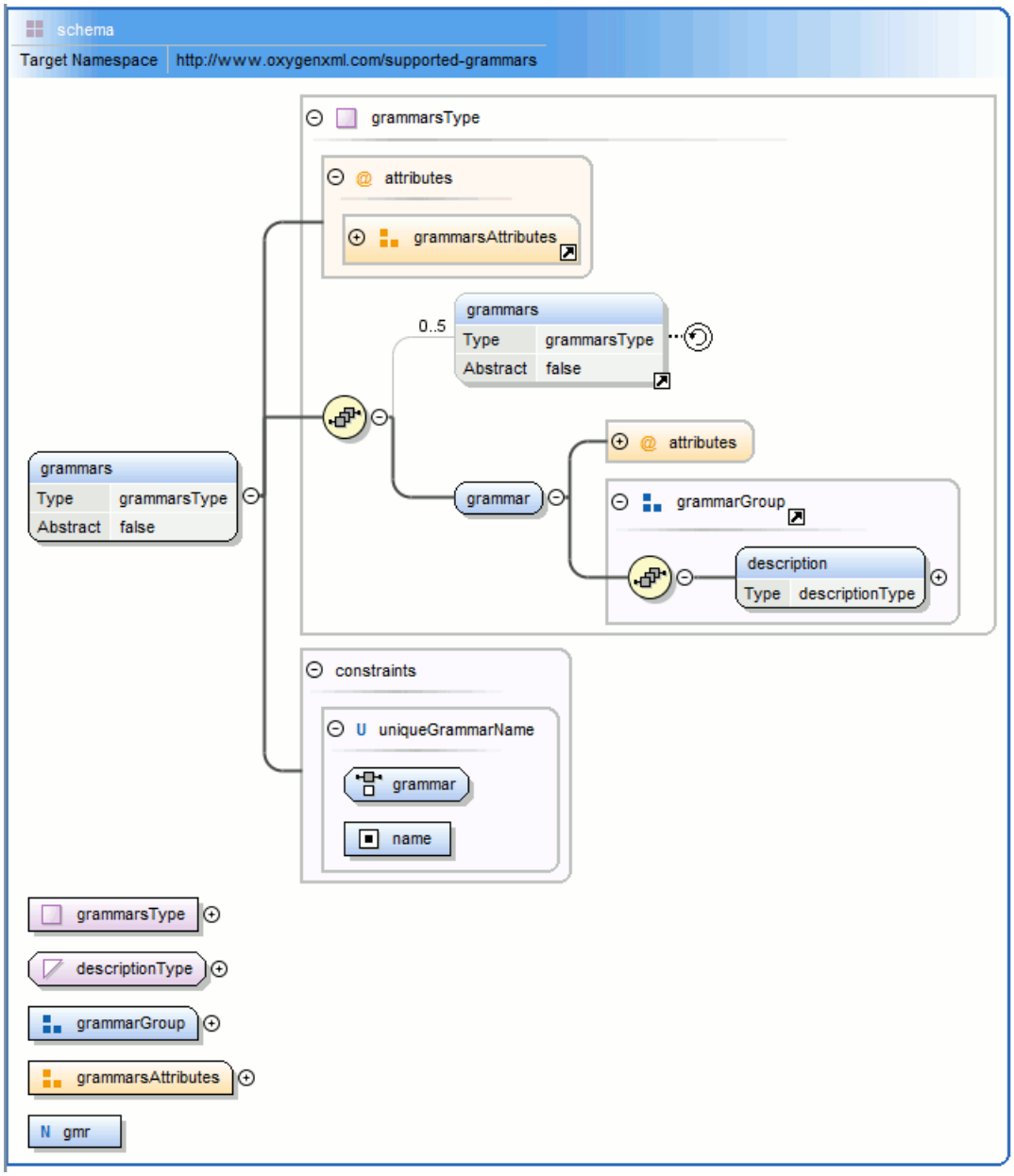
XML Schemas allow document designers to specify the allowed structure and content of an XML document and to check if an XML document is valid. Oxygen XML Developer Eclipse plugin provides a simple and expressive XML Schema diagram editor (**Design** mode) for editing XML Schemas. The schema diagram helps both the content authors who want to understand a schema and schema designers who develop complex schemas.

The **Design** mode offers a diagram view of the XML Schema document by rendering all the XML Schema components. You can edit XML Schema component features directly within the diagram (for instance, the component name, its type, etc.), you can quickly navigate to the referenced definitions (elements, attributes, types, groups, etc.), and you can use drag-and-drop operations to move, copy, or make references. It also features some specialized helper views (such as the **Palette** view *(on page 408)* and **Facets** view *(on page*

409)) to further enhance the diagram editor, various contextual menu actions, validation support, and much more.

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

Figure 33. XML Schema Diagram



For more information about designing and editing XML Schemas, and all the details about the features that are available in the **Design** mode, see the **Editing XML Schemas** section *(on page 406)* and the **Working with the XML Schema Design Mode (XML Schema Diagram Editor)** subsection *(on page 406)*.

Related Information:

Editing XML Schemas *(on page 406)*

Working with the XML Schema Design Mode (XML Schema Diagram Editor) *(on page 406)*

7.

Working With Any Type of Document

Oxygen XML Developer Eclipse plugin includes a variety of general features that can be used when working with any type of document. More specialized features are available when working with specific type of documents, such as the various types of XML documents, CSS, JavaScript, Markdown, and more. For details about those specialized features for specific types of documents, see Editing Documents *(on page 222)*.

This chapter includes information about how to create and work with documents, working with projects, and various editing features that are provided in Oxygen XML Developer Eclipse plugin for all document types. This chapter also includes information about some of the special tools that are included in Oxygen XML Developer Eclipse plugin, such as the file and directory comparison tools.

Creating, Opening, Saving, and Closing Documents

Oxygen XML Developer Eclipse 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 Eclipse plugin includes a helpful **New Document** wizard that allows you to customize and create new files from a large list of document types and built-in templates. You can also create your own templates *(on page 178)* and share them with others *(on page 182)*.

To quickly create a new document:

1. Click the **New** button on the toolbar or select **File > New > Other > Oxygen XML Developer Eclipse plugin**.
2. Select the type of document that you want to create.

   Tip: You can use the text filter field at the top of the dialog box to search for a specific template.
3. Click the **Next** button, then **Finish**.

New Document Wizard

Oxygen XML Developer Eclipse 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 creates a skeleton document that may contain a root element, the document prolog, and possibly other child elements depending on options that are specific for each schema type. The wizard also

provides access to the **New from Templates** option *(on page 177)* that opens a wizard where you can create a document based upon built-in templates or custom templates.

New Document Wizard

To create a new document in Oxygen XML Developer Eclipse plugin, follow these steps:

1. Click the **New** button on the toolbar or select **File > New > Other > Oxygen XML Developer Eclipse plugin**.

   Result: The **New Document** wizard is displayed with all the supported document types.
2. Select the type of document that you want to create. Oxygen XML Developer Eclipse plugin includes a series of Eclipse wizards that help you create the new document based upon the type you choose.

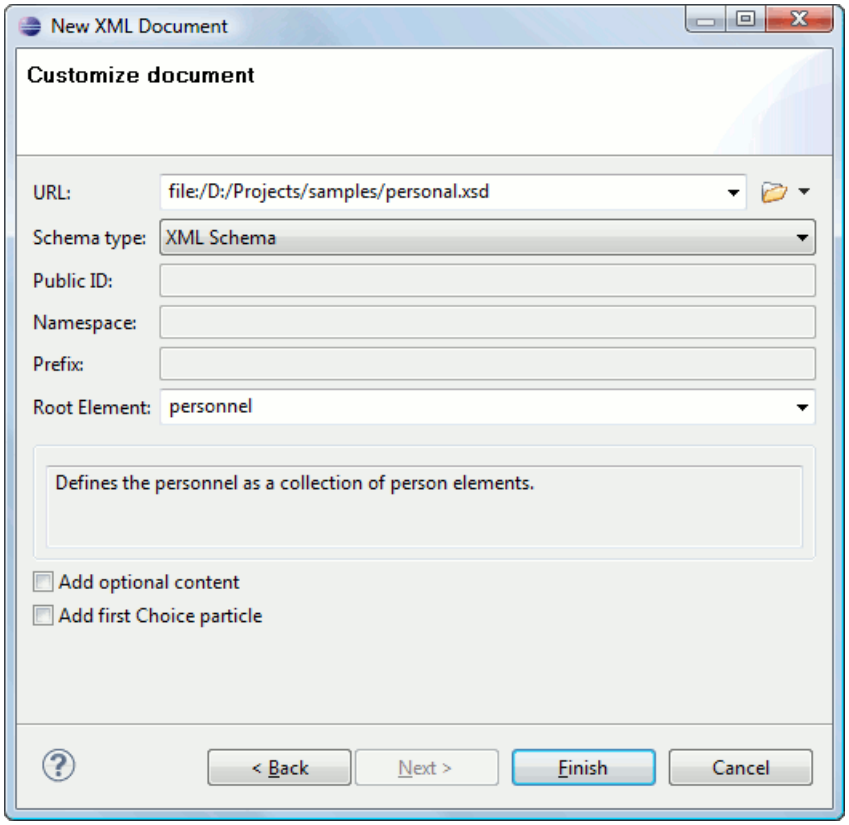
   Tip: You can use the text filter field at the top of the dialog box to search for a specific template.
3. Click **Next**.

   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.

   Note: For DITA documents, the dialog box includes some additional options for generating a title, file name, and root ID attribute.
4. After configuring the options for the particular type of document, 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.

New XML Document Wizard

Figure 34. 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 Eclipse 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

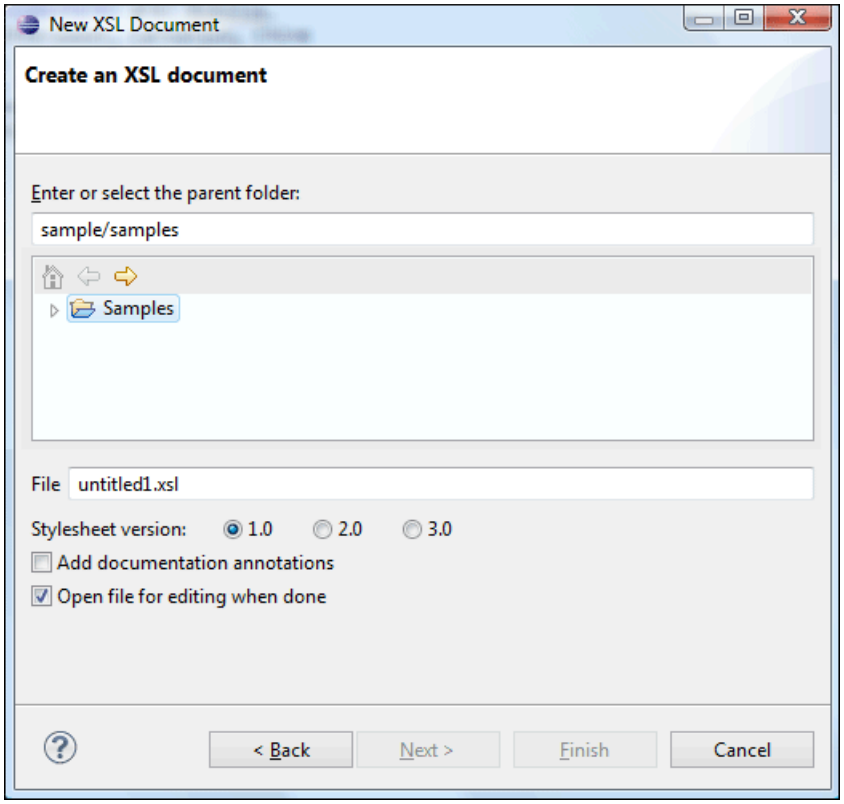
If you select this option, the elements and attributes defined in the XML Schema as optional are generated in the skeleton XML document.

Add First Choice Particle

If you select this option, Oxygen XML Developer Eclipse plugin generates the first element of an `<xs:choice>` schema element in the skeleton XML document. Oxygen XML Developer Eclipse plugin creates this document in a new editor panel when you click **Finish**.

XSL Document Wizard

Figure 35. 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

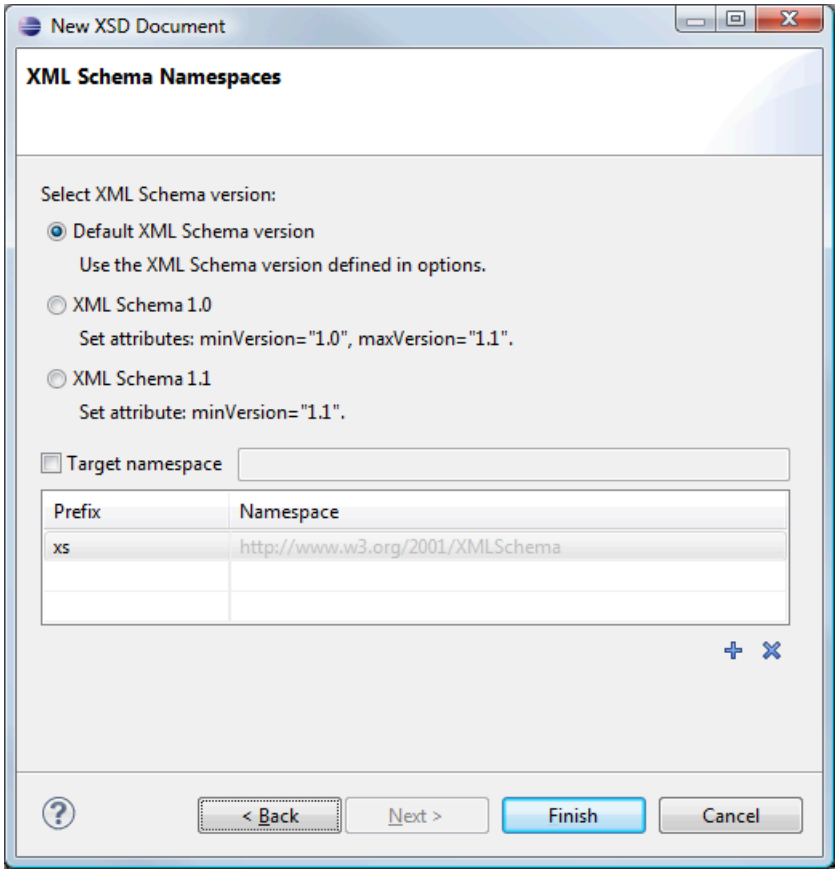
Allows you to select the **Stylesheet version** number. You can select from: `1.0`, `2.0`, and `3.0`.

Add documentation annotations

Select this option to generate the stylesheet annotation documentation.

XML Schema (XSD) Document Wizard

Figure 36. 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 *(on page 126)*.

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.

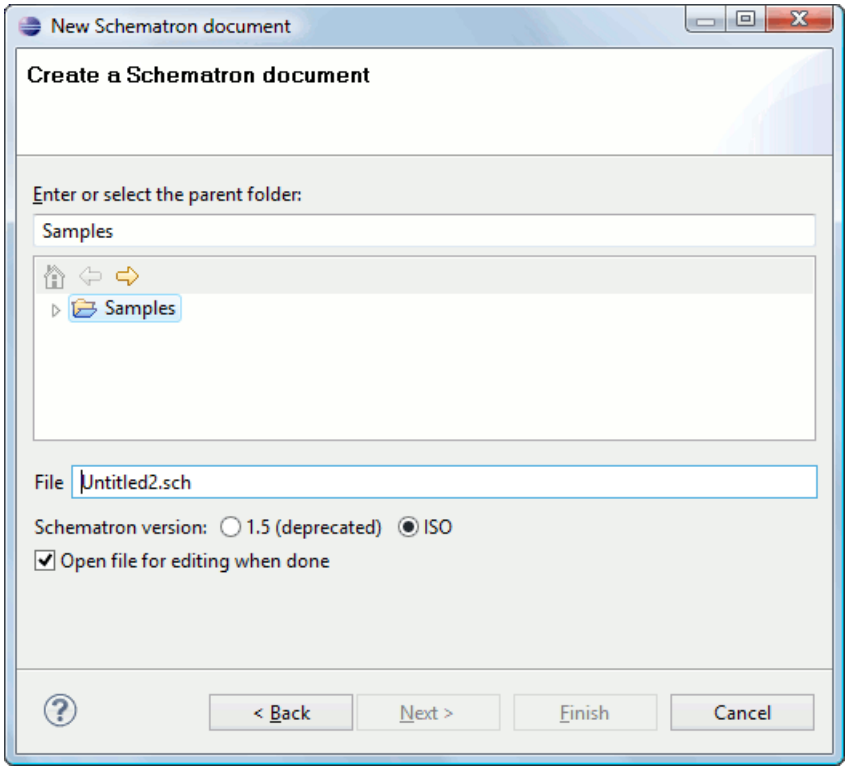
Namespace prefix declaration table

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 *(on page 484)*.

Schematron Document Wizard

Figure 37. 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:

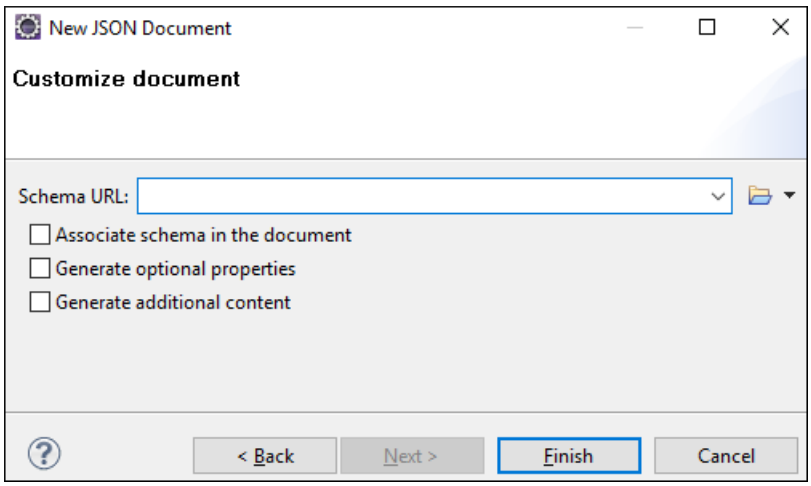
Schematron version

Specifies the Schematron version. Possible options: `1.5` (deprecated) and `ISO`.

Note: Starting with version 16.0 of Oxygen XML Developer Eclipse plugin, the support for Schematron 1.5 is deprecated. It is recommended to use ISO Schematron instead.

JSON Document Configuration Page

Figure 38. New JSON Configuration Wizard Page



If you select **JSON** for the type of file you want to create and select the **Customize** option, the configuration dialog box will include the following options:

Schema URL

Specifies the path to a JSON Schema file that will be used to generate key-value pairs.

Associate Schema in the Document

If you select this option, the JSON instance will be generated with the JSON Schema associated directly in the document.

Generate Optional Properties

If you select this option, the JSON instance will be generated with optional properties that are defined in the JSON schema. Otherwise, only the required properties will be generated.

Generate Additional Content

If you select this option, the JSON instance will be generated with additional properties that are defined in the JSON schema as `additionalProperties` and additional items that are defined as `additionalItems` (in the case of an Array).

Creating New Documents Based on Templates

The **New from Templates** wizard *(on page 171)* allows you to select built-in templates or custom templates that you or other users created in previous sessions. You can access this wizard by selecting the **New from Templates** option in the **New Document** wizard *(on page 171)*.

The categories of templates presented in the wizard include:

- **User-defined template directory** - You can add your own custom templates by creating template files *(on page 178)* in a directory and then add that directory to the list of template directories that Oxygen XML Developer Eclipse plugin uses in the **Document Templates** preferences page *(on page 107)*. This user-defined directory will appear in the **New from templates** wizard.
- **Global templates** - Contains a list of built-in templates as well as any user-defined custom templates *(on page 178)* that are saved in the `templates` directory of the Oxygen XML Developer Eclipse plugin installation folder (*`[OXYGEN_INSTALL_DIR]`*`/templates`).
- **Framework templates** - Contains the list of templates defined in the **Document Type** configuration dialog box (**Templates** tab) *(on page 72)* for each *framework (on page 1325)*.

New from Templates Wizard

To create a new document using this wizard, follow these steps:

1. Click the **New** button on the toolbar and select **New from Templates** (or select **File > New > Other > Oxygen XML Developer Eclipse plugin > New From Templates**).

   Result: The **New from Templates** wizard is displayed where 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 type of editor.

Creating New Document Templates

Oxygen XML Developer Eclipse plugin allows you to create your own custom document templates and they will appear in the **New from templates** wizard *(on page 177)*.

Creating a New Document Template

To create your own custom document template and have it appear in the new document 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 *(on page 148)* in the template file content and they will be expanded when the files are opened. Also, see Customizing Document Templates *(on page 179)* for other template customization tips (for example, you could add placeholders or hints *(on page 181)* to assist authors).
2. Save the new document template and reference that location in Oxygen XML Developer Eclipse plugin. There are several options for doing this:
   - **Saving the new template in a specific framework's directory** - Save the new template in a directory (for example, called `templates`) within that specific framework directory. Then open the **Document Type** configuration dialog box *(on page 52)* for that specific framework, go to the **Templates** tab *(on page 72)*, and click the + button in the bottom-right corner to add your new directory to the list. It is recommended that the reference be made relative to the framework directory (for example, `${frameworkDir}/templates`). You can also remove any existing entries in the list that aren't applicable or won't be used in your custom framework. Click **OK** to close the configuration dialog box and then **OK** or **Apply** to save your changes.
   - **Saving the new template in the Oxygen installation directory** - Save the new template in the `templates` directory of the Oxygen XML Developer Eclipse plugin installation directory (*`[OXYGEN_INSTALL_DIR]`*`/templates`). Document templates saved in this directory will appear in the **Global templates** category in the **New from templates** wizard *(on page 177)*.
   - **Saving the new template in a custom directory** - Save the new template in any directory of your choice and then add that directory to the list of templates in the **Document Templates** preferences page *(on page 107)*. This user-defined directory will appear in the **New from templates** wizard *(on page 177)* along with all the new document templates that you save inside it. Click **OK** or **Apply** to save your changes.

Tip: If you want to create a new template for a binary file (e.g. a zip archive), you need to add *.bin* to the end of the file name (for example, **.zip.bin* or **.epub.bin*). Otherwise, the files will be treated as XML/text documents and you will be prompted to choose the editor type.

Attention: The name that you use to save the template will be the name that appears in the new document wizard, including capitalization, space, and characters (for example, `My Custom Template1.xml` will appear in the new file wizard as **My Custom Template1**). You can also configure the displayed name in a properties file by following the procedure found in the Configure the Displayed Names for Document Templates *(on page 181)* section.

3. Open the new document wizard (**New** toolbar button or **File > New > New from Templates**) and you should see your custom template in the appropriate folder.

Note: For DITA templates, they will also appear in the dialog box for creating new DITA topics, but if you customize the template *(on page 179)*, you need to set the `type` property to **dita** in the corresponding properties file.

Related Information:

Customizing Document Templates *(on page 179)*

Sharing Custom Document Templates *(on page 182)*

Customizing Document Templates

Oxygen XML Developer Eclipse plugin allows you to customize certain aspects of built-in 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 *(on page 177)*.

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 following these guidelines:
   a. If you want to create a new properties file, you can use the **Properties** template found in the **New Document** folder in the **New from templates** wizard *(on page 177)*. If you want to edit an existing template, you can find them within the subfolders in the `templates` folder for each framework (for example, the DITA topic properties file is located in: `OXYGEN_INSTALL_DIR/frameworks/dita/templates/topic/topic.properties`).
   b. Use the same name as your custom template file except with a `.properties` extension (for example, `MyTemplate.properties`).
   c. In this properties file, specify the paths to the icons that will be used in the new file wizard. The properties file should look like this:

```
type=general
smallIcon=../icons/Article_16.png
```

```
bigIcon=../icons/Article_48.png
```

Tip: 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.

Important: 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 *(on page 178)*), make sure the path to the icons is relative to that directory.

2. Save the properties file in the same directory as your custom 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 click the **New** button on the toolbar (or **File > New > Other > Oxygen XML Developer Eclipse plugin**). The prefix or suffix is added to the name of the file in the **File** field.
- 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 *(on page 192)*. 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 *(on page 180)*.
2. Save the properties file in the same directory as the document template.

3. Open the new document wizard (using the methods described above *(on page 180)*) 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.

Configure the Displayed Names for Document Templates

To change the name that is displayed for a document template, use 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`).
2. Add a `displayName` property in the properties file:

   ```
   displayName=My Template Name
   ```

   Tip: The names for *framework (on page 1325)*-specific document templates (such as DITA *Topic* or DocBook *Article*, as you would see in the **Framework templates** folder in the **New** file wizard) can be translated via the internationalization support. In this case, the properties file should contain something like:

   ```
   displayName=${i18n(tag)}
   ```

   where *tag* refers to an entry in the `translation.xml` file for that specific framework (for example, `OXYGEN_INSTALL_DIR/frameworks/dita/i18n/translation.xml` for DITA).
3. Save the properties file in the same directory as the document template.
4. Open the new file wizard (**File > New > New from Templates**) and you should see the new name for the template.

Adding Placeholders or Hints in a Document Template

A document template might contain empty elements and it may not be clear to the Author what should be inserted. You can define placeholders in document templates that provide hints for Authors to help them understand what type of content should be added in any particular empty element within the document. The placeholder text is specified using a processing instruction and the placeholders are removed when the Author inserts content in the corresponding element.

To define placeholders in a document template to provide authors with hints, follow this procedure:

1. Edit the document template.
2. Add placeholders in the form of processing instructions within the elements where you want hints to be displayed when an Author creates a document from the template. For example:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd">
   <topic id="pi">
     <title><?oxy-placeholder content="Enter a title"?></title>
       <shortdesc><?oxy-placeholder content="Writing short descriptions
            induces the writer to clarify the main thesis of the topic.
            We recommended a 50 word limit."?></shortdesc>
   ```

```
  <body>
    <p><?oxy-placeholder content="A paragraph element should be a self-contained
         unit dealing with one idea or point."?></p>
  </body>
</topic>
```

3. Save the template file.
4. Use the **New from templates** wizard *(on page 177)* to create a new document using your customized template and you should see the hints in the open document.

Related Information:

Creating New Document Templates *(on page 178)*

Sharing Custom Document Templates *(on page 182)*

Sharing Custom Document Templates

Your custom document templates *(on page 178)* can be shared with the other members of your team so that they all have access to the templates in the **New from templates** wizard *(on page 177)*. The best way to share them is by integrating them in an extended *framework (on page 1325)* (document type) configuration and then sharing the whole framework with the other users.

Sharing Custom Document Templates

To share custom document templates with other members of your team:

1. Create a custom framework by extending an existing one, if you haven't already done so.
2. Create the new document template *(on page 178)*, if you haven't already done so.
3. Save the new template in a directory (for example, called `templates`) within your custom framework directory. Then open the **Document Type** configuration dialog box *(on page 52)* for that specific framework, go to the **Templates** tab *(on page 72)*, and click the + button in the bottom-right corner to add your new directory to the list. It is recommended that the reference be made relative to the framework directory (for example, `${frameworkDir}/templates`). You can also remove any existing entries in the list that aren't applicable or won't be used in your custom framework.
4. Click **OK** to close the configuration dialog box and then **OK** or **Apply** to save your changes.
5. All that remains is to share the entire framework with anyone who needs to have access to the custom templates.

Opening Documents

To open a document in Oxygen XML Developer Eclipse 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 where you can specify a URL (defined by a protocol, host, resource path, and an optional port) or use the browsing actions in the **Browse for remote file** drop-down menu.
- Select the **Open** or **Open with** action from the contextual menu of the **Navigator** view *(on page 192)*.

Opening the Current Document in a 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.

Closing Documents

To close open documents, you can simply click the close icon (×) for the particular editor tab or use one of the following actions that are available by right-clicking 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.

Working with Remote Documents

Oxygen XML Developer Eclipse 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 *(on page 184)*

A WebDAV resource can be locked when it is opened in Oxygen XML Developer Eclipse plugin by selecting the **Lock WebDAV files on open** option *(on page 110)* to prevent other users to modify it concurrently on the server. If a user tries to edit a locked file, Oxygen XML Developer Eclipse 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 Eclipse plugin.

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

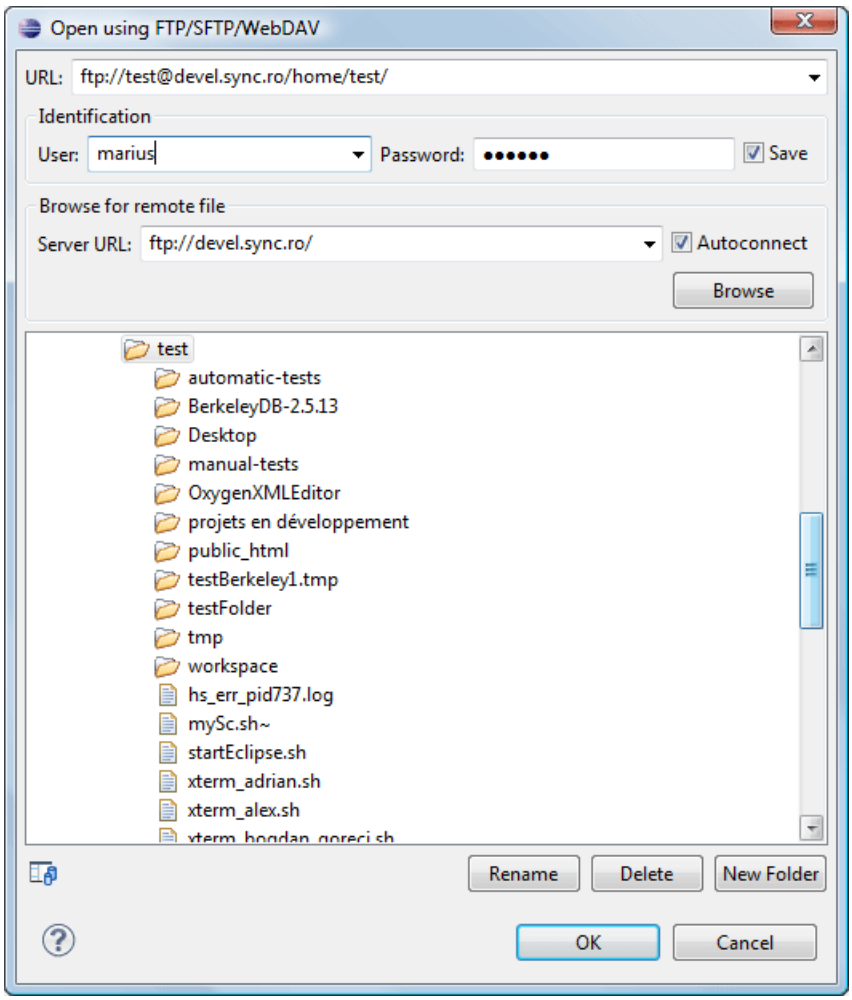
To improve the transfer speed, the content exchanged between Oxygen XML Developer Eclipse plugin and the HTTP / WebDAV server is compressed using the GZIP algorithm.

The current WebDAV Connection *(on page 1181)* details can be saved by switching to the **Database** *perspective (on page 1327)* and then you can browse and manage the connection in the **Data Source Explorer** view *(on page 1131)*.

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 39. Open URL Dialog Box



The displayed dialog box is composed of several parts:

- The editable **URL** combo box where 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** checkbox. In the **Server URL** combo box, you can specify the protocol, the server host name, or server IP.

Tip: When accessing an 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 checkboxes.

WebDAV over HTTPS

If you want to access a WebDAV repository across a non-secure network, Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin. This means that Oxygen XML Developer Eclipse plugin can find the certificate in the key store of the Java Runtime Environment where 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 Eclipse 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 Eclipse 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* that comes bundled with Oxygen XML Developer Eclipse plugin.

Tip: To make Oxygen XML Developer Eclipse plugin accept a certificate even if it is invalid, open the **Preferences** dialog box *(on page 36)*, go to **Network Connection settings > HTTP(S)/WebDAV**, and select the **Automatically accept a security certificate, even if invalid** option.

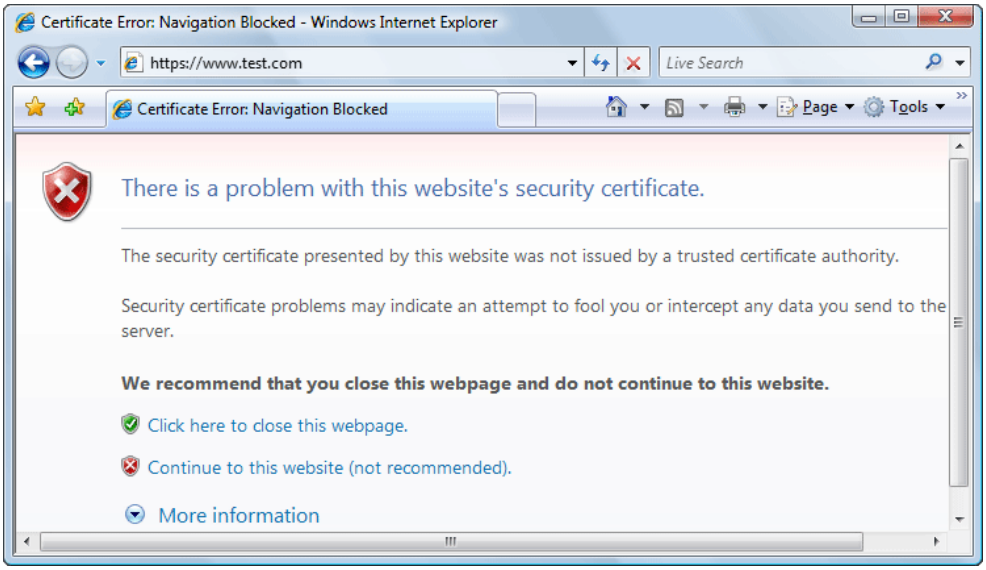
Note: This **Automatically accept a security certificate, even if invalid** option does not influence the entire Eclipse workspace. It is limited to URLs that are opened directly by Oxygen XML Developer Eclipse plugin.

1. Export the certificate into a local file

   a. Point your HTTPS-aware Web browser to the repository URL.

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

   Figure 40. Security alert - untrusted certificate

   b. Go to **Tools > Internet Options**.
   The **Internet Options** dialog box is opened.

   c. Select the **Security** tab.

   d. Select the **Trusted sites** icon.

   e. Click the **Sites** button.
   This will open the **Trusted sites** dialog box.

   f. Add the repository URL to the **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 the **Continue to this website** option.
A clickable area with a red icon and the text **Certificate Error** is added to the Internet Explorer address bar.

j. Click the **Certificate Error** area.
A dialog box that contains a **View certificates** link is displayed.

k. Click the **View certificates** link.
Certificate dialog box is displayed.

l. Select the **Details** tab of the **Certificate** dialog box.

m. Click the **Copy to File** button.
The **Certificate Export Wizard** is started.

n. Follow indications of the wizard for the DER encoded binary X.509 certificate. Save the certificate to the local file `server.cer`.

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

a. Open a text-mode console with administrative rights.
If Oxygen XML Developer Eclipse 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 Eclipse plugin.
You can find the home directory of the JRE in the *java.home* property that is displayed in the **About** dialog box (**Installation Details > Configuration**).
On OS X, for the distribution of Oxygen XML Developer Eclipse 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. The *keytool* requires a password before adding the certificate to the JRE *keystore (on page 1326)*. 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 Eclipse plugin.

Related Information:

HTTP(S)/WebDAV Preferences *(on page 110)*

HTTP Authentication Schemes

Oxygen XML Developer Eclipse 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** *(on page 189)* - 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 Eclipse 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 Eclipse plugin obtains from the operating system.

Restriction: This *Single sign-on* support is not available for SharePoint integrations.

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
```

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 open 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 open editors.

Viewing File Properties

The **Editor Properties** view displays 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.

Simple Text Editor

While Oxygen XML Developer Eclipse plugin specializes in XML-related technologies, you can also use it to create and edit various types of non-XML files. Non-XML files are opened in a simple text editor and many of the helpful features that are commonly used when editing XML files in the Oxygen XML Developer Eclipse plugin **Text** editing mode *(on page 222)* are available in this simple editor.

Types of Non-XML Files That are Supported in the Simple Text Editor

The types of non-XML files that can be created and edited in the simple text editor include:

- Java
- C++
- C
- PHP
- Perl
- Properties
- SQL
- Shell executables
- Batch
- Python
- Text

Features Available in the Simple Text Editor

When editing files in the simple text editor, the features that are available include the following:

- **Project Support** - The unique features that are designed to help you work with projects *(on page 191)* are available for all types of files.
- **Shortcut Actions** - Many of the shortcut actions that are available in **Text** mode *(on page 225)* are also available in the simple text editor.
- **Drag and Drop** - The normal drag and drop support is available in the simple text editor.
- **Content Selection Features** - The content selection shortcuts *(on page 232)* that are available in **Text** mode are also available in the simple text editor.
- **Convert Hexadecimal Characters** - You can convert a sequence of hexadecimal characters to the corresponding Unicode character *(on page 260)*.
- **Encoding/Decoding Actions** - Contextual menu actions are available to encode or decode Base 64, Base 32, and Hex schemes *(on page 261)*.
- **Code Templates** - You can define your own code templates *(on page 238)* for any type of file and use the *Content Completion Assistant (on page 1324)* to invoke them.
- **Syntax Highlighting** - Non-XML files also support syntax highlighting with dedicated coloring schemes. To customize them, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Syntax Highlight** *(on page 101)*. Select and expand the appropriate section in the top pane for the type of file you are editing and you can see the effects of your changes in the **Preview** pane.

Using Projects to Group Documents

Oxygen XML Developer Eclipse plugin includes a **Navigator** view *(on page 192)* that helps you organize your projects. Oxygen XML Developer Eclipse plugin offers a variety of helpful features for working with projects and makes it easy to share your projects with other members of your team. This section presents various unique features that will help you to create and work with projects.

Creating a New Project

Oxygen XML Developer Eclipse 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 *(on page 192)* 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 *(on page 192)*.

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 *(on page 192)* will display the existing files and subdirectories. If your files are scattered among 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 *(on page 192)* 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 192)*.

Related Information:

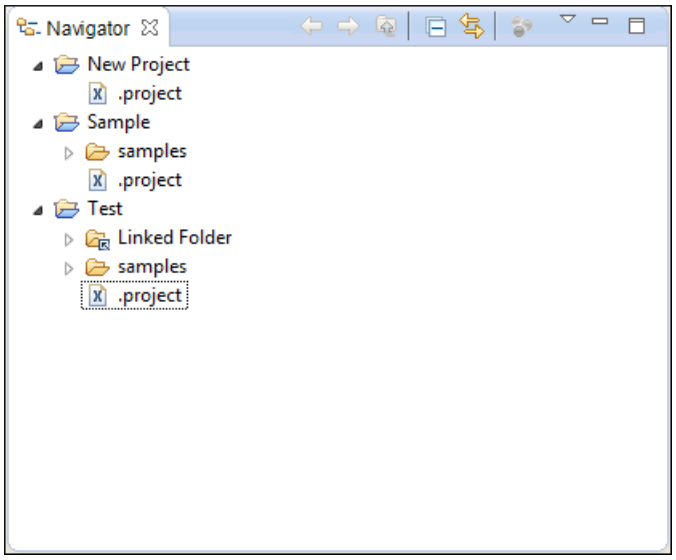
Using Projects to Group Documents *(on page 191)*

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 in the contextual menu and on the toolbar associated to this panel allows you to create XML projects and provide shortcuts to various operations for the project documents.

Tip: You can also use the **Project Explorer** view for many of the same purposes as the **Navigator** view.

Figure 41. Navigator View



By default, the view is positioned on the left side of the Oxygen XML Developer Eclipse plugin window, above the **Outline** view *(on page 240)*. If the view has been closed, it can be reopened at any time from the **Window > Show View** menu (the **Project Explorer** view is in the **Other** submenu).

Project View Toolbar

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 (on page 1327)*.

View Menu

Drop-down menu that contains various settings contributed by the Eclipse plugin.

File Explorer Area

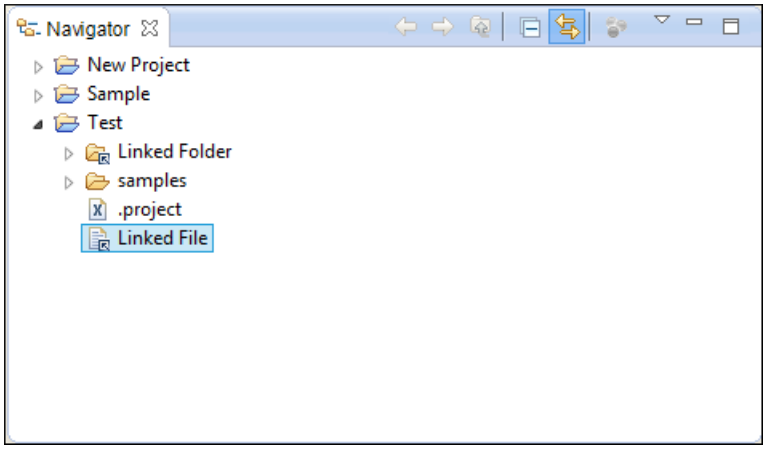
The rest of the view is basically a file explorer similar to most other commonly used file explorers. The XML project (`.xpr` file) is a logical container with a collection of resources (folders and files). The types of resources displayed include:

- **Logical folders with Linked folders/files** - This folder type is used as containers for linked resources (shortcuts). The icons for file shortcuts include a shortcut symbol ( ). The linked folders/files are added using **New > Folder > Advanced** or **New > File > Advanced**, or by dragging and dropping files/

folders from the view or the system file explorer. **Delete** can be used to remove them from the project.

- **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**, **New > Folder**, **Copy**, and **Paste**) or by dragging and dropping files/folders from the view or the system file explorer. Also, the contextual menu action **Delete** can be used to remove them from the project and local file system.

Figure 42. Navigator View with Both Types of Resources



Creating New Projects

The following actions are available from the **New** menu when right-clicking any item, or the **File > New** menu:

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.

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.

Managing Project Contents

There are various contextual menu actions, shortcuts, and ways to organize the folders and files inside the project:

Creating New Folders and Files

Right-click any item > New > File

Opens a New Document Wizard *(on page 171)* that helps you create a new file and adds it to the project structure.

Right-click any item > New > New from Templates

Opens a wizard where you can create a new document based on a template *(on page 177)*and adds it to the project structure.

Right-click any item > 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.

Adding Resources

You can add resources by using drag and drop (or **Copy** and **Paste**) actions from within the **Navigator** view or dragging them from the system file explorer. Files may have multiple instances within the folder system, but cannot appear twice within the same folder.

Removing Folders and Files

To remove logical folders or the linked resources inside them from the project, use **Delete** from the contextual menu (or press **Delete** on your keyboard) and confirm by clicking **OK** in the resulting dialog box.

To remove folders or files from both the project and the local file system, use Delete from the contextual menu (or press **Delete** on your keyboard) and confirm by clicking **OK** in the resulting dialog box.

Moving Folders and Files

You can move the resources by using drag and drop actions from within the **Navigator** view.

You can also use the usual **Copy** and **Paste** actions (or the **Move** action) from the contextual menu to move resources in the project.

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)** - Select this option to update the references to the resource you are moving, based upon the selected scope. You can select or configure the scope *(on page 318)* by using the button.

Renaming Folders and Files

There are several ways to rename a folder or file in the project (this works for both physical and linked resources):

- Select **Rename** from the contextual menu.
- Press **F2** on your keyboard.

You also can 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** - Select this option to update the references to the resource you are renaming. You can select or configure the scope *(on page 318)* by using the button.

Opening Files

There are several ways to open a file:

- Double-click the file.
- Select it and press **Enter** on your keyboard.
- Right-click the file and select **Open**.
- Drag the file from the project tree and drop it in the editor area.
- If you want to choose the application or location where to open it, you can right-click the file and select **Open with**.

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.

Other Contextual Menu Actions

Numerous other actions are available in the contextual menu, depending on the type of file or folder where it is invoked from (some actions are available for multiple selected files):

Refactoring submenu

Oxygen XML Developer Eclipse 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 (Available for certain types of XML documents)

Opens the **Rename resource** dialog box *(on page 198)* where you can change the name of a resource. It also includes an option to update the references to the renamed resource and you can choose between various scopes for the operation.

Move resource (Available for certain types of XML documents)

Opens the **Move resource** dialog box *(on page 198)* where you can choose a destination and change the name of a resource. It also includes an option to update the references to the moved resource and you can choose between various scopes for the operation.

XML Refactoring

Opens the **XML Refactoring** tool wizard *(on page 325)* that presents refactoring operations to assist you with managing the structure of your XML documents.

Show Resource Hierarchy

Opens the **Resource Hierarchy/Dependencies** view *(on page 318)* that allows you to see the resource hierarchy for an XML document.

Show Resource Dependencies

Opens the **Resource Hierarchy/Dependencies** view *(on page 318)* that allows you to see the resource dependencies for an XML document.

Refresh

Refreshes the content and the dependencies between the resources in the **Master Files** directory *(on page 200)*.

XPath in Files

Opens the **XPath/XQuery Builder** view *(on page 1119)* 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. *(on page 214)*

Format and Indent Files

Opens the **Format and Indent Files** dialog box *(on page 256)* that allows you to configure the format and indent (*pretty-print (on page 1327)*) action that will be applied on the selected documents.

Transform submenu

The currently selected files associated with the Oxygen XML Developer Eclipse plugin in the **Package Explorer** view or **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 *(on page 670)*.

Configure Transformation Scenario(s)

Opens a dialog box *(on page 765)* 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.

Validate submenu

The currently selected files associated with the Oxygen XML Developer Eclipse plugin in the **Package Explorer** view or **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. For EPUB files, this action triggers an **EPUB Validate and Check for Completeness** *(on page 1128)* operation.

Validate with Schema

Validates the selected file of files against a specified schema.

Configure Validation Scenario(s)

Allows you to configure and run a validation scenario *(on page 287)*.

Clear Validation Markers

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

Generate XML Schema Documentation

Opens the XML Schema Documentation Dialog Box *(on page 466)*.

Generate Stylesheet Documentation

Opens the XSLT Stylesheet Documentation Dialog Box *(on page 398)*.

Generate XQuery Documentation

Opens the XQuery Documentation Dialog Box *(on page 494)*.

Generate WSDL Documentation

Opens the WSDL Documentation Dialog Box *(on page 514)*.

Properties

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

Enable Master Files Support (Available from the project container)

Allows you to enable the **Master Files Support** *(on page 201)* for each project you are working on.

Detect Master Files (Available from the project container when Master Files Support is enabled)

Opens the **Detect Master Files** wizard *(on page 201)* that enables the automatic detection of *master files*.

Add to Master Files (Available when Master Files Support is enabled)

Adds the selected files to the Master Files folder *(on page 202)*.

Moving/Renaming Resources in the Navigator View

The **Refactoring** submenu in the contextual menu of the **Navigator** view *(on page 192)* provides actions for moving or renaming resources in the current project while offering the option to update the references to the resources.

Moving Resources

You can 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)** - Select this option to update the references to the resource you are moving, based upon the selected scope. You can select or configure the scope *(on page 318)* by using the button.

Renaming Resources

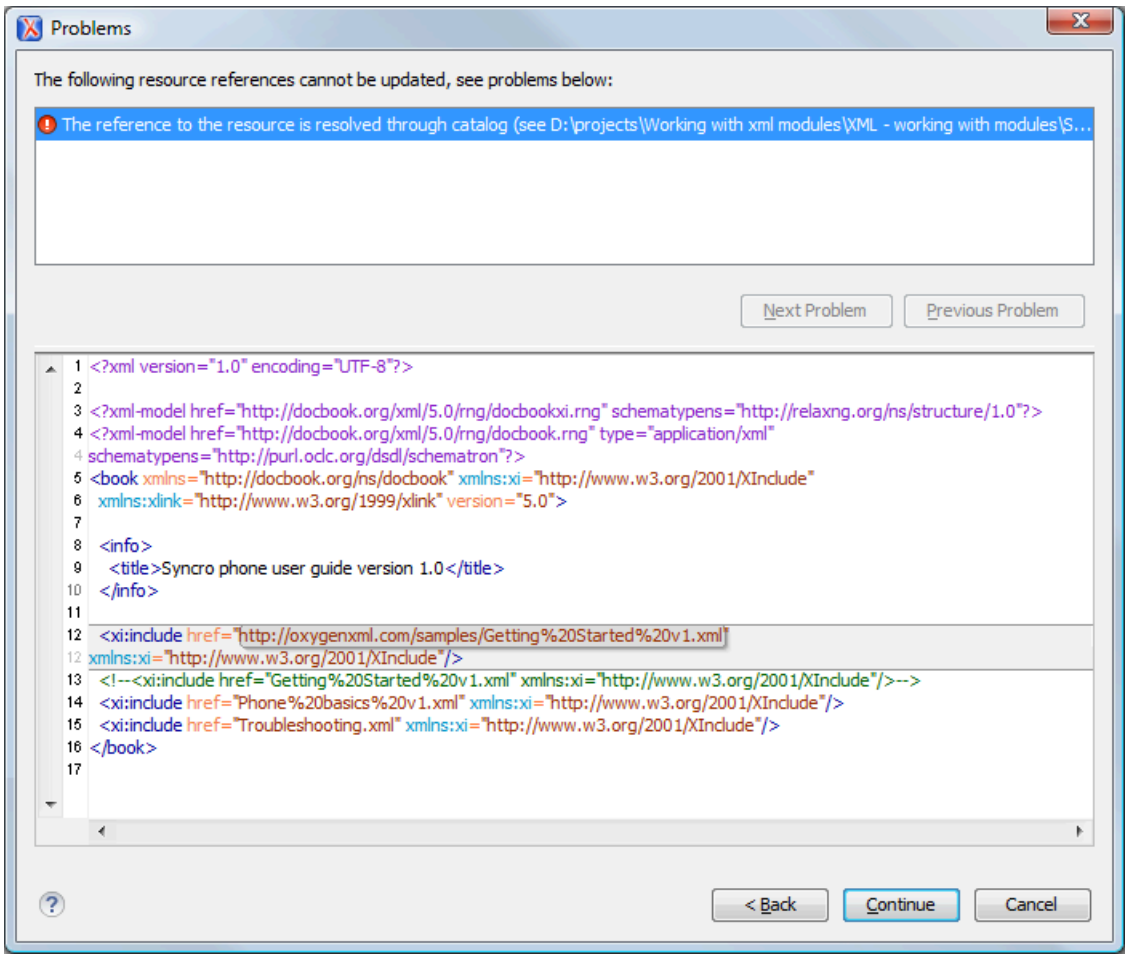
You can rename certain types of files (such as XML, XML Schema, Relax NG, WSDL, and XSLT) by using the **Refactoring > Rename resource** action from the contextual menu. This action opens the **Rename resource** dialog box that includes the following options:

- **New name** - Presents the current name of the edited resource and allows you to modify it.
- **Update references of the renamed resource** - Select this option to update the references to the resource you are renaming. You can select or configure the scope *(on page 318)* by using the button.

Problems Updating References of Moved/Renamed Resources

In some cases, the references of a moved or a renamed resource cannot be updated. For example, when a resource is resolved through an *XML Catalog (on page 1329)* or when the path to the moved or renamed resource contains entities. For these cases, Oxygen XML Developer Eclipse plugin displays a warning dialog box.

Figure 43. Problems Dialog Box



Master Files Support

Oxygen XML Developer Eclipse plugin allows you to define *Master Files (on page 1326)* at project level. These *master files* are automatically used by Oxygen XML Developer Eclipse plugin to determine the context for operations such as validation, transformation, content completion, refactoring, or searches for XML, XSD, XSL, WSDL, and RNG modules. Oxygen XML Developer Eclipse plugin maintains the hierarchy of the *master files*, helping you to determine the editing context.

For more information about the *Master Files* support for XML documents, watch our video demonstrations:

https://www.youtube.com/embed/e2oo4RWNxW8

https://www.youtube.com/embed/UZwg385RKNw

https://www.youtube.com/embed/FQNSsg57S4E

https://www.youtube.com/embed/gn_YPD5xDCo

Master Files Benefits

Using the *Master Files* support in Oxygen XML Developer Eclipse plugin includes the following benefits:

- When the *master file* is validated, Oxygen XML Developer Eclipse plugin automatically identifies the modules included in the *master file* and validates all of them.
- When the *master file* is transformed, Oxygen XML Developer Eclipse plugin automatically identifies the modules included in the *master file* and transforms them accordingly.
- The *Content Completion Assistant (on page 1324)* presents all the components that are collected from the *master files* for the modules they include.
- The **Outline** view *(on page 240)* 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 *(on page 318)*. Oxygen XML Developer Eclipse 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 Eclipse plugin stores the *master files* in a folder located in the **Navigator** view *(on page 192)*, as the first child of the project root. The *Master Files Support* is disabled by default and Oxygen XML Developer Eclipse plugin allows you to enable or disable the *Master Files Support* for each project you are working on.

To enable *Master Files* support, select **Enable Master Files Support** from the contextual menu of the project root folder in the **Navigator** view *(on page 192)*.

Related Information:

Detecting Master Files *(on page 201)*

Adding Files to the Master File Directory *(on page 202)*

Detecting Master Files

Oxygen XML Developer Eclipse 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 from Project**.
- 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 Eclipse 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*.

Note: For DITA projects, only *DITA Maps (on page 1324)* are reported as possible *master files*.

- **Cycles** - The files that are presented on the first level have circular dependencies between them. Any file 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 select their checkboxes. Oxygen XML Developer Eclipse 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*. The **Deselect All** button simply removes all of your selections.

Tip: It is recommended that you only add top-level files (files that are at the root of the include/import graph) in the `Master Files` directory.

Related Information:

Enabling the Master Files Support *(on page 201)*

Adding Files to the Master File Directory *(on page 202)*

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 *(on page 318)*, use the **Add to Master Files** action.

You can view the *master files* for the currently edited resource in the **Editor Properties** view *(on page 190)*.

Related Information:

Enabling the Master Files Support *(on page 201)*

Detecting Master Files *(on page 201)*

Search and Find/Replace Features

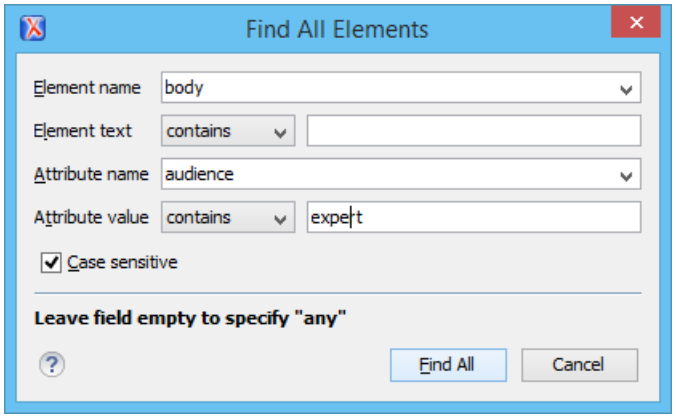
Oxygen XML Developer Eclipse plugin includes advanced search capabilities to help you locate documents and resources. The search features are powered by Apache Lucene. Apache Lucene is a free open source information retrieval software library. You can perform simple text searches or more complex searches using the Apache Lucene - Query Parser Syntax.

Note: When Oxygen XML Developer Eclipse plugin performs the indexing of resources, referenced content is not taken into account. For example, when DITA documents are indexed, the content referenced in a `@conref` or `@conkeyref` attribute is not parsed. The files that make up the index are stored on disk in the folder.

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 44. 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 selected, operations are case-sensitive.

When you select **Find All**, Oxygen XML Developer Eclipse 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 Eclipse plugin uses the Java regular expression syntax. It is **similar** to that used in Perl 5, with several exceptions. Thus, Oxygen XML Developer Eclipse 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 Eclipse plugin/Java in the text searches. 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:

- **/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:

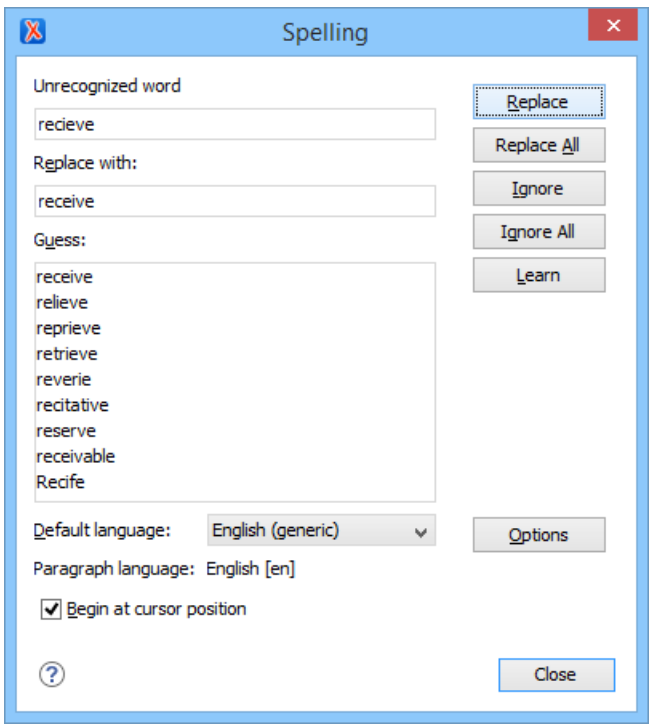
Comparison between the Java and Perl 5 regular expression syntax

Spell Checking

Oxygen XML Developer Eclipse plugin includes an automatic (as-you-type) spell checking feature *(on page 213)*, 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 45. 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 suggested words 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 Eclipse 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, starting from the cursor's position to the end of the document.

Note: This action is case-sensitive.

Ignore

Ignores the first occurrence of the unrecognized word and allows you to continue checking the document. Oxygen XML Developer Eclipse plugin skips the content of the XML elements marked to be ignored *(on page 213)*.

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 *(on page 98)* where you can configure various options regarding the feature.

Spell Check Dictionaries and Term Lists

Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse 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 Eclipse plugin spell checker by following this procedure *(on page 208)*.

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 Eclipse plugin provides a way of adding personalized *term lists (on page 210)* 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 Eclipse plugin uses for storing *learned words (on page 212)*.

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 Custom Spell Check Dictionaries *(on page 208)*

Adding Custom Spell Check Term Lists *(on page 210)*

Building and Testing Hunspell Dictionaries

Adding Custom Dictionaries and Term Lists

The Oxygen XML Developer Eclipse 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 the `.aff` file extension). The personalized term lists are custom files (with the `.tdi` file extension) that you can create to include specialized terms or specify forbidden words in the Oxygen XML Developer Eclipse plugin spell checker.

You can add dictionaries *(on page 208)* and personalized term lists *(on page 210)* 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 *(on page 100)* and the **Include dictionaries and term list from** option *(on page 100)* 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 Eclipse plugin.

Related Information:

Replacing a Spell Check Dictionary *(on page 211)*

Adding Custom Spell Check Dictionaries

There are three possible scenarios for adding Hunspell dictionaries to the Oxygen XML Developer Eclipse 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 the `.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 for the language code, followed by an underscore or hyphen, then two letters that indicate the country code, followed by another 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 for a less specific English medical dictionary, you could omit the country code like this: `en_medical.dic`). For a list of language codes, see https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.
2. Open the **Preferences** dialog box *(on page 36)* and go to **Editor > Spell Check > Dictionaries** *(on page 100)*.
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 *(on page 100)*.
b. Copy both files (`.dic` and `.aff`) to any other directory, select the **Include dictionaries and term list from** option *(on page 100)*, and select that directory. If you choose this option, make sure you read this important note *(on page 100)*.

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 the `.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 for the language code, followed by an underscore or hyphen, then two letters that indicate the country code, followed by another 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 for a less specific English medical dictionary, you could omit the country code like this: `en_medical.dic`). 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
   ```

   Tip: Words stored in dictionaries are not handled as case-sensitive. Therefore, you do not need to include both uppercase and lowercase versions of the words.

   Note: If you save the `.dic` file using UTF-8 encoding, then the corresponding `.aff` file should specify the encoding as a property inside it (if you do not specify the encoding, the default platform encoding will be used):

   ```
   SET UTF-8
   ```
3. Open the **Preferences** dialog box *(on page 36)* and go to **Editor > Spell Check > Dictionaries** *(on page 100)*.
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 *(on page 100)*.
   b. Save both files (`.dic` and `.aff`) to any other directory, select the **Include dictionaries and term list from** option *(on page 100)*, and select that directory. If you choose this option, make sure you read this important note *(on page 100)*.
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. Create your Hunspell dictionary. For more information on how to do this, see: Hunspell GitHub Project Repository.

   Step 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 for the language code, followed by an underscore or hyphen, then two letters that indicate the country code, followed by another 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 for a less specific English medical dictionary, you could omit the country code like this: `en_medical.dic`). For a list of language codes, see https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.
2. Open the **Preferences** dialog box *(on page 36)* and go to **Editor > Spell Check > Dictionaries** *(on page 100)*.
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 *(on page 100)*.
   b. Save both files (`.dic` and `.aff`) to any other directory, select the **Include dictionaries and term list from** option *(on page 100)*, and select that directory. If you choose this option, make sure you read this important note *(on page 100)*.
4. Restart the application for the spell checker to start using the new dictionary.

Related Information:

- Adding Custom Spell Check Term Lists *(on page 210)*
- Editing the Spell Checking Dictionaries

Adding Custom 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 the `.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
```

   Note: Words stored in term lists are not handled as case-sensitive. Therefore, you do not need to include both uppercase and lowercase versions of the words.
3. Open the **Preferences** dialog box *(on page 36)* and go to **Editor > Spell Check > Dictionaries** *(on page 100)*.
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 *(on page 100)*.
   b. Save the file (`.tdi`) to any other directory, select the **Include dictionaries and term list from** option *(on page 100)*, and select that directory. If you choose this option, make sure you read this important note *(on page 100)*.
5. Restart the application for the spell checker to start using the new term list.

Related Information:

Adding Custom Spell Check Dictionaries *(on page 208)*

Replacing a Spell Check Dictionary

There are several possible scenarios for replacing an existing Hunspell dictionary for the Oxygen XML Developer Eclipse 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 the `.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 *(on page 36)* and go to **Editor > Spell Check > Dictionaries** *(on page 100)*.
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 *(on page 100)*. Leave the **Include dictionaries and term list from** option deselected.

b. Replace existing files (`.dic` and `.aff`) for the particular language in a directory specified in the **Include dictionaries and term list from** option *(on page 100)*. If you choose this option, make sure you read this important note *(on page 100)*.

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.

   Step Result: You should end up with a *dictionary* file (with a `.dic` file extension) and an *affix* file (with the `.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 *(on page 36)* and go to **Editor > Spell Check > Dictionaries** *(on page 100)*.
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 *(on page 100)*. Leave the **Include dictionaries and term list from** option deselected.
   b. Replace existing files (`.dic` and `.aff`) for the particular language in a directory specified in the **Include dictionaries and term list from** option *(on page 100)*. If you choose this option, make sure you read this important note *(on page 100)*.
4. Restart the application for the spell checker to start using the new dictionary.

Related Information:

Adding Custom Dictionaries and Term Lists *(on page 207)*

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**.
- Click the **Learn** button from the **Spelling** dialog box *(on page 205)* 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 *(on page 100)*.

Related Information:

Adding Custom Spell Check Term Lists *(on page 210)*

Ignored Words (Elements)

You may want the content of certain XML elements to always be skipped during the spell check process (for example, `<programlisting>`, `<codeblock>`, `<codeph>`, `<filepath>`, or `<screen>`). This can be done in one of several ways:

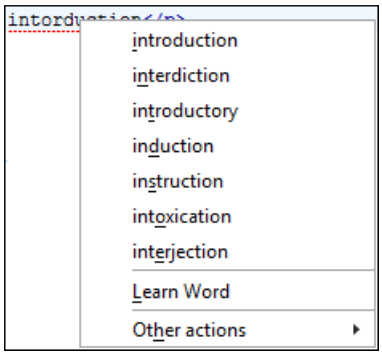
- You can skip through them manually, word by word, using the **Ignore** button in the **Spelling** dialog box *(on page 205)* that is invoked by using the **Check Spelling** action on the toolbar.
- You can automatically skip the content of certain elements by maintaining a set of known element names that should never be checked. You can manage this set of element names by using the **Ignore elements** section *(on page 99)* in the **Spell Check** preferences page.

Automatic Spell Check

Oxygen XML Developer Eclipse 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 *(on page 98)*. When the **Automatic Spell Check** option *(on page 98)* is selected, 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.

Tip: You can configure the color and how spelling errors are shown from the Eclipse **Annotations** preferences page (**Window ('Eclipse' on Mac OSX) > Preferences > General > Editors > Text Editors > Annotations**).

Figure 46. 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 *(on page 212)*.

Other actions

This submenu give you access to all the usual contextual menu actions.

Related Information:

Learned Words *(on page 212)*

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 *(on page 192)*.

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 *(on page 248)* at the bottom of the editor and you can group the reported errors as a tree with two levels.

Tip: If you want to instruct the spell checking engine to not report a particular word as being a spelling error in the future, use the **Learn Word(s)** action from the contextual menu in the **Results** view.

The following scopes are possible, depending on where the action was invoked:

- **All opened files** - The spell check is performed in all open files.
- **Current file directory** - All the files in the folder of the currently edited file.
- **Current DITA map hierarchy** - Option available when the dialog is invoked from the **DITA Maps Manager** view. Checks the spelling in all references contained in the DITA map.
- **Project** - All files from the current project.
- **Selected project resources** - 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** - Allows you to filter the files from the selected scope.
- **Recurse subdirectories** - When selected, 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 selected, 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 page *(on page 98)*.

Working with Special Characters and Encoding

While regular characters make up the English and European alphabets and the corresponding basic set of figures and symbols, there are many other *special characters* that belong to various other language representations, such as Arabic, Indian, Japanese, Chinese, or Korean. Oxygen XML Developer Eclipse plugin provides support for special characters in various ways:

Opening and Saving Documents

The Unicode standard provides support for all the character symbols in all known languages andOxygen XML Developer Eclipse plugin provides support for all Unicode characters *(on page 215)*. There are various encoding options and features to help determine how to handle documents with unsupported characters *(on page 216)*.

Fonts

Oxygen XML Developer Eclipse plugin provides the ability to choose the fonts to be used in the various editing modes *(on page 108)*. In some cases, changing the font may be a solution when special characters are not rendered as expected.

For special characters that are not included in any of the default fonts, Oxygen XML Developer Eclipse plugin tries to find that symbol in a *fallback* font *(on page 216)*.

Tip: For documents written in languages that use special characters (such as 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. To change the font in Oxygen XML Developer Eclipse plugin, open the **Preferences** dialog box *(on page 36)*, go to **Fonts**. You can select a font for each editing mode in this preferences page.

Navigation and Layout

Oxygen XML Developer Eclipse plugin supports bidirectional text, such as Arabic, Hebrew, and certain Asian languages, or other special characters that are combined into a single glyph.

Editing

Oxygen XML Developer Eclipse plugin includes a contextual menu action that converts a sequence of hexadecimal characters to the corresponding Unicode character *(on page 260)*.

If you do not have a special way of inserting special characters using your keyboard, you can insert special characters using the **Character Map** feature *(on page 217)*.

Unicode Support

Unicode is a standard for providing consistent encoding, representation, and handling of text. There is a unique Unicode number for every character, independent of the platform and language. Unicode is

internationally recognized and is required by modern standards (such as XML, Java, JavaScript, LDAP, CORBA 3.0, WML, etc.).

Oxygen XML Developer Eclipse 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 Eclipse plugin uses 16-bit characters covering the Unicode Character set.

Note: Oxygen XML Developer Eclipse 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**.

Related Information:

Unicode Fallback Font Support *(on page 216)*

Inserting Special Characters with the Character Map *(on page 217)*

Opening and Saving Documents with Unsupported Characters

When loading documents, Oxygen XML Developer Eclipse 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.

Saving Documents with Unsupported Characters

When saving a document with UTF-16 encoding, 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 one 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.

Unicode Fallback Font Support

Oxygen XML Developer Eclipse plugin provides fonts for most common Unicode ranges. However, if you use special symbols or characters *(on page 217)* 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 *(on page 215)* 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 Eclipse 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 Eclipse 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 Eclipse plugin. To allow Oxygen XML Developer Eclipse plugin to recognize and render the symbol correctly, you can add **Segoe UI Symbol** as a *fallback* font.

Adding a Fallback Font in Windows (7 or Later)

To add a fallback font to the Oxygen XML Developer Eclipse 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 Eclipse plugin for the changes to take full effect.

Result: Whenever Oxygen XML Developer Eclipse 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.

Adding a Fallback Font in 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 Eclipse plugin, open the **Preferences** dialog box *(on page 36)*, go to **Fonts**.
4. Click the **Change** button for the particular editing mode and select your custom font from the drop-down list in the subsequent dialog box.
5. Restart Oxygen XML Developer Eclipse plugin for the font changes to take full effect.

Related Information:

Unicode Support *(on page 215)*

Inserting Special Characters with the Character Map *(on page 217)*

Inserting Special Characters with the Character Map

Oxygen XML Developer Eclipse plugin includes a **Character Map** for inserting special characters. It can also be used to find the decimal, hexadecimal, or *character entity* equivalent for a particular character or symbol.

Inserting Special Characters

To insert a special character at the current location within a document, follow these steps:

1. Open the **Character Map** dialog box *(on page 218)* by **Edit >** Ω **Insert from Character Map**.
2. Find the symbol you want to insert and double-click it (or select it and click **Insert**).

Tip: The most recently used characters and some of the most common characters are listed when you click the Ω **Symbols** drop-down button so you can easily insert any of those characters by simply selecting it from the drop-down.

Finding the Decimal, Hexadecimal, or Character Entity Equivalent

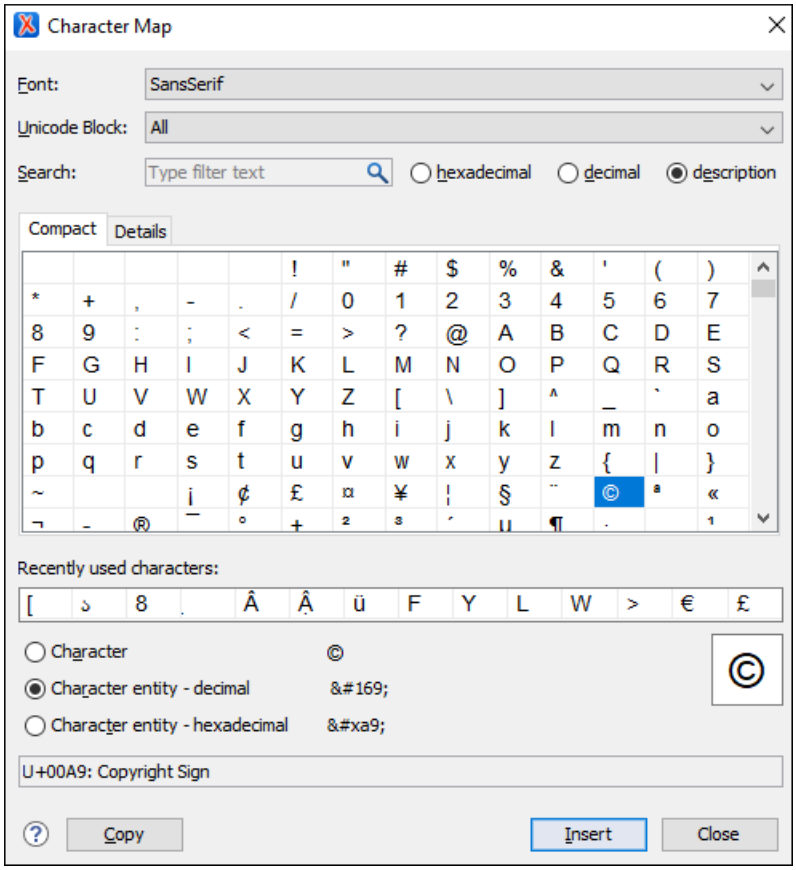
You can see the hexadecimal value for any character that is already inserted in your document by placing the cursor right after the character and you can see its value in the status bar at the bottom of the application.

For other characters, or to find the decimal equivalent, or even the *character entity* equivalent, following these steps:

1. Open the **Character Map** dialog box *(on page 218)* by **Edit >** Ω **Insert from Character Map**.
2. Find the symbol and select it. You can use the filters and the **Search** field at the top of the dialog box to narrow the search.
3. Click the **Details** tab on top of the preview window to see the decimal, hexadecimal, and description of the character. The *character entity* equivalent (both its decimal and hexadecimal values) are displayed at the bottom of the dialog box.

Character Map Dialog Box

Figure 47. 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 that will have characters displayed.

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 *(on page 219)*. If you enter a character description in the **Search** field, the **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.

Click the **Insert** button to insert the selected character in the current editor at the cursor position. You will see the character in the editor if the editor font *(on page 108)* 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 *(on page 168)*.

Related Information:

Working with Special Characters and Encoding *(on page 215)*

Handling Read-Only Files

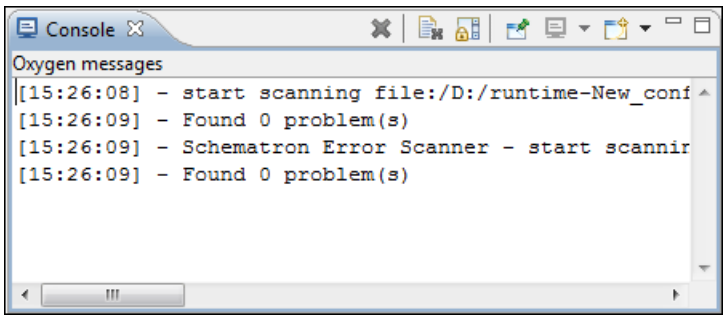
The default workbench behavior applies when editing read-only files in the **Text** mode. For all other modes no modification is allowed provided that the file remains read-only.

You can check out the read-only state of the file by looking in the **Properties** view *(on page 190)*. 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.

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 *(on page 112)* must be selected in the **View** preferences page *(on page 112)*).

Figure 48. 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** *(on page 112)* 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 All Elements** *(on page 203)*
- **XPath in Files** *(on page 197)*
- **Search References** *(on page 267)*
- **Search Declarations** *(on page 267)*

By default, Oxygen XML Developer Eclipse 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 *(on page 76)*. 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 editor pane.
- Clicking the **Next** and **Previous** buttons ( ) from the bottom of the range ruler, located at the right side of the editor pane.

  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 *(on page 248)* 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, located at the right side of the editor pane.
- 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 on the right side of the **Results** panel *(on page 248)* at the bottom of the editor.

8.

Editing Documents

Oxygen XML Developer Eclipse plugin includes built-in *frameworks* for the most popular XML document types (DITA, DocBook, TEI, XHTML, JATS) *(on page 646)* with a full set of features (full editing support, document templates, enhanced CSS rendering, specific actions, validation, content completion, transformation scenarios, and more). In addition, Oxygen XML Developer Eclipse plugin provides support for editing numerous other types of documents (all XML document types and even some non-XML formats).

Each type of document has unique features and options and this chapter includes a large amount of information about editing numerous types of documents and various editing features that are provided in Oxygen XML Developer Eclipse plugin, including general information about editing XML documents in **Text** mode *(on page 222)*, and **Grid** mode *(on page 268)*.

Related Information:

Built-in XML Frameworks (Document Types) *(on page 646)*

Editing XML Documents

The structure of an XML document and the required restrictions on its elements and their attributes are defined with an XML schema. For more information about schema association, see Associating a Schema to XML Documents *(on page 303)*.

Oxygen XML Developer Eclipse plugin includes fully supported built-in *frameworks (on page 1325)* for the most popular XML document types (DITA, DocBook, TEI, XHTML, JATS) *(on page 646)* with a full set of features. These built-in *frameworks* are defined according to a set of rules and a variety of settings that improve editing capabilities for its particular file type.

This section 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 168)*

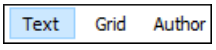
Grid Editing Mode *(on page 168)*

Built-in XML Frameworks (Document Types) *(on page 646)*

Editing XML Documents in Text Mode

This section includes topics that describe how to work with XML documents in **Text** mode, including various features, actions that are available, and much more.

The Oxygen XML Developer Eclipse 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 Eclipse plugin also includes numerous specialized editing actions, a powerful *Content Completion Assistant (on page 234)*, a helpful **Outline** view *(on page 240)*, and many other unique features.

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

Navigating the Document Content in Text Mode

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

Navigation Keyboard Shortcuts

Ctrl + CloseBracket (Command + CloseBracket on OS X)

Navigate to the next XML node.

Ctrl + OpenBracket (Command + OpenBracket on OS X)

Navigate to the previous XML node.

Ctrl + RightArrow (Command + RightArrow on OS X)

Navigate one word forward.

Ctrl + LeftArrow (Command + LeftArrow on OS X)

Navigate one word backward.

Ctrl + Home (Command + Home on OS X)

Position the cursor at the beginning of the document.

Ctrl + End (Command + End on OS X)

Position the cursor at the end of the document.

Navigating to a Modification

Oxygen XML Developer Eclipse plugin includes some actions to help you quickly navigate to a particular modification. They can be invoked using keyboards shortcuts or from the **Navigation** menu:

Last Edit Location (Ctrl+Q)

Navigates to the last modification in any open tab.

Back (Alt+LeftArrow (Command+OpenBracket on OS X))

Navigates to the last selected editor tab or to the last selected element/content in the current tab. You can also go back after clicking on links.

Forward (Alt+RightArrow (Command+CloseBracket on OS X))

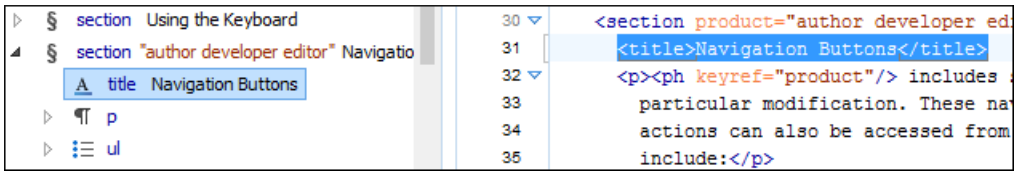
Available after you use the **Back** button at least once, and it navigates in the opposite direction as the **Back** button.

Navigating with the Outline View

Oxygen XML Developer Eclipse plugin includes an **Outline** view *(on page 240)* 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.

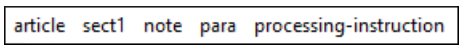
Figure 49. Outline View Navigation in Text Mode



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 50. 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 pressing **(Ctrl+L (Command+L on OS X))** or selecting **Go To Line** from the **Navigation** menu.

Navigating with Bookmarks

By using *bookmarks*, you can mark positions in an edited document so that you can return to it later. This is especially helpful for navigating through large documents or while editing multiple documents.

To insert a *bookmark* in **Text** mode, right-click the desired location in the vertical stripe on the left side of the editor and select **Add Bookmark** (you can remove it by selecting **Remove Bookmark** from the same contextual menu).

To navigate to any of the *bookmarks*, click their corresponding markers in the vertical stripe on the right side of the editor.

Tip: You can configure the color and how the *bookmarks* are shown from the Eclipse **Annotations** preferences page (**Window ('Eclipse' on Mac OSX) > Preferences > General > Editors > Text Editors > Annotations**).

Smart Editing in Text Mode

Oxygen XML Developer Eclipse 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 Eclipse plugin will mirror-edit the name of the matching end tag. This feature can be controlled from the **Content Completion** option page *(on page 77)*.
- **Auto-breaking the edited line** - The **Hard line wrap** option *(on page 91)* automatically breaks the edited line when its length exceeds the maximum line length defined for the format and indent operation *(on page 91)*.
- **Indent on Enter** - The **Indent on Enter** option *(on page 90)* indents the new line inserted when you press **Enter**.
- **Smart Enter** - The **Smart Enter** option *(on page 90)* 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 are 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 Eclipse 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 currently selected content from the document and places it in the clipboard.

Copy (Ctrl + C (Command + C on OS X))

Places a copy of the currently 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 Eclipse 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) or F3

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.
- Open the target for DITA references (such as a `@conref`, `@conkeyref`, `@keyref`, and more). [Only **F3** works for these types of references]
- 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.

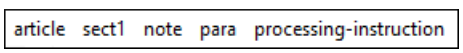
Editing XML Markup in Text Mode

Oxygen XML Developer Eclipse 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 51. 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 Eclipse 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, select 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 *(on page 78)* is selected in the **Content Completion** preferences page, the cursor is placed between the start and end tag.
- If the **Position cursor between tags** option *(on page 78)* is not selected 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 Refactoring Actions

Contains built-in 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.

Convert attribute to element

Allows you to change an attribute into an element.

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 Refactoring Actions

Contains built-in 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.

Elements Refactoring Actions

Contains built-in 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 Refactoring Actions

Contains built-in 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.

Related Information:

Refactoring XML Documents *(on page 325)*

Contextual Menu Actions in Text Mode *(on page 258)*

Folding XML Elements in Text Mode

When working with a large document, the *folding (on page 1325)* support in Oxygen XML Developer Eclipse 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 52. 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.

For more information about the *folding* support in Oxygen XML Developer Eclipse plugin, watch our video demonstration:

https://www.youtube.com/embed/eR9HfN_peAE

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 Eclipse 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 *(on page 225)* 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 *(on page 225)* 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

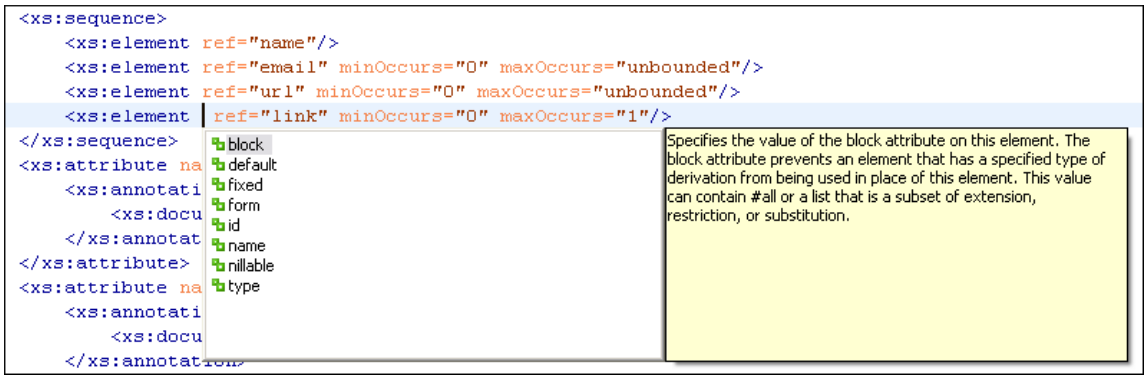
Selects the entire parent element at the current cursor position.

Content Completion Assistant in Text Mode

Oxygen XML Developer Eclipse plugin includes an intelligent *Content Completion Assistant (on page 1324)* that offers proposals for inserting structured language elements, attributes, and attribute values that are valid in the current editing context.

The *Content Completion Assistant* is enabled by default. To disable it, open the **Preferences** dialog box *(on page 36)*, go to **Editor > Content Completion**, and deselect the **Enable content completion** option *(on page 78)*.

Figure 53. Content Completion Assistant



Content Completion and the Associated Schema

The *Content Completion Assistant* feature is schema-driven and the list of proposals in the *Content Completion Assistant (on page 1324)* depend on the associated schemas (DTD, XML Schema, Relax NG, or NVDL schema). For information about the various ways to associate a schema and the order of their precedence, see the Associating a Schema to XML Documents *(on page 303)* section.

Using the Content Completion Assistant in Text Mode

The feature is activated in **Text** mode in the following situations:

- After you enter 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 *(on page 79)* from the **Content Completion** preferences page.
- After typing a partial element or attribute name, you can manually 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.

You can navigate through the list of proposals by using the **Up** and **Down** keys on your keyboard. In some cases, the *Content Completion Assistant* displays a documentation window with information about the particular proposal and some of them have links to additional information (for example, DITA elements might have a link to the DITA Style Guide). You can also change the size of the documentation window by dragging its top, right, and bottom borders.

To insert the selected proposal 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 *(on page 78)* (in the **Content Completion** preferences page) is selected. 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** *(on page 78)* and **Add first Choice particle** *(on page 78)* 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 allow 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 allow 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:

Element Structure Specified in DTD or Schema

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.

Current Cursor Position

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.

Unique ID Attribute Values

A schema may declare certain attributes as *ID* or *IDREF/IDREFS*. When the document is validated, Oxygen XML Developer Eclipse 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 for *xml:id* Attributes

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.

Links/References in DITA

When entering values for the various types of links and references in DITA (for example, values for `@href` or `@conref` elements), the *Content Completion Assistant* will propose potential targets when you use the forward slash key (*/*).

ID Values for DITA Key References

In DITA, when inserting key references (`@keyref`) or content key references (`@conkeyref`), the ID values that are defined in the key reference are presented as possible targets. The *Content Completion Assistant* will only propose targets that are valid in the current context.

Element and Attribute Values

For documents that use an XML Schema or Relax NG schema, the *Content Completion Assistant* offers proposals for attribute and element values as long as the allowed values are defined in

the schema. Also, if a default value or fixed value is defined in the schema, then that value is offered in the *Content Completion Assistant*.

Related Information:

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 (on page 1324)*.
- 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** *(on page 79)* 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** *(on page 80)* option in the **Annotations** preferences page. When this option is deselected, 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 Eclipse 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 like this:

```
<define name="person" >
  <element name="person">
    <a:documentation xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0">
            Information about a person. </a:documentation>
      <ref name="name"/>
        <zeroOrMore>
      <ref name="email"/>
        </zeroOrMore>
  </element>
</define>
```

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, select the **Use all Relax NG annotations as documentation** *(on page 80)* option in the **Annotations** preferences page.

Collecting Annotations from Relax NG Compact Syntax Schemas

For Relax NG Compact Syntax schema, annotations are made using comments like this:

```
##  Information about a person.
element person { name, email*}
```

Collecting Annotation from DTDs

For DTD, Oxygen XML Developer Eclipse plugin defines a custom mechanism for annotations using comments enabled by the **Prefer DTD comments that start with "doc:" as annotations** *(on page 79)* option in the **Annotations** preferences page. The following is an example of a DTD annotation:

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

Content Completion Helper Views (Text Mode)

Information about the current element being edited is also available in various *dockable (on page 1324)* views, such as the **Model** view *(on page 245)*, **Attributes** view *(on page 243)*, **Elements** view *(on page 247)*, and **Entities** view *(on page 247)*. By default, they are located on the right-hand side of the main editor window. These views, along with the powerful **Outline** view *(on page 240)*, provide spatial and insight information about the edited document and the current element. If any particular view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

Code Templates

Code templates are code fragments that can be inserted quickly at the current editing position. Oxygen XML Developer Eclipse plugin includes a set of built-in code templates for CSS, Schematron, XSL, XQuery, JSON,

HTML, and XML Schema document types. You can also define your own code templates for any type of file and share them with others.

Code templates are displayed with a symbol in the content completion list (**Enter** in **Author** mode or **Ctrl + Space (Command + Space on OS X)** in **Text** mode). Also, in **Text** mode you can press **Ctrl + Shift + Space** to see a complete list of the available code templates. To enter the code template at the cursor position, select it from the content completion 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.

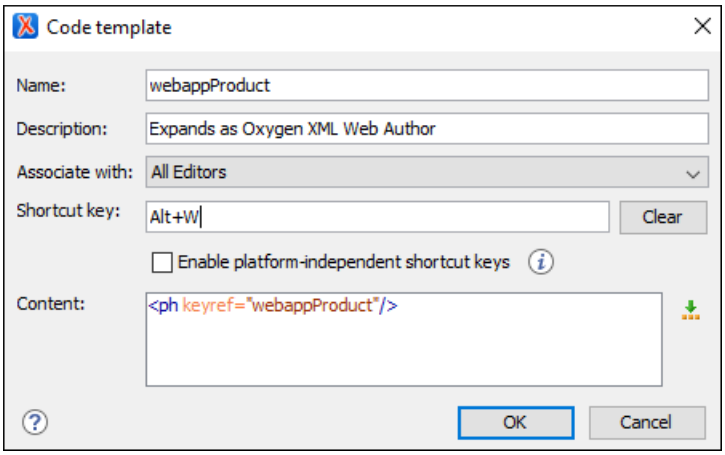
How to Create Code Templates

To create a code template, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Editor > Templates > Code Templates**.
2. Click **New** to open a code template configuration dialog box.

   Tip: You can use one of the existing code templates as a starting point by selecting that template and clicking **Duplicate**.

   Figure 54. Code Template Configuration Dialog Box

3. Configure your template using the fields in the code template configuration dialog box:
   - **Name** - The name of the code template.
   - **Description** - [Optional] 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 (on page 1324)*. 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** - [Optional] If you want to assign a shortcut key that can be used to insert the code template, place the cursor in the **Shortcut key** field and press the desired key combination on your keyboard. Use the **Clear** button if you make a mistake. If the **Enable platform-independent shortcut keys** checkbox is selected, 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.

    An editor variable *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button.
4. Click **OK** to save your new code template.

Result: Your code template can now be selected using the *Content Completion Assistant (on page 1324)* (**Enter** in **Author** mode or **Ctrl + Space (Command + Space on OS X)** in **Text** mode). The code templates are displayed with a symbol.

How to Share Code Templates

There are two ways to easily share all of your code templates with other members of your team:

Method 1: Export/Import

1. Open the **Preferences** dialog box *(on page 36)* and go to **Editor > Templates > Code Templates**.
2. Click the **Export** button to export all of your code templates into an XML file.
3. Save the XML file.
4. Share the XML file with other members of your team.
5. Instruct them to open the **Preferences** dialog box *(on page 36)*, go to **Editor > Templates > Code Templates**, click the **Import** button, and select the file you sent them.

   Result: The code templates will be now available in their content completion list.

Method 2: Share Project

1. Open the **Preferences** dialog box *(on page 36)* and go to **Editor > Templates > Code Templates**.
2. Select **Project Options** at the bottom of the dialog box. This stores the preferences in the project file (`.xpr`).
3. Share the project file with the other members of your team. For example, you can commit it to your version control system and have them update their working copy.

   Result: When they open the updated project file in their , the code templates will be available in their content completion list.

Text Mode Views

There is a variety of *dockable (on page 1324)* helper views that are displayed by default in **Text** mode. There are also a large selection of additional views available in the **Window > Show View** menu. This section presents some of the most helpful views for editing in **Text** mode.

Outline View for XML Documents

The **Outline** view 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 by selecting it 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 (such as `*` or `?`) and separate multiple patterns with commas.

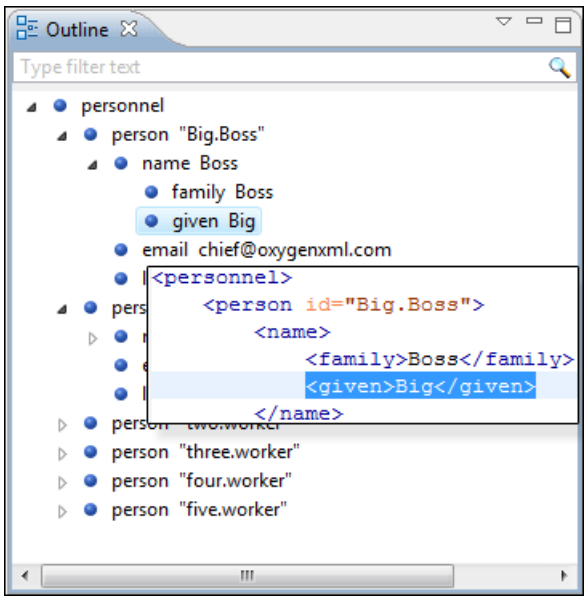
It also includes a **View menu** in the top-right corner that presents a variety of options to help you filter the view even further.

Drag 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.

Figure 55. Outline View



Outline View Filters

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

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

Filter returns exact matches

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

Selection update on cursor move (Available in Text mode)

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 *(on page 144)*.

Configure displayed attributes

Displays the **XML Structured Outline** preferences page *(on page 144)*.

Outline View Contextual Menu Actions

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

Edit Attributes

Displays an in-place attributes editor that allows you to edit the attributes of a selected node.

Append Child

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

Insert Before

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

Insert After

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

Cut, Copy, Paste, Delete common editing actions

Executes the typical editing actions on the currently selected elements. The **Cut** and **Copy** operations preserve the styles of the copied content.

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.

Expands the structure tree of the currently selected element.

Collapse All

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

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

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.

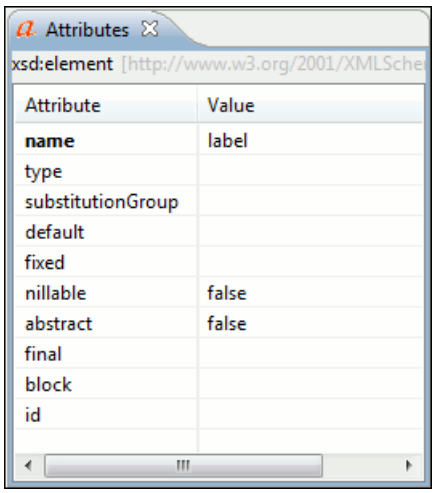
The attributes are rendered differently depending on their state:

- 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 56. 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

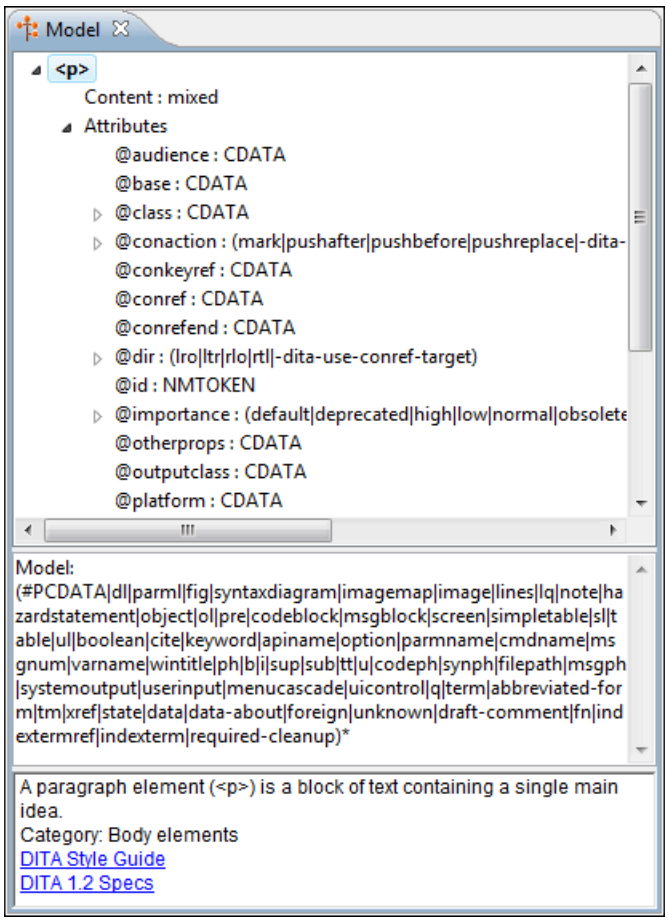
Depending on the content of the clipboard, the following cases are possible:

- 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 by selecting it from the **Window > Show View** menu.

Figure 57. Model View

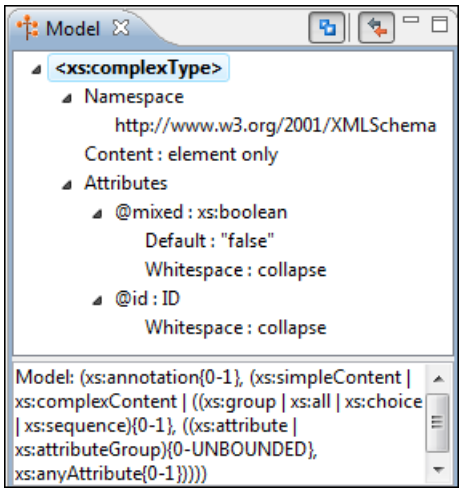


The **Model** view is comprised of two sections, an element structure panel and an annotations panel.

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.

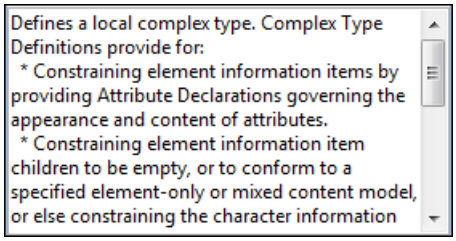
Figure 58. Element Structure Panel



Annotation Panel

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

Figure 59. Annotation panel

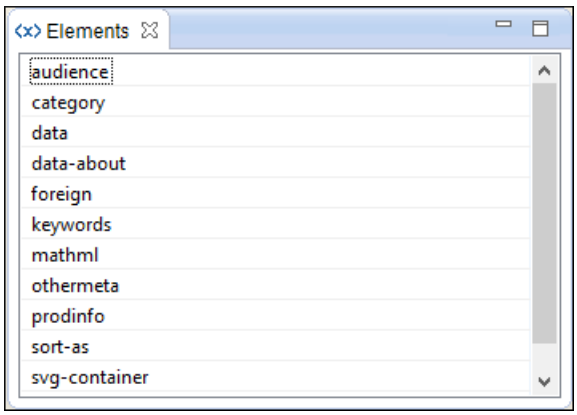


Elements View in Text Mode

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 by selecting it 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. Pressing **F2** with an element selected will display information about that particular element.

Figure 60. 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:

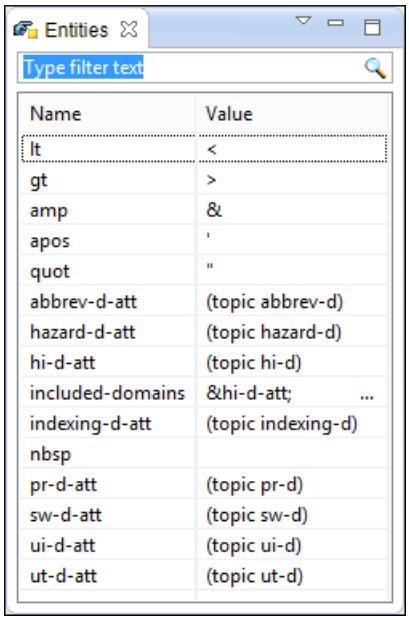
- **Predefined** - Entities that are part of the predefined XML markup (`<`, `>`, `&`, `'`, `"`).
- **Internal** - Defined in the DOCTYPE declaration header of the current XML.
- **External** - Defined in an external DTD module included in the DTD referenced in the XML DOCTYPE declaration.

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 by selecting it 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 61. 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

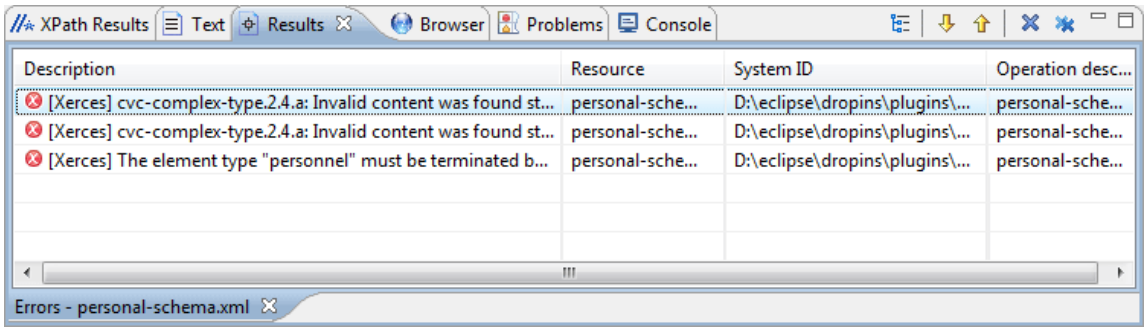
The **Results** view displays the messages generated as a result of user actions such as validations, transformations, search operations, and others. Each message is a link to the location related to the event that triggered the message. Double-clicking a message opens the file containing the location and positions the cursor at the location offset. The **Results** view is automatically opened when certain actions generate result messages. By default, the view normally opens at the bottom of the editor, but it is *dockable (on page 1324)*, so it can be moved to another UI location alongside other side views.

The actions that contribute messages to this view include:

- **Validation** actions *(on page 279)*

- **Transformation** actions *(on page 669)*
- **Check Spelling in Files** action *(on page 214)*
-
-
- **Search References** action *(on page 386)*
- SQL results *(on page 1185)*

Figure 62. 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:

Learn Word(s) (Available when spelling errors are reported in the Results view)

Adds the word(s) to a list of learned words to instruct the spell checker engine to not report the word(s) as spelling errors in the future.

Remove

Removes selected messages from the view.

Remove all

Removes all messages from the view.

Copy

Copies information associated with the selected messages. For example:

- 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.

Copy Description

Copies the description values for all selected items.

Show message

Opens a dialog box that displays the details of 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 *(on page 250)*.

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 *(on page 250)*.

Save Results as HTML

Saves the complete list of messages in a file in HTML format. For each message, the included details are the same as the ones for the **Copy** action *(on page 250)*.

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 Highlighting in XML Documents

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for XML files, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
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 **XML** tab in the **Preview** pane to see the effects of your changes.

Tip: Oxygen XML Developer Eclipse plugin also allows you to specify syntax highlighting colors for specific XML elements and attributes with specific namespace prefixes. This can be done in the **Editor > Syntax Highlight > Elements/Attributes by Prefix** preferences page *(on page 101)*.

Related Information:

Customize Syntax Highlight colors *(on page 101)*

Syntax Highlight Depending on Namespace Prefix

The syntax highlight scheme of an XML file type *(on page 101)* 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 *(on page 101)* allows easier identification of the tags.

Figure 63. Example of Coloring XML Tags by Prefix

```
<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>
```

Related Information:

Changing the colors displayed in the Text Mode Editor *(on page 101)*

Formatting and Indenting XML Documents

Oxygen XML Developer Eclipse plugin creates XML documents using several edit modes *(on page 168)*. In **Text** mode *(on page 168)*, 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 Eclipse plugin must decide how to

format and indent the XML. Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin Determines When Whitespace is Significant

When Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse 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.

Notes:

- 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 Eclipse plugin evaluates 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 Eclipse 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 Eclipse 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 Eclipse 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 *(on page 36)*, go to **Editor > Format > XML**, and select/deselect the **Schema-aware format and indent** option *(on page 95)*.

Preserve space elements list

If an element is listed in the **Preserve space** tab of the **Element Spacing** list *(on page 94)* in the XML formatting preferences *(on page 92)*, 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 *(on page 94)* in the XML formatting preferences *(on page 92)*, 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 *(on page 94)* in the XML formatting preferences *(on page 92)*, 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 Eclipse plugin formats and indents XML

You can control how Oxygen XML Developer Eclipse 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 *(on page 89)*
- **XML Formatting** preferences page *(on page 92)*
- **Whitespaces** preferences page *(on page 95)*

When Oxygen XML Developer Eclipse plugin formats and indents XML

Oxygen XML Developer Eclipse 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 *(on page 86)* is selected in the **Grid** preferences page.

Setting an Indent Size to Zero

Oxygen XML Developer Eclipse plugin will automatically format and indent *(on page 251)* 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 Eclipse 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 Eclipse plugin to maintain a zero indent when editing or formatting those documents:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Editor > Format** *(on page 89)*.
2. Select **Detect indent on open**.
3. Select **Use zero-indent if detected**.

Oxygen XML Developer Eclipse 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 Eclipse 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 *(on page 36)* and go to **Editor > Format** *(on page 89)*.
2. Deselect **Detect indent on open**.
3. Set **Indent size** to `0`.

All documents will be formatted and indented with an indent of zero.

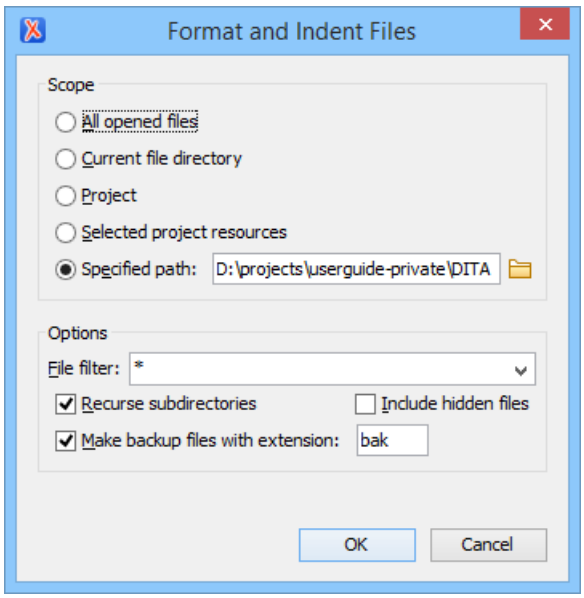
Warning: Setting the indent size to zero can change the meaning of some file types, such as Python source files.

Format and Indent (Pretty-Print) Multiple Files

Oxygen XML Developer Eclipse plugin provides support for formatting and indenting (*pretty-print (on page 1327)*) 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 *(on page 192)*. This opens the **Format and Indent Files** dialog box that allows you to configure options for the action.

Figure 64. Format and Indent Files Dialog Box



The **Scope** section allows you choose from the following scopes:

- **All opened files** - The *pretty-print (on page 1327)* is performed in all opened files.
- **Directory of the current file** - All the files in the folder of the currently edited file.
- **Project files** - All files from the current project.
- **Selected project files** - The selected files from the current project.
- **Specified path** - the *pretty-print (on page 1327)* is performed in 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 selected, the *pretty-print (on page 1327)* 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 selected, the *pretty-print (on page 1327)* is also performed in the hidden files.
- **Make backup files with extension** - When selected, Oxygen XML Developer Eclipse plugin makes backup files of the modified files. The default extension is `.bak`, but you can change the extension as you prefer.

Quick Assist Support for IDs and IDREFS

The *Quick Assist* support *(on page 1328)* 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** (**Command+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 *(on page 318)*. 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** *(on page 318)* 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** *(on page 318)* dialog box, this scope will be used instead.

Change scope

Opens the **Select the scope for the Search and Refactor operations** *(on page 318)* dialog box.

Rename in File

Renames the ID you are editing and all its occurrences from the current file.

Search Occurrences

Searches for the declaration an references of the ID located at the cursor position in the current document.

Related Information:

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 *(on page 36)* and go to **Editor > Mark Occurrences** *(on page 96)*.

Related Information:

Contextual Menu Actions in Text Mode

When editing XML documents in **Text** mode, Oxygen XML Developer Eclipse 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 (or property for JSON documents) to the clipboard.

Toggle Line Wrap (Ctrl + Shift + Y (Command + Shift + Y on OS X))

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.

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 (Command + Shift + G on OS X))

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 *(on page 89)*.

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 (on page 1327) 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 *(on page 215)*. 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 and the characters they will be converted to:

- `0x0045` will be converted to `E`
- `0X0125` to ĥ
- `265` to ĩ
- `2190` to ←

Note: For more information about finding the hexadecimal value of a character, see Finding the Decimal, Hexadecimal, or Character Entity Equivalent *(on page 218)*.

Base64 Encode/Decode submenu

This submenu include the following actions for encoding or decoding **base 64** 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 (on page 1323)* 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.

Go to Definition (Ctrl + Shift + Enter)

Navigates 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. For JSON documents, it navigates to the definition of the current JSON property in the associated JSON Schema.

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 Eclipse 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, select 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 *(on page 78)* is selected in the **Content Completion** preferences page, the cursor is placed between the start and end tag.
- If the **Position cursor between tags** option *(on page 78)* is not selected 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 Refactoring Actions

Contains built-in 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.

Convert attribute to element

Allows you to change an attribute into an element.

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 Refactoring Actions

Contains built-in 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.

Elements Refactoring Actions

Contains built-in 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 Refactoring Actions

Contains built-in 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.

Manage IDs submenu

This submenu is available for XML documents that have an associated DTD, XML Schema, or Relax NG schema (not available for DITA). 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 *(on page 318)*. 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** *(on page 318)* 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** *(on page 318)*.

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** *(on page 318)* 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** *(on page 318)*.

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 (on page 1328)* 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.

Show Resource Hierarchy

Opens the **Resource Hierarchy/Dependencies** view *(on page 318)* that allows you to see the resource hierarchy for an XML document.

Show Resource Dependencies

Opens the **Resource Hierarchy/Dependencies** view *(on page 318)* that allows you to see the resource dependencies for an XML document.

Editing XML Documents in Grid Mode

This section includes topics that describe how to work with XML documents in **Grid** mode, including various features, actions that are available, and much more.

The **Grid** mode in Oxygen XML Developer Eclipse 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.

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

You can easily expand or collapse elements within the table and the document structure can be changed with simple contextual menu actions, drag/drop, or copy/paste operations. The text content can be modified simply by editing the value of cells that contain the text and a useful *Content Completion Assistant (on page 1324)* is also available to help you edit or insert XML elements.

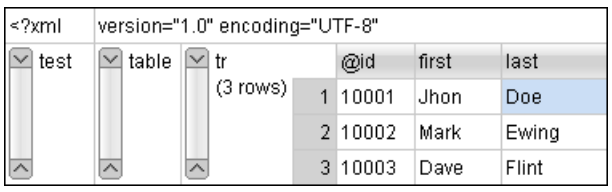
For more information about some of the features available in the **Grid** editor, watch our video demonstration:

https://www.youtube.com/embed/PoYm2VqisWk

Layouts: Grid and Tree

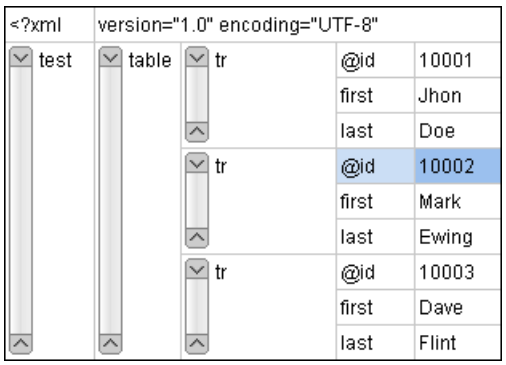
The **Grid** editor offers two layout modes. The default one is the grid layout. This smart layout detects the recurring elements in the XML document and creates tables having the children (including the attributes) of these elements as columns. This way, it is possible to have tables nested in other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

Figure 65. Grid Layout



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

Figure 66. Tree Layout



To switch between the two modes, select **Grid mode/Tree mode** from the contextual menu.

Grid Mode Navigation

When you first open a document in **Grid** mode, the content is collapsed. Only the root element and its attributes are displayed. An arrow sign ( ) displayed at the left of the node name indicates that this node has child nodes. To display the children, click this arrow sign. To collapse a node, click the reverse arrow sign ( ). The expand/collapse actions can also be invoked with the **NumPad+** and **NumPad-** keys, or from the **Expand/Collapse** submenu of the contextual menu.

Expand/Collapse Submenu

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

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.

Keyboard Shortcuts

A variety of other keyboard shortcuts are also available in **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.

Related Information:

Editing Actions in Grid Mode *(on page 270)*

Editing Actions in Grid Mode

Since **Grid** mode presents XML content in a structured grid of nested tables, editing content in this mode can be done with a combination of the *Content Completion Assistant (on page 275)* and actions that allow you to work with the structure or content of the nested tables much like you would with any table. Oxygen XML Developer Eclipse plugin provides ways to edit content in the cells of the nested tables or to edit the structure of the tables.

Tip: There are two different types of layouts available in **Grid** mode. Most people prefer to leave it on the default **Grid mode** layout, but there is also a **Tree mode** layout that presents the structure of the document in more of a vertical tree-like manner. You can switch between the two layouts to see which one works best for you particular situation from the **Document > Grid Layout** menu.

Expanding/Collapsing Nodes

An arrow sign ( ) displayed at the left of a node indicates that it has child nodes. To display the children, click this arrow sign. To collapse a node, click the reverse arrow sign ( ). The expand/collapse actions can also

be invoked with the **NumPad+** and **NumPad-** keys, or from the **Expand/Collapse** submenu of the contextual menu.

To expand all child nodes, right-click the cell that contains the parent node and select **Expand All** from the **Expand/Collapse** submenu. To collapse all node, right-click any cell and select **Collapse All** from the **Expand/Collapse** submenu.

Editing Elements or Attributes

To edit elements or attributes in **Grid** mode, simply double-click the cell that contains the element or attribute (or select the cell and press **Enter**) to invoke the *Content Completion Assistant (on page 275)*. This opens a pop-up window that offers a list of proposals that are valid for that particular node.

Editing Text Content in Cells

To edit the text value of a cell, simply select the grid cell and press **Enter** (or double-click the cell), and start editing.

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.

Editing the Structure of the Nested Tables

To edit the structure of the nested tables in **Grid** mode, Oxygen XML Developer Eclipse plugin provides the following actions in the contextual menu (many of them also appear in the submenus of the **Document** menu, or the toolbar):

Cut, Copy, Paste, Delete common editing actions

Executes the typical editing actions on the currently selected elements. The **Cut** and **Copy** operations preserve the styles of the copied content.

Paste as Child

Pastes the copied content as the last child of the current selection.

Duplicate

Creates a new node by duplicating the currently selected one.

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.

Sort Ascending, Sort Descending

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*.

Insert Row

Inserts a new row below the current selection. To insert a new row, you could also select the row header (the zone to the left of the row that holds the row number) and press **Enter**.

Insert Column

Inserts a column after the current selection.

Clear Content

Removes all content from the current cell.

Expand/Collapse > Expand All

Expands the selection and all its children.

Expand/Collapse > Collapse All

Collapses the selection and all its children.

Expand/Collapse > Expand Children

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

Expand/Collapse > Collapse Children

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

Expand/Collapse > Collapse Others

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

Refresh Selected

Forces the layout to be recomputed.

Related Information:

Grid Mode Navigation *(on page 269)*

Copy and Paste in the Grid Editing Mode *(on page 273)*

Drag and Drop in the Grid Editing Mode *(on page 272)*

Content Completion Assistant in Grid Mode *(on page 275)*

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**.

Tip: When using drag and drop to reorganize the document, the resulting layout can be different from what you expected. 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.

Copy and Paste in the Grid Editing Mode

Selecting content in the **Grid** mode is similar to working with any table with a little more complexity. Specifically, depending on the type of node, when you select a cell, the selection may automatically include additional cells that are implied by the currently selected node. For example, if you click a node that contains any child nodes, all cells that contain the parent and child nodes will be selected. In this case, 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

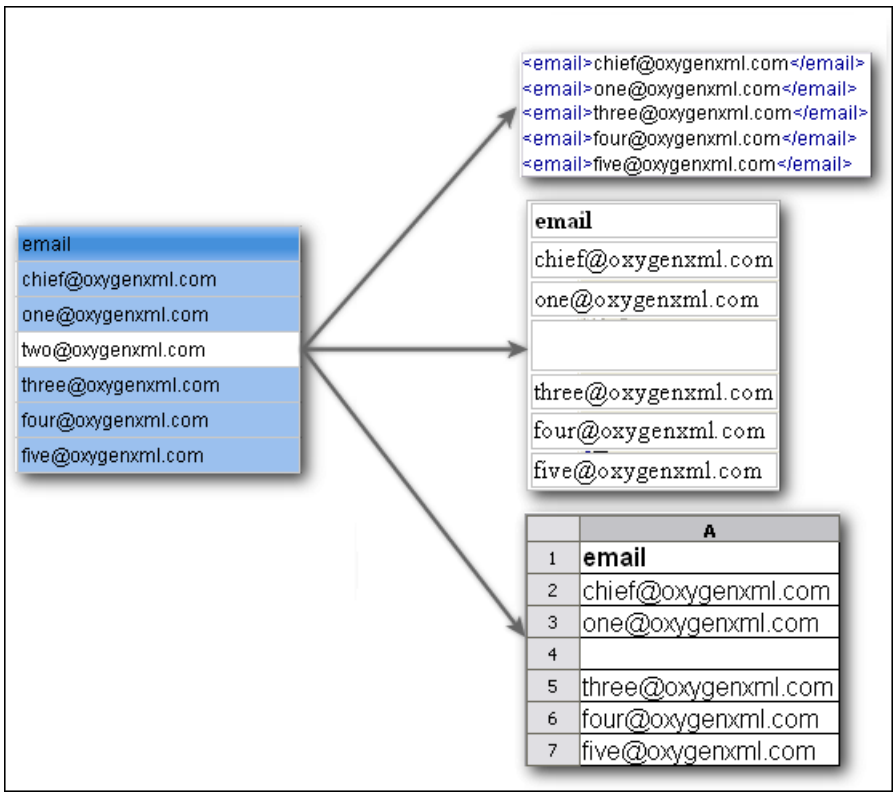
You can paste copied nodes relative to the currently selected cell using one of the following actions (available in the contextual menu):

- **Paste (Ctrl + V (Command + V on OS X))** - Pastes copied content, as a sibling, just below (after) the current selection.
- **Paste as Child** - Pastes copied 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 **Text** mode or other external applications. When pasting copied content from **Grid** mode, 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.

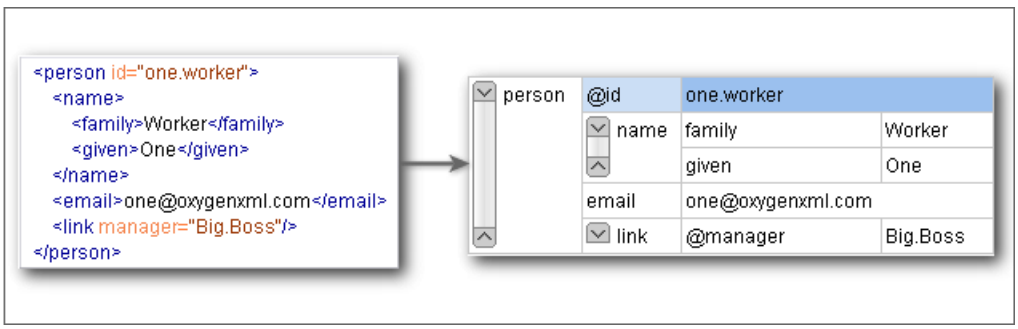
Figure 67. Copying from Grid to Other Editors



Pasting Content from Other Editors into Grid Mode

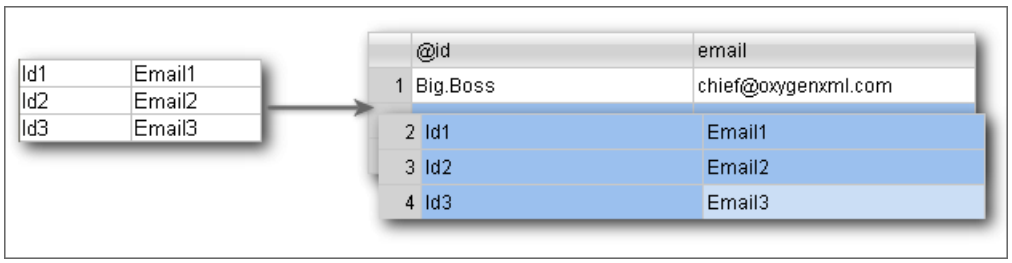
You can also paste well-formed XML content or tab-separated values from other editors into the **Grid** editor. If you paste XML content, the result will be the insertion of the nodes obtained by parsing this content.

Figure 68. Pasting XML Data into Grid



If the pasted text contains multiple lines of tab-separated values, it can be considered as a matrix of values. By pasting this matrix of values into the **Grid** editor, the result will be a matrix of cells. If the operation is performed inside existing cells, the existing values will be overwritten and new cells will be created when needed.

Figure 69. Pasting Tab-Separated Values into Grid



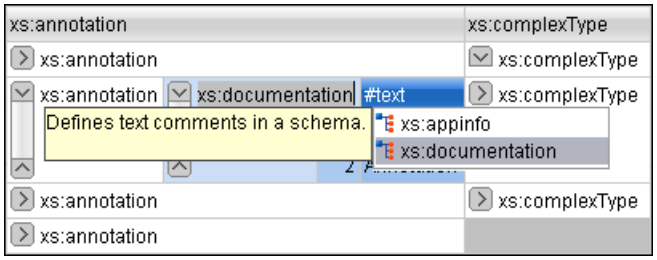
If you need to add copied content to your existing content (rather than overwriting existing cells), you need to first insert new cells by using the **Insert row** or **Insert column** actions from the contextual menu. This is useful, for example, when trying to transfer data from spreadsheet-like editors to the **Grid** editor.

Content Completion Assistant in Grid Mode

If the edited document is associated with a schema (DTD, XML Schema, Relax NG, etc.), the **Grid** editing mode offers a *Content Completion Assistant (on page 1324)* for the names and values of elements and attributes. If you choose to insert an element that has required content, the sub-tree of needed elements and attributes are also automatically included.

To display the content completion pop-up menu, simply double-click a cell that contains an element or attribute (or press **Enter** on your keyboard).

Figure 70. Content Completion in Grid Editing Mode



Special Character Support in Grid Mode

If you are editing documents with a bidirectional text orientation or other special characters (such as combining characters), 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.

Figure 71. Default left to right text orientation

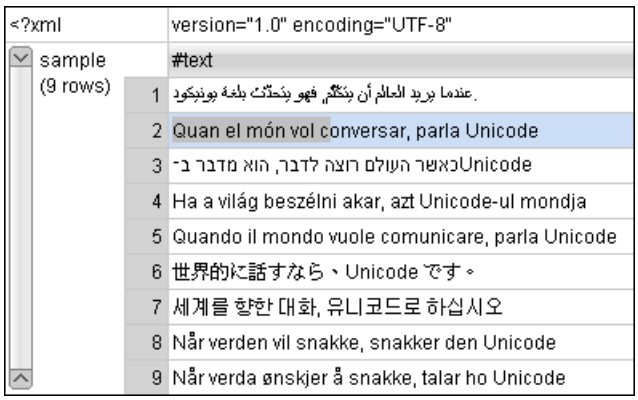
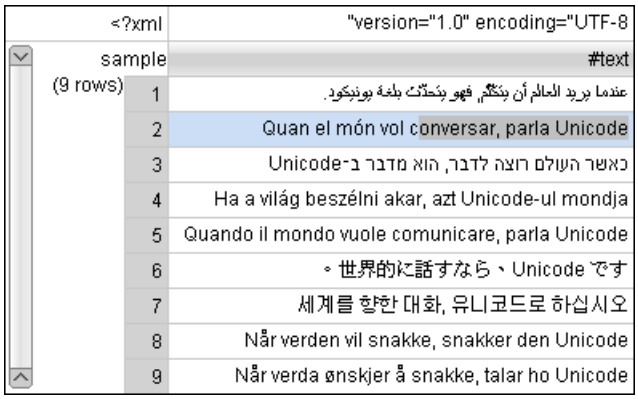


Figure 72. Right to left text orientation



Related Information:

Inserting Special Characters with the Character Map *(on page 217)*

Exporting XML Content to Excel

For use-cases where you have XML content that needs to be exported to Excel (or any other spreadsheet application) but the content is not already in some sort of table format, **Grid** mode offers you a way to display the content of an XML document as a structured grid of nested tables and you can work with the cells in those tables much like you would with any spreadsheet application. This makes it possible to export content to Excel by copying cells that contain the specific content and then pasting the copied cells in Excel the same as you would when working with any table or spreadsheet.

To export XML content from **Grid** mode to Excel or other spreadsheet applications, follow this procedure:

1. Open the XML document in Oxygen XML Developer Eclipse plugin and switch to **Grid** mode.
2. Expand the nodes *(on page 270)* to gain access to the particular nested table that contains the content you want to export.
3. Copy the cells that contain the content you want to export (**Copy** from the contextual menu or **Ctrl +C**).
4. Switch to your spreadsheet application and paste the copied cells.

5. You may need to make some manual adjustments depending on the complexity of the structure in the original XML document.

Note that Oxygen XML Developer Eclipse plugin also supports the reverse scenario (copying cells from a spreadsheet application and pasting them in **Grid** mode). For more information, see Import from MS Excel Files – Grid Mode Method *(on page 1200)*.

For more information about exchanging data between Oxygen XML Developer Eclipse plugin and spreadsheet applications, watch our video demonstration:

https://www.youtube.com/embed/8VwsF58zLkU

Related Information:

Import from MS Excel Files - Grid Mode Method *(on page 1200)*

Pasting Content from Other Editors into Grid Mode *(on page 274)*

Validating XML Documents

The W3C XML specification states that a program should not continue to process an XML document if it finds a validation error. The reason is that XML software should be easy to write and all XML documents should be compatible. With HTML, 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 Eclipse plugin provides validation functions that allow you to easily identify errors and their location.

Related Information:

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* include:

- 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 include:

- 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 include:

- 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* and *Namespace-valid*:

- Select the **Check Well-Formedness (Alt + Shift + V, W (Command + Alt + V, W on OS X))** action from the **Validation** drop-down menu on the toolbar (or the **XML** menu).
- A selection of files can be checked for well-formedness by selecting the **Check Well-Formedness** action from the **Validate** submenu when invoking the contextual menu in the **Navigator** view *(on page 192)*.

Result: 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.

Example: 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>.

Example: A non *Namespace-wellformed* Document

```
<prefix:elem></prefix:elem>
```

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

```
The prefix "prefix" for element "prefix:elem" is not bound.
```

Example: 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.

This section contains topics that explain the automatic and manual validation possibilities in Oxygen XML Developer Eclipse plugin, how validation errors are presented, and information about built-in and custom validation scenarios.

For information about how to associate a schema for the purposes of validation (and content completion), see the Associating a Schema to XML Documents *(on page 303)* section.

Automatic Validation

By default, Oxygen XML Developer Eclipse plugin automatically checks for validation errors in the document as you are editing. The **Enable automatic validation** option *(on page 84)* in the **Document Checking** preferences page *(on page 84)* 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 *(on page 84)* 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 279)*

Presenting Validation Errors in Text Mode *(on page 281)*

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 Eclipse plugin.

Tip: Status information generated by certain operations (such as *validation*) are fed into the **Console** view. This could be helpful for troubleshooting problems encountered during such operations. To open this view, select **Window > Show View > Console**.

Manual Validation Actions

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 *(on page 192)*.

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 (on page 1329)* 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 *(on page 306)*.

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 *(on page 192)*.

This action opens a dialog box that allows you to specify a schema for validating all selected files *(on page 307)*.

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 *(on page 84)*.

Related Information:

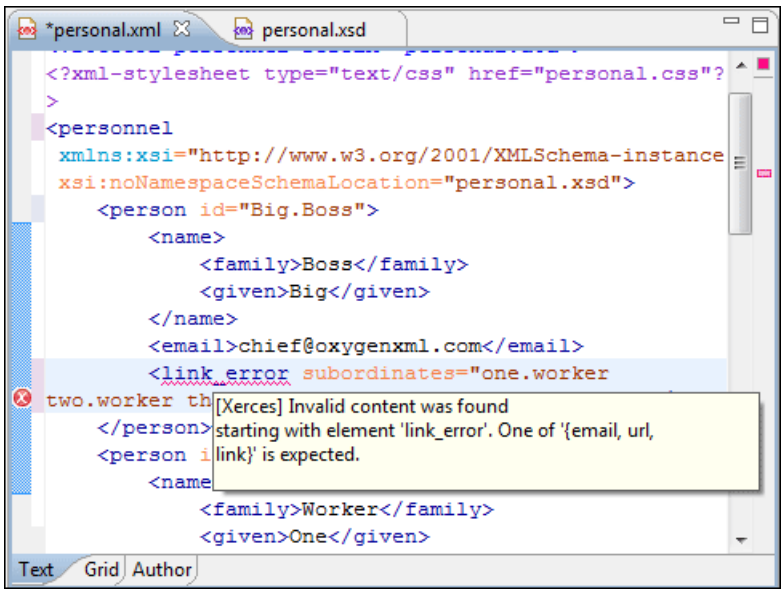
Automatic Validation *(on page 279)*

Presenting Validation Errors in Text Mode *(on page 281)*

Presenting Validation Errors in Text Mode

By default, Oxygen XML Developer Eclipse plugin automatically validates documents *(on page 279)* while editing in the **Text** mode, and actions are also available to manually validate documents *(on page 279)* on-request.

Figure 73. Presenting Validation Errors in Text Mode



Validation Marker Locations

In **Text** mode, validation issues are marked in the following locations:

- In the main editing pane, with the issue underlined in a color according to the type of issue.
- In the right-side vertical stripe, with a marker that is colored according to the type of issue.
- For attributes with detected issues, in the **Attributes** view *(on page 243)*, with the attribute and its value colored according to the type of issue.

Validation Marker Colors

The colors for each type of issue are as follows:

- **Validation Errors** [Red] - By default, the underline in the editing pane, the marker in the right vertical stripe, and the foreground color of the attribute in the **Attributes** view are colored in red.
- **Validation Warnings** [Yellow] - By default, the underline in the editing pane, the marker in the right vertical stripe, and the foreground color of the attribute in the **Attributes** view are colored in yellow.
- **Validation Info** [Blue] - By default, the underline in the editing pane, the marker in the right vertical stripe, and the foreground color of the attribute in the **Attributes** view are colored in blue.

You can configure the colors and how the various types of validation problems are rendered from the Eclipse **Annotations** preferences page (**Window ('Eclipse' on Mac OSX) > Preferences > General > Editors > Text Editors > Annotations**).

Validation Markers in the Right-Side Stripe

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 tooltip when you hover over indicator square. If there are numerous problems, only the first three are presented in the tooltip.

Middle Part of the Stripe

Errors are presented 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 *(on page 36)*, go to **Editor > Document checking**, and specify the desired limit in the **Maximum number of validation highlights** option *(on page 84)*.

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

Bottom Part of the Stripe

Two navigation arrows ( ) can be used to jump to the next or previous issue. The same actions can be triggered from **Document > Automatic validation > Next Error/Highlight** (**Ctrl + Period (Command + Period on OS X)**) and **Document > Automatic validation > Previous Error/Highlight** (**Ctrl + Comma (Command + Comma on OS X)**).

Hovering Over Validation Issues

Hovering over a validation issue presents a tooltip message with more details about the problem and possible *quick fixes (on page 301)* (if available for that issue). Also, when hovering over an issue, pressing **F2** will change the focus to the tooltip where you can use **Tab** and **Shift + Tab** to navigate between quick fixes and **Space** to trigger them.

Details About Validation Issues

- Information about the issue is also displayed in the message area on the bottom of the editor panel (clicking the **Document checking options** button opens the **Document Checking** preferences page *(on page 84)* where you can configure some validation options.
- Status messages from every validation action are logged in the **Console** view *(on page 220)* (the **Enable Oxygen consoles** option *(on page 112)* must be selected in the **View** preferences page).
- If you want to see all the validation messages grouped in the **Results** view *(on page 248)*, 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 279)*

Presenting Schematron Validation Issues *(on page 591)*

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 *(on page 82)*. After a custom validation engine is properly configured *(on page 82)*, 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 *(on page 285)* of Oxygen XML Developer Eclipse plugin for linked messages:

- **Saxon-EE** - Included in Oxygen XML Developer Eclipse 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 *(on page 126)*.

- **MSXML 4.0** - Included in Oxygen XML Developer Eclipse 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 Eclipse 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.
- **LIBXML** - Not included in Oxygen XML Developer Eclipse plugin and, depending on your operating system, the libraries need to be downloaded and installed separately from http://xmlsoft.org/downloads.html. Afterward, the `PATH` environment variable needs to be updated to contain the parent folder of the `xmllint` executable. Alternatively, you can go to **Options > Preferences > Editor > Custom Validation Engines**, edit the **LIBXML** validation engine and set a custom path to the `xmllint` executable.

  The LIBXML validator is associated with the 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. Support for *XML Catalogs (on page 1329)* (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 (on page 1329)* files of the built-in document types (DocBook, TEI, DITA, etc.) are not handled by LIBXML if Oxygen XML Developer Eclipse 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.
- **XSV** - Not included in Oxygen XML Developer Eclipse 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 Eclipse plugin *(on page 82)* 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. *(on page 82)* 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 Eclipse 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. *(on page 82)* 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 Eclipse 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:

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
```

Using Saxon Integrated Extension Functions

Saxon, the transformation and validation engine used by Oxygen XML Developer Eclipse plugin, can be customized by adding custom functions (called Integrated Extension Functions) that can be called from XPath.

To define such a function, follow these steps:

1. Create a file with a Java class that extends `net.sf.saxon.lib.ExtensionFunctionDefinition`. Here is an example:

```
private static class ShiftLeft extends ExtensionFunctionDefinition {
  @Override
  public StructuredQName getFunctionQName() {
    return new StructuredQName("eg", "http://example.com/saxon-extension", "shift-left");
  }
  @Override
  public SequenceType[] getArgumentTypes() {
   return new SequenceType[] {SequenceType.SINGLE_INTEGER, SequenceType.SINGLE_INTEGER};
  }
  @Override
  public SequenceType getResultType(SequenceType[] suppliedArgumentTypes) {
   return SequenceType.SINGLE_INTEGER;
  }
  @Override
  public ExtensionFunctionCall makeCallExpression() {
   return new ExtensionFunctionCall() {
   public SequenceIterator call(SequenceIterator[] arguments, XPathContext context)
                    throws XPathException {
       long v0 = ((IntegerValue)arguments[0].next()).longValue();
    long v1 = ((IntegerValue)arguments[1].next()).longValue();
    long result = v0<<v1;
        return Value.asIterator(Int64Value.makeIntegerValue(result));
      }
    };
  }
}
```

2. Compile the class and add it to a JAR file.
3. Add a file called **net.sf.saxon.lib.ExtensionFunctionDefinition** that contains the fully qualified name of the Java class in the `META-INF/services/` folder of the JAR file.

Note: To add more function definitions in the same JAR file, you need to add their fully qualified names on different lines.

To enable Oxygen XML Developer Eclipse plugin to pick up your custom function definition, the JAR file should be added to the classpath of the transformer. Here are some possibilities:

- If you develop a framework, you just need to link the JAR file in the **Classpath** tab *(on page 55)*.
- In a validation scenario *(on page 288)*, you can use the **Extensions** button to open a dialog box where you can add libraries.
- In a transformation scenario, you can use the **Extensions** button in the **XSLT** tab *(on page 693)* to open a dialog box where you can add libraries.

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 Eclipse 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 regarding 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 validations. 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 Eclipse plugin (Saxon 9) or any connection to a database that supports validation (Berkeley DB XML Database, eXist XML Database, MarkLogic version 5 or newer) can be set as a validation engine.
- An XML document where the *master file (on page 1326)* 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*, along with any Schematron schema defined in the default scenarios for that particular *framework*, are used for the validation. These take precedence over other types of validation units defined in the default scenarios for the particular *framework*. For more information on *master files*, see Master Files Support *(on page 200)* or Working with Modular XML Files in the Master Files Context *(on page 315)*.

Tip: Status information generated by certain operations (such as *validation*) are fed into the **Console** view. This could be helpful for troubleshooting problems encountered during such operations. To open this view, select **Window > Show View > Console**.

Related Information:

Validating XML Documents Against a Schema *(on page 279)*

Presenting Validation Errors in Text Mode *(on page 281)*

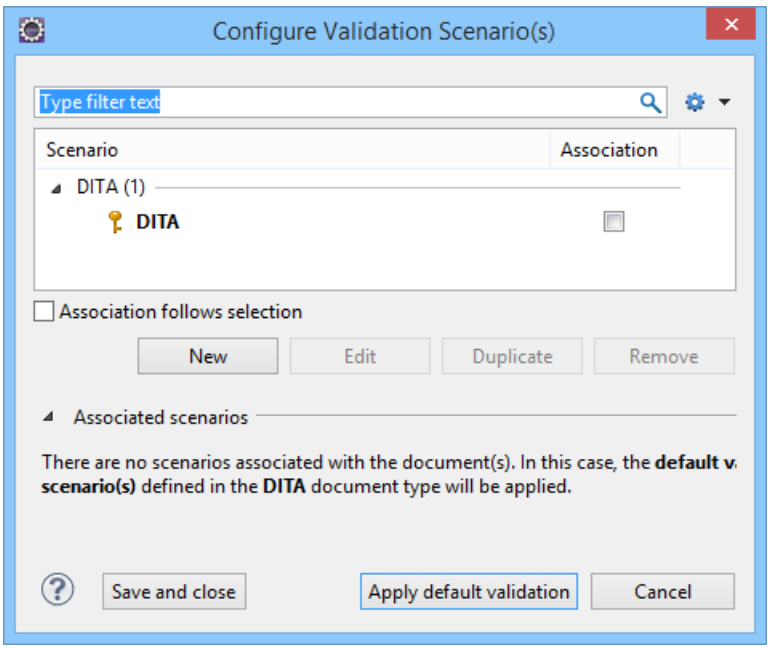
Creating a New Validation Scenario

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 *(on page 192)*).
   The **Configure Validation Scenario(s)** dialog box is displayed. It contains built-in and user-defined scenarios. The built-in 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 (on page 1325)* are rendered in bold.

   Note: If a *master file* is associated with the current file, the validation scenarios defined in the *master file*, along with any Schematron schema defined in the default scenarios for that particular *framework*, are used for the validation. These take precedence over other types of validation units defined in the default scenarios for the particular *framework*. For more information on *master files*, see Master Files Support *(on page 200)* or Working with Modular XML Files in the Master Files Context *(on page 315)*.

Figure 74. 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 Eclipse 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 Eclipse 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.

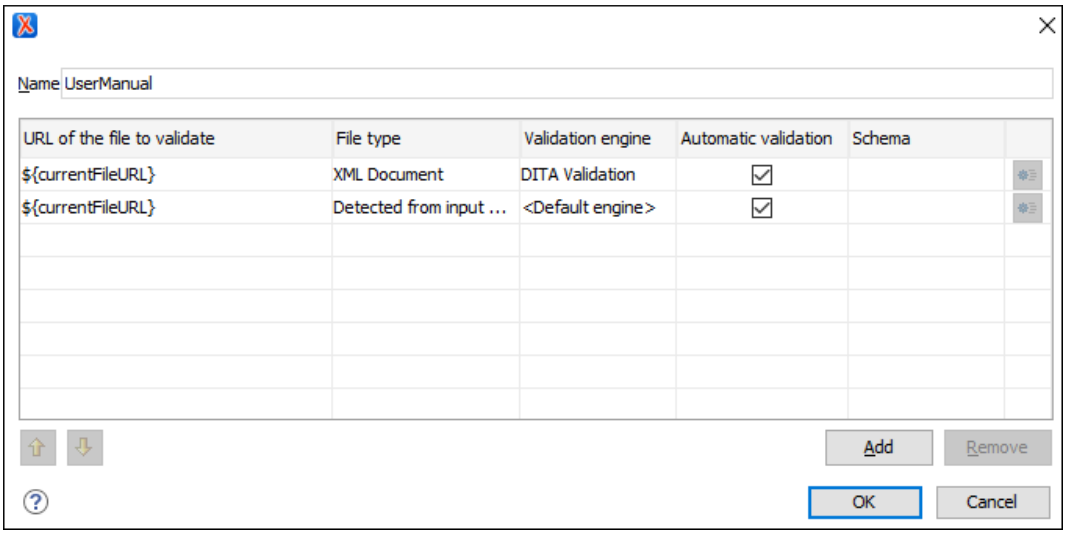
Note: When you keep both scenarios, Oxygen XML Developer Eclipse plugin adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export selected scenarios individually. Oxygen XML Developer Eclipse plugin creates a `scenarios` file that contains the exported scenarios. 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.
   A validation scenario configuration dialog box is displayed and it lists all the validation units for the scenario.

Figure 75. Validation Scenario Configuration Dialog Box



This scenario configuration dialog box allows you to configure the following information and options:

Name

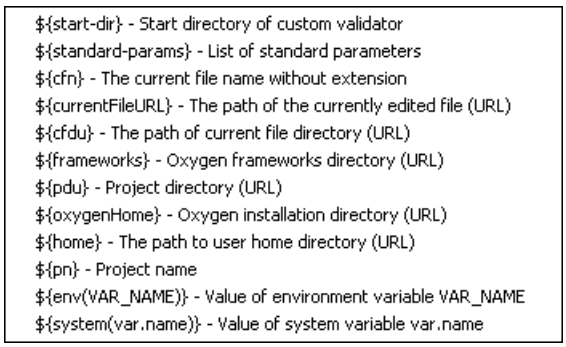
The name of the validation scenario.

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 *(on page 148)* or a custom editor variable *(on page 155)*.

Figure 76. Insert an Editor Variable



File type

The type of the document that is validated in the current validation unit. Oxygen XML Developer Eclipse 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 Eclipse 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 *(on page 137)*, XQuery preferences page *(on page 134)*, XML Schema preferences page *(on page 126)*).

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 selected, the validation operation defined by this row is also applied by the automatic validation feature *(on page 279)*. If the **Automatic validation** feature is disabled in the Document Checking preferences page *(on page 84)*, then this option is ignored, as the preference setting has a higher priority.

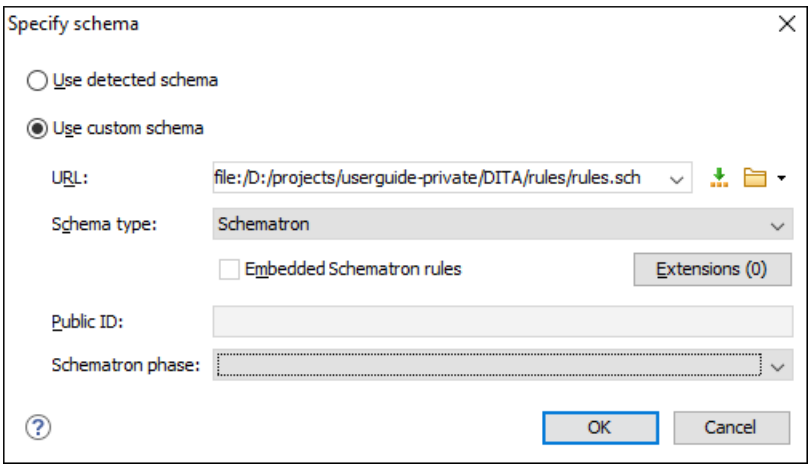
Schema

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 77. Specify Schema Dialog Box



The **Specify Schema** dialog box contains the following options:

Use detected schema

Uses the schema detected for the particular document *(on page 304)*.

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** *(on page 148)* button, or the browsing actions 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, select this option.
- **Extensions**- Opens a dialog box that allows you to specify *Java extension JARs (on page 1326)* 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. Click **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 click the **Apply associated** button to run the validation scenario.

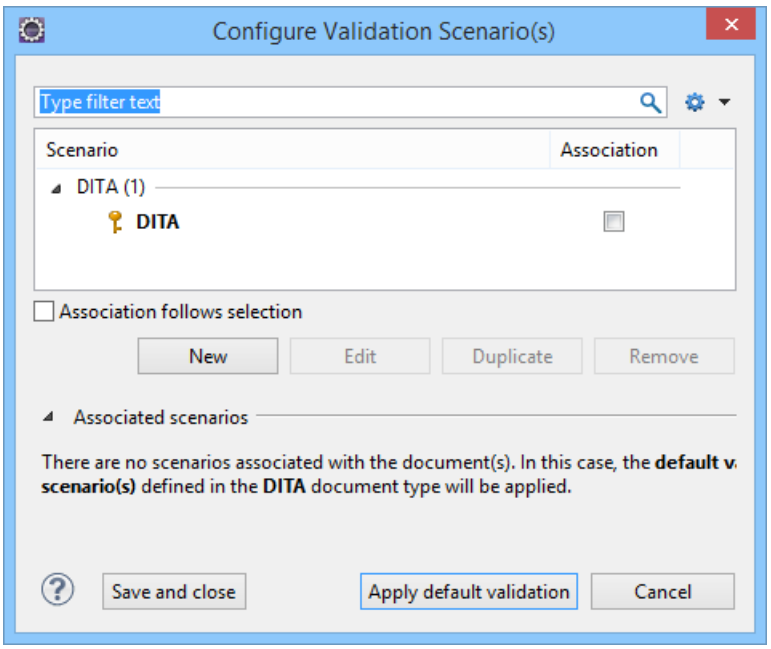
Editing a Validation Scenario

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 *(on page 192)*).
   The **Configure Validation Scenario(s)** dialog box is displayed. It contains built-in and user-defined scenarios. The built-in 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 (on page 1325)* are rendered in bold.

   Note: If a *master file* is associated with the current file, the validation scenarios defined in the *master file*, along with any Schematron schema defined in the default scenarios for that particular *framework*, are used for the validation. These take precedence over other types of validation units defined in the default scenarios for the particular *framework*. For more information on *master files*, see Master Files Support *(on page 200)* or Working with Modular XML Files in the Master Files Context *(on page 315)*.

Figure 78. 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 Eclipse 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 Eclipse 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.

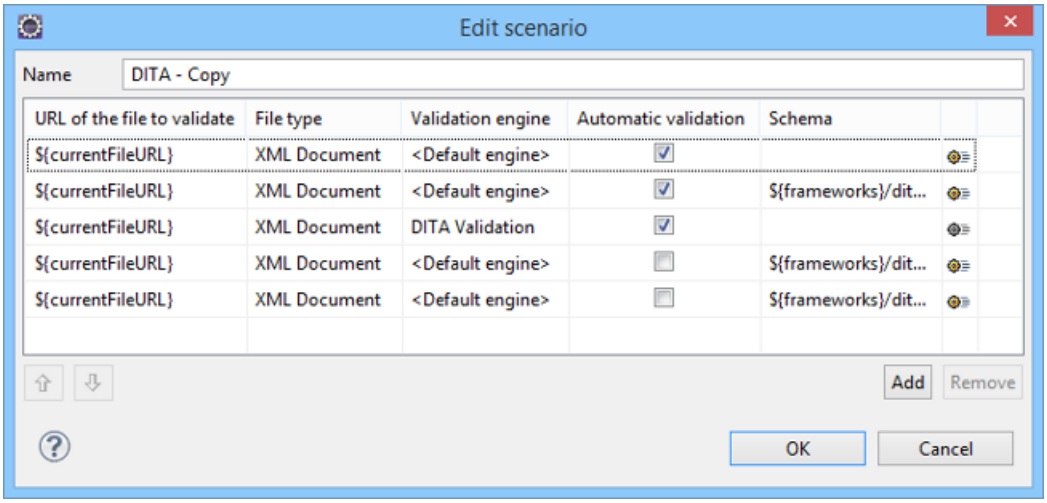
Note: When you keep both scenarios, Oxygen XML Developer Eclipse plugin adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export selected scenarios individually. Oxygen XML Developer Eclipse plugin creates a `scenarios` file that contains the exported scenarios. This is useful if you want to share scenarios with others or export them to another computer.

2. Select the scenario and click the **Edit** button. If you try to edit one of the *read-only* built-in scenarios, you will receive a warning message that Oxygen XML Developer Eclipse plugin needs to creates 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 79. Edit Validation Scenario



This scenario configuration dialog box allows you to configure the following information and options:

Name

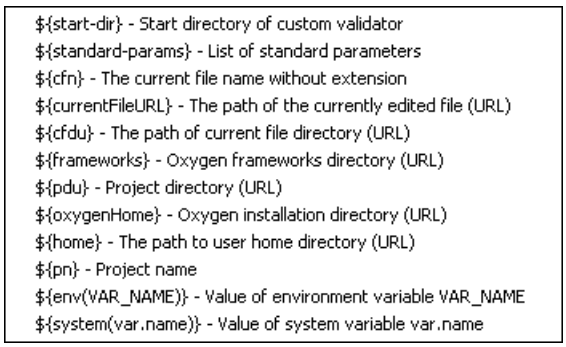
The name of the validation scenario.

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 *(on page 148)* or a custom editor variable *(on page 155)*.

Figure 80. Insert an Editor Variable



File type

The type of the document that is validated in the current validation unit. Oxygen XML Developer Eclipse 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 Eclipse 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 *(on page 137)*, XQuery preferences page *(on page 134)*, XML Schema preferences page *(on page 126)*).

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 selected, the validation operation defined by this row is also applied by the automatic validation feature *(on page 279)*. If the **Automatic validation** feature is disabled in the Document Checking preferences page *(on page 84)*, then this option is ignored, as the preference setting has a higher priority.

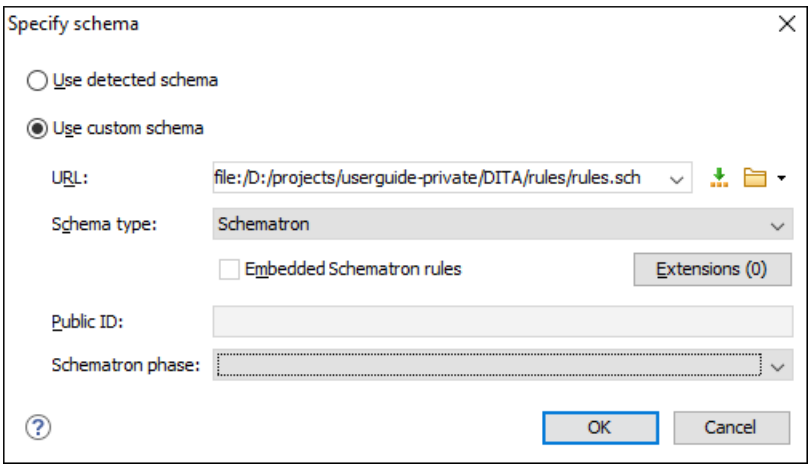
Schema

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 81. Specify Schema Dialog Box



The **Specify Schema** dialog box contains the following options:

Use detected schema

Uses the schema detected for the particular document *(on page 304)*.

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** *(on page 148)* button, or the browsing actions 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, select this option.
- **Extensions**- Opens a dialog box that allows you to specify *Java extension JARs (on page 1326)* 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, click **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 *(on page 147)* 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 (on page 1329)*.

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 an 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"/>
```

Related Information:

Working with XML Catalogs *(on page 312)*

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|funcsynopsis
|classsynopsis|fieldsynopsis|constructorsynopsis|destructorsynopsis|methodsynopsis
|formalpara|para|simpara|address|blockquote|graphic|graphicco|mediaobject|mediaobjectco
|informalequation|informalexample|informalfigure|informaltable|equation|example|figure
|table|msgset|procedure|sidebar|qandaset|task|anchor|bridgehead|remark|highlights
|abstract|authorblurb|epigraph|indexterm|beginpage)+".
```

This error message is a little more difficult to understand, so understanding of the syntax or processing rules for the DocBook XML DTD `<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 read about the element. 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.

Embedding Schematron Rules in XML Schema or RELAX NG

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 Eclipse plugin supports Schematron validation schemas and it is able to extract and use the embedded rules.

Validating XML Documents with XML Schema and Embedded Schematron

To validate an XML document with XML Schema and its embedded Schematron, you can associate the document like this:

```
<?xml-model href="percent.xsd" type="application/xml"
            schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

Validating XML Documents with Relax NG and Embedded Schematron

To validate an XML document with RELAX NG schema and its embedded Schematron rules, you need to associate the document with both schemas like this:

```
<?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.

Note: When you work with XML Schema or Relax NG documents that have embedded Schematron rules Oxygen XML Developer Eclipse 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 XML Schema

```
<xsd:appinfo>
  <sch:pattern>
    <sch:rule context="...">
 <sch:assert test="...">Message.</sch:assert>
    </sch:rule>
  </sch:pattern>
</xsd:appinfo>
```

Example: Embedded Schematron in Relax NG Schema

```
<grammar
    xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:sch="http://purl.oclc.org/dsdl/schematron" >
    <sch:pattern>
      <sch:rule context="...">
        <sch:assert test="...">Message.</sch:assert>
      </sch:rule>
    </sch:pattern>
    <start>
     ..............
    </start>
</grammar>
```

Related Information:

Embedding Schematron Quick Fixes in Relax NG or XML Schema *(on page 620)*

XML Quick Fixes

The Oxygen XML Developer Eclipse plugin *Quick Fix* support *(on page 1328)* 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

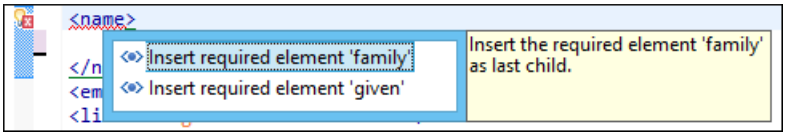
To activate this feature, hover over or place the cursor in the highlighted area of text where a validation error or warning occurs. If a *Quick Fix* is available for that particular error or warning, you can access the *Quick Fix* proposals with any of the following methods:

- **When** hovering over the error or warning, the proposals may be presented in a tooltip pop-up window and the available quick *Quick Fixes* include a link that can be used to perform the fix.

  Figure 82. 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 Eclipse plugin displays the list of available fixes.

Figure 83. 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** preferences *(on page 77)* are configured.

Quick Fixes for DTD, XSD, and Relax NG Errors

Oxygen XML Developer Eclipse plugin offers *Quick Fixes (on page 1328)* 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 Eclipse 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 303)*

Schematron Quick Fixes (SQF)

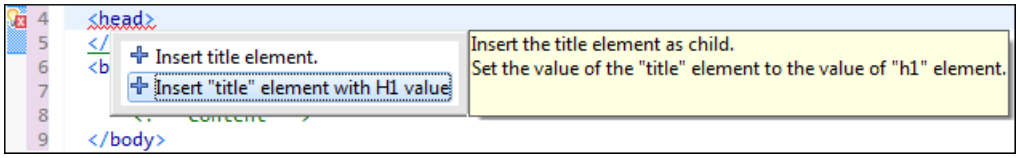
Oxygen XML Developer Eclipse plugin provides support for Schematron *Quick Fixes (on page 1328)* (SQF). They help you resolve issues that appear in XML documents that are validated against Schematron schemas by offering you solution proposals. The Schematron *Quick Fixes* are an extension of the Schematron 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, see the following blog post: https://blog.oxygenxml.com/topics/SchematronBCs.html.

Displaying the Schematron Quick Fix Proposals

The defined Schematron *Quick Fixes* are displayed on validation errors in **Text** mode.

Figure 84. Example of a Schematron Quick Fix



Related Information:

Editing Schematron Quick Fixes *(on page 606)*

Schematron Quick Fix Specifications

Presenting Schematron Validation Issues *(on page 591)*

Associating a Schema to XML Documents

To provide as-you-type validation and to compute valid proposals for the *Content Completion Assistant (on page 1324)*, Oxygen XML Developer Eclipse plugin requires a schema to be associated with the XML document. The schema specifies how the internal structure of the XML is defined.

Supported Types of Schema

The following schema types are supported:

- **W3C XML Schema 1.0 and 1.1** (with and without embedded Schematron rules) - 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 is in the namespace.
    - `@xsi:noNamespaceSchemaLocation` attribute, if the root element is not in the namespace.
- **DTD** - The association with a DTD is added as a `DOCTYPE` declaration.
- **Relax NG - XML Syntax** (with and without embedded Schematron rules) - The association is added as an *xml-model* processing instruction.
- **Relax NG - Compact Syntax** - The association is added as an *xml-model* processing instruction.
- **NVDL** - The association is added as an *xml-model* processing instruction.
- **Schematron** (both ISO Schematron and Schematron 1.5) - The association is added as an *xml-model* processing instruction.

Detecting the Schema(s) for Validation

For validation, Oxygen XML Developer Eclipse plugin tries to detect one or more schemas by searching multiple locations, in the following order:

1. The schema or multiple schemas referenced in validation stages from the validation scenario(s) *(on page 305)* associated with the current XML document.
2. If no validation scenario is selected to be used with the current XML document, then it falls back to the schema or multiple schemas defined in validation stages from the validation scenarios specified as default in the particular document type configuration *(on page 308)*.
3. If a schema is still not detected, then it falls back to the schema or multiple schemas associated directly in the XML document *(on page 310)*.

   Tip: To quickly open the schema used for validating the current document, select the **Open Associated Schema** action from the toolbar (or **XML** menu).
4. If a schema is still not detected, then it falls back to the schema defined in the **Schema** tab of a *framework* (document type) configuration *(on page 311)*.

Detecting a Schema for Content Completion

For content completion, Oxygen XML Developer Eclipse plugin uses just one schema and tries to detect that schema by searching multiple locations, in the following order:

1. If no schema is detected in the document, then it falls back to the highest ranking schema defined in validation stages from the validation scenario(s) associated with the current document *(on page 305)*.
2. If a schema is still not detecting, then it falls back to the highest ranking schema defined in validation stages from validation scenarios specified as default in the particular document type configuration *(on page 308)*.
3. Oxygen XML Developer Eclipse plugin determines the most appropriate or highest ranking schema that is associated directly in the XML document *(on page 310)* and uses it for content completion.
4. If a schema is still not detecting, then it falls back to the schema defined in the **Schema** tab of a *framework* (document type) configuration *(on page 311)*.

Related Information:

W3C: Associating Schemas with XML Documents

Associating a Schema Through a Validation Scenario

Oxygen XML Developer Eclipse 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 *(on page 287)* and there are several methods that can be used to do so.

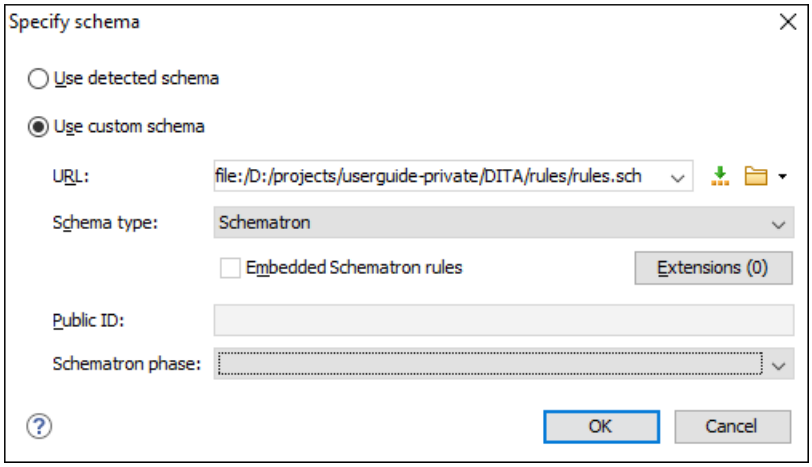
Configure a Validation Scenario and Specify the Schema

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 *(on page 192)*).
2. Click the **New** button to create a new validation scenario or the **Edit** button to modify an existing one.
3. Add or configure validation units *(on page 295)* according to your needs and click the **Specify Schema** button.

   Step Result: The **Specify Schema** dialog box is displayed:

   Figure 85. Specify Schema Dialog Box

   The **Specify Schema** dialog box contains the following options:

   Use detected schema

   Uses the schema detected for the particular document *(on page 304)*.

   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** *(on page 148)* button, or the browsing actions 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, select 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 (on page 1326)* 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 86. 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** *(on page 148)* button, or the browsing actions 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, select 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 (on page 1326)* 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 87. 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** *(on page 148)* button, or the browsing actions 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, select 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 (on page 1326)* 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 Eclipse plugin uses rules defined in the schema that is detected in the validation scenarios that are associated to each particular document type.

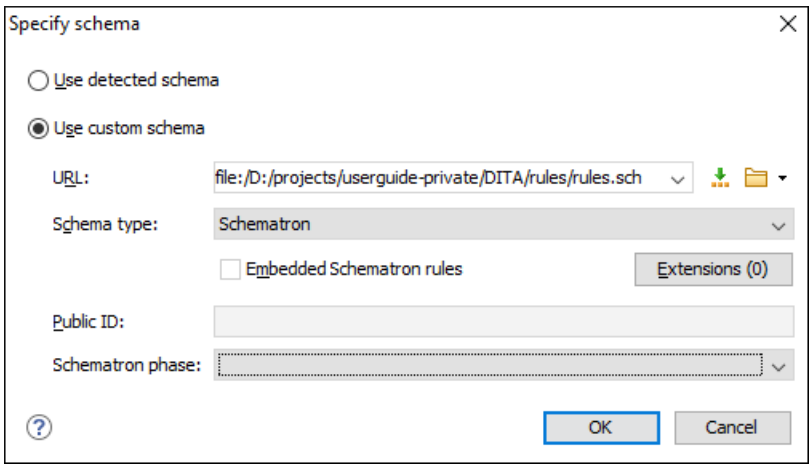
To associate a schema in validation scenarios defined in the *framework (on page 1325)* (document type) configuration, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* 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 *(on page 52)*.
3. Go to the **Validation** tab *(on page 74)*.
4. Create or edit a validation scenario:
   a. To create a new validation scenario *(on page 288)*, click the **New** button.
   b. To edit the properties of an existing validation scenario *(on page 293)*, 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 *(on page 295)* according to your needs and click the **Specify Schema** button.

   Step Result: The **Specify Schema** dialog box is displayed:

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 *(on page 304)*.

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** *(on page 148)* button, or the browsing actions 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, select 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 (on page 1326)* 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

The schema used by the *Content Completion Assistant (on page 1324)* and document validation engine can be directly 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 or `@xsi:noNamespaceSchemaLocation` attribute is used.
- **DTD** - The `DOCTYPE` declaration is used.

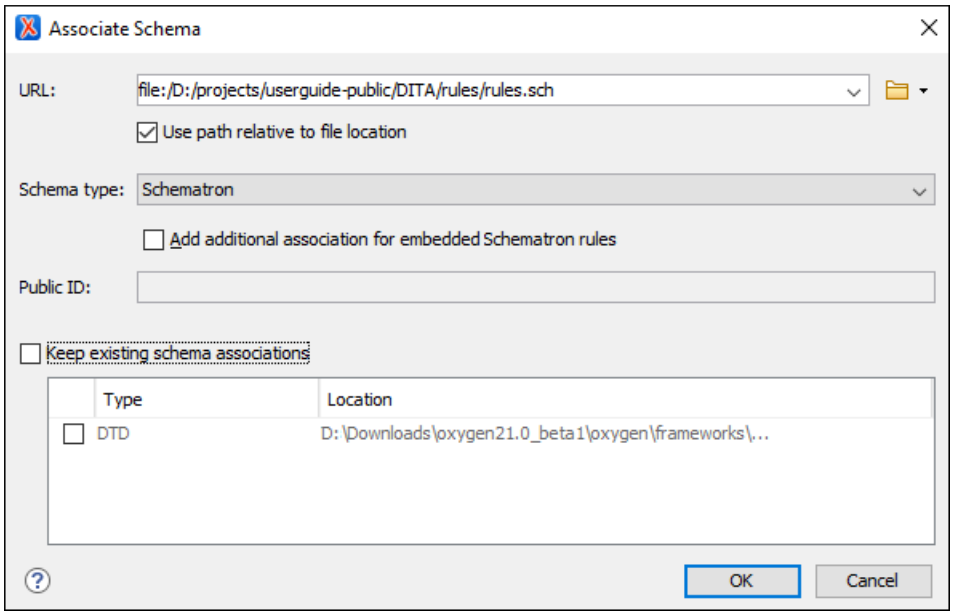
The association can specify a relative file path or a URL of the schema. The advantage of relative file path is that you can configure the schema at file level instead of *framework (on page 1325)* level.

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 89. 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** - Select 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, select this option.
- **Public ID** - Allows you to specify a public ID if you have selected a DTD.
- **Keep existing schema associations** - Select 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 is in the namespace.
  - `@xsi:noNamespaceSchemaLocation` attribute, if the root element is not in the 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 an *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).

Related Information:

Validating XML Documents *(on page 277)*

Content Completion Assistant in Text Mode *(on page 234)*

Associating a Schema in a Framework (Document Type) Configuration

The schema used to compute valid proposals in the *Content Completion Assistant (on page 1324)* and by the document validation engine to report errors and warnings can be defined in each particular *framework (on page 1325)* (document type). This schema will be used only if one is not detected in the current XML file *(on page 310)*.

To associate a schema in a particular *framework* (document type), follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Document Type Association**.

2. Select your particular document type and click the **Edit** *(on page 50)*, **Extend** *(on page 50)*, or **Duplicate** *(on page 50)* 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 *(on page 52)*.
3. Go to the **Schema** tab *(on page 55)*.
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.

Learn Document Structure when Schema is not Detected

When working with documents that do not specify a schema, or the schema is not known or does not exist, Oxygen XML Developer Eclipse 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 *(on page 234)* and document validation *(on page 277)*. 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 Eclipse plugin automatically learns the document structure and uses it for content completion *(on page 234)*. To disable this feature, deselect the **Learn on open document** option in the user preferences *(on page 77)*.

Related Information:

Detecting a Schema *(on page 303)*

Create a DTD from Learn Document Structure Option

When there is no schema associated with an XML document, Oxygen XML Developer Eclipse plugin can learn the document structure by parsing the document internally. This feature is enabled by the **Learn on open document** option *(on page 77)* that is available in the user preferences.

To create a DTD from the learned structure, follow these steps:

1. Open the XML document that will be used to create the DTD.
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. Click the **Save** button.

Working with XML Catalogs

Oxygen XML Developer Eclipse plugin uses *XML Catalogs (on page 1329)* to resolve references for validations and transformations and they are especially helpful for resolving external resources when internet access is not available or your connection is slow.

Oxygen XML Developer Eclipse 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 Eclipse plugin uses different catalog mappings.

Table 3. Catalog Mappings

Doc Type	Referenced Resource	Mappings
XML	DTD	*system* or *public* The **Prefer** option *(on page 121)* 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 *URI* catalog mappings. 2. Resolve the schema using *system* catalog mappings. This happens only if the **Resolve schema locations also through system mappings** option *(on page 122)* is selected (it is by default). 3. Resolve the root *namespace* using *URI* catalog mappings.
	Relax NG	
	Schematron	
	NVDL	
XSL	XSL/ANY	*URI*
CSS	CSS	*URI*
JSON	JSON	*URI*
XML Schema	XML Schema	The following strategy is used (if one step fails to provide a resource, the next is applied): 1. Resolve schema reference using *URI* catalog mappings. 2. Resolve schema reference using *system* catalog mappings. This happens only if the **Resolve schema locations also through system mappings** option *(on page 122)* is selected (it is by default).
Relax NG	Relax NG	

Table 3. Catalog Mappings (continued)

Doc Type	Referenced Resource	Mappings
		3. Resolve schema *namespace* using *URI* catalog mappings. This happens only if the **Process namespaces through URI mappings for XML Schema** option *(on page 122)* is selected (it is not by default).

Creating an XML Catalog with a Template

An *XML Catalog (on page 1329)* file can be created quickly in Oxygen XML Developer Eclipse plugin starting from the two document templates called *OASIS XML Catalog 1.0* and *OASIS XML Catalog 1.1*. They are available when creating new document templates *(on page 177)*.

```
<?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 Eclipse plugin Determines which Catalog to Use

Oxygen XML Developer Eclipse plugin uses *XML Catalogs (on page 1329)* to resolve references for validations and transformations and it maps such references to the built-in local copies of the schemas associated with the various *frameworks (on page 1325)* (DocBook, DITA, TEI, XHTML, SVG, etc.)

Oxygen XML Developer Eclipse plugin includes default global catalogs as well as default catalogs for each of the built-in *frameworks*, and you can also create your own.

Oxygen XML Developer Eclipse plugin looks for catalogs in the following order:

- Global user-defined catalogs that are set in the **XML Catalog** preferences page *(on page 121)*.
- User-defined catalogs that are set for each *framework (on page 1325)* in the **Catalog** tab *(on page 73)* of the **Document Type** configuration dialog box *(on page 52)*.
- Default built-in catalogs.

Example:

An *XML Catalog* can be used to map an 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 121)*

Resolving Schema Locations Through XML Catalogs

Schema locations can be mapped using an *XML Catalog (on page 1329)*. Oxygen XML Developer Eclipse 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 end here.
- If the **Resolve schema locations also through system mappings** option *(on page 122)* is selected in the **XML Catalog** preferences page, it attempts to resolve the schema location as a system ID (`system`, `systemSuffix`, `rewriteSuffix`, `rerwriteSystem` from the *XML Catalog*). If this succeeds, the process ends here.
- If the **Process "schemaLocation" namespaces through URI mappings for XML Schema** option *(on page 122)* is selected in the **XML Catalog** preferences page, 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 *(on page 304)* is used.

Related Information:

Working with XML Catalogs *(on page 312)*

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 Eclipse 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 files (on page 1326)*.

You cat set a main XML document either using the *master files* support from the Navigator view *(on page 200)*, or using a validation scenario.

To set a *master file* using a validation scenario, add validation units that point to the main modules. Oxygen XML Developer Eclipse 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 (on page 1326)* include:

- Correct validation of a module in the context of a larger XML structure.
- *Content Completion Assistant (on page 1324)* displays all collected entities and IDs starting from the *master files*.
- Oxygen XML Developer Eclipse 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 *(on page 318)* for ID/IDREFS values and for updating references when renaming/moving a resource. Oxygen XML Developer Eclipse plugin performs the search and refactoring actions in the context that the *master files* determine, improving the speed of execution.

For more information about editing modular XML files in the *master files* context, watch our video demonstration:

https://www.youtube.com/embed/e2oo4RWNxW8

Related Information:

Master Files Support *(on page 200)*

XML Resource Hierarchy/Dependencies View *(on page 318)*

Search and Refactoring Actions for IDs and IDREFS

Oxygen XML Developer Eclipse plugin allows you to search for ID declarations and references (IDREFS) and to define the scope of the search and refactor operations *(on page 318)*. 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** *(on page 257)* 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 *(on page 318)*. 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** *(on page 318)* 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** *(on page 318)*.

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** *(on page 318)* 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** *(on page 318)*.

Note: Available in the **Text** mode only.

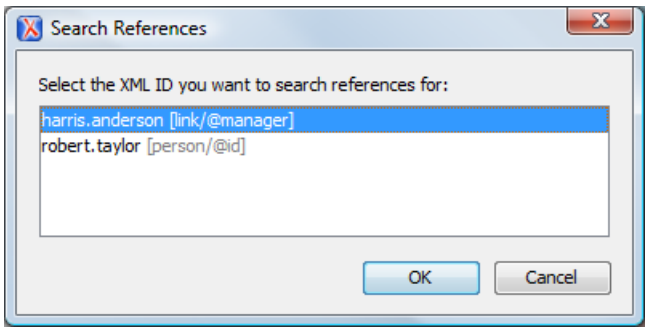
Search Occurrences in file

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 Eclipse 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 90. Selecting an ID in the Author Mode

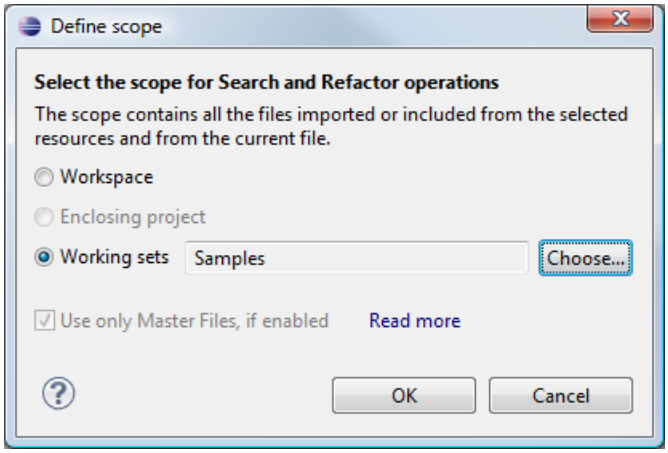


Related Information:

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 (on page 1329)*. 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 *(on page 200)*.

Figure 91. Change Scope Dialog Box



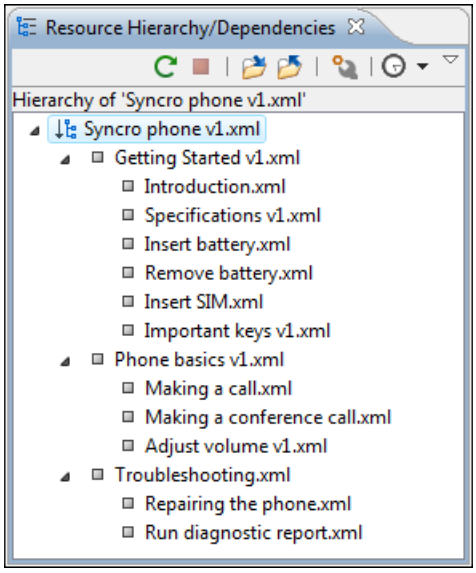
The scope you define is applied to all future search and refactor operations until you modify it. Contextual menu actions allow you to add or delete files, folders, and other resources to the *working set (on page 1329)* structure.

XML Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view displays the hierarchy or dependencies for resources included in an XML document. The tree structure presented in this view is built based on the *XIinclude* and *External Entity* mechanisms. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

If you want to see the hierarchy or dependencies of an XML document, select the document in the **Navigator** view *(on page 192)* and choose **Show Resource Hierarchy** or **Show Resource Dependencies** from the contextual menu.

Figure 92. Resource Hierarchy/Dependencies View



The following actions are available on the toolbar of 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 dependencies search scope

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 for a resource listed in the **Resource Hierarchy/Dependencies** view contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Go to reference

Opens the source document where the resource is referenced.

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 *(on page 200)*.

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 *(on page 320)*. Only the references made through the *XInclude* and *External Entity* mechanisms are handled.

Related Information:

Search and Refactor Operations Scope *(on page 318)*

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 of the renamed resource(s)** - Select this option to update the references to the resource you are renaming. A **Preview** option is available that allows you to see what will be updated before selecting **Rename** to process the operation.

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)** - Select this option to update the references to the resource you are moving, in accordance with the new location and name. A **Preview** option is available that allows you to see what will be updated before selecting **Move** to process the operation.

Combining XML Content Using DTD Entities and XInclude

When documenting large projects, it is likely that there are multiple people involved. In this case, it is usually more efficient to using a modular approach so that everyone involved in the project can work in parallel. Other advantages of modular documentation include: reusable content possibilities, smaller file units are easier to edit, and better version control.

Two possible solutions for this are to use **DTD Entities** or **XInclude** to combine XML content in a *master file* *(on page 1326)*. A master document can be created with references to various document parts, users can edit those documents individually, and then apply an XSLT stylesheet over the master document to obtain the output files in various formats (for example, PDF or HTML).

Combining XML Document Content Using DTD Entities

There are two conditions for including a document fragment using DTD entities:

- The master document should declare the DTD to be used, while the external entities should declare the XML fragments to be referenced.
- The referenced documents that contain the fragments cannot also define the DTD because the main document will not be valid. If you want to validate the parts separately you have to use XInclude *(on page 322)* for assembling the parts together with the *master file*.

The 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 (*testing.xml*) looks like this:

```
<section> ... here is 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.

The content from the referenced file (in the example above, it is a `<section>` in the *test.xml* file) can be inserted somewhere in the master document:

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

To obtain output in various formats (for example, PDF or HTML), you simply need to apply an XSLT stylesheet over the master document using a transformation scenario.

Viewing the Expanded Content in Oxygen XML Developer Eclipse plugin

When a transformation scenario is applied on the *master file*, an intermediary file combines all the referenced content and it will be expanded in the final output. If you want to see how the referenced content will be expanded before applying the transformationCreate a minimal XSLT stylesheet that simply copies the XML content, then create a transformation scenario that applies the stylesheet over the XML. The XSLT stylesheet would look like this:

```
<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="3.0">
    <xsl:template match="node() | @*">
        <xsl:copy>
            <xsl:apply-templates select="node() | @*"/>
        </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

Combining XML Documents and Fragments Using XInclude

XInclude is a standard for assembling XML instances into another XML document through inclusion. A *master file (on page 1326)* can be dynamically created from smaller XML documents without having to physically duplicate the content of the smaller files. The advantage of using XInclude instead of the DTD Entities method *(on page 321)* is that 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).

Enabling XInclude Support in Oxygen XML Developer Eclipse plugin

The XInclude support in Oxygen XML Developer Eclipse plugin is enabled by default. It is controlled by the **Enable XInclude processing** option *(on page 123)* in the **XML > XML Parser** preferences page *(on page 123)*. When enabled, Oxygen XML Developer Eclipse plugin will be able to validate and transform documents comprised of parts added using XInclude.

Example: Using XInclude to Combine Files

A 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>
```

The main article (*master 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.

Example: Using XInclude to Combine Fragments of Files

If you want to include only a fragment of a file in the *master file (on page 1326)*, 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.

Note: Oxygen XML Developer Eclipse 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 file (`a.xml`) 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>
```

Viewing the Expanded Content in Oxygen XML Developer Eclipse plugin

When a transformation scenario is applied on the *master file*, an intermediary file combines all the referenced content and it will be expanded in the final output. If you want to see how the referenced content will be expanded before applying the transformationCreate a minimal XSLT stylesheet that simply copies the XML content, then create a transformation scenario that applies the stylesheet over the XML. The XSLT stylesheet would look like this:

```
<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="3.0">
    <xsl:template match="node() | @*">
        <xsl:copy>
            <xsl:apply-templates select="node() | @*"/>
        </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

XInclude 1.1 Features

Oxygen XML Developer Eclipse plugin offers partial support for XInclude 1.1 features. This includes support for fragment identifiers and attribute copying.

- **Fragment Identifiers**

  You can use `<xi:include>` to reference a text file and specify the `@fragid` value so that you only get part of that text file in the main document. For some examples and to see how the `<xi:include>` gets expanded when the `@fragid` specifies a line range or character range, see **Textual Inclusion Examples with RFC5147 Fragment Identifiers**.
- **Attribute Copying**

  Any *namespaced* attribute defined on the `<xi:include>` element will be passed to the root element of the included content.

  For example, if you have this:

  ```
  <xi:include href="section2.xml" xmlns:xi="http://www.w3.org/2001/XInclude"
    xml:id="sectInner1"/>
  ```

  and `section2.xml` looks like this:

  ```
  <sect2 xmlns="http://docbook.org/ns/docbook" version="5.0"
      xmlns:xlink="http://www.w3.org/1999/xlink" xml:id="section2">
      <title>FS2</title>
      <para>P2</para>
  </sect2>
  ```

  then the final processed result will have the original `xml:id="section2"` replaced with the value specified in the *xi:included* section.

  For more information, see **Attribute Copying when Processing XML**. Also, to see more examples, see **Attribute Copying Examples**.

Related Information:

W3C Specifications: XML Inclusions (XInclude) Version 1.1

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 Eclipse 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 *(on page 192)*.

Note: The built-in 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 Eclipse 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.

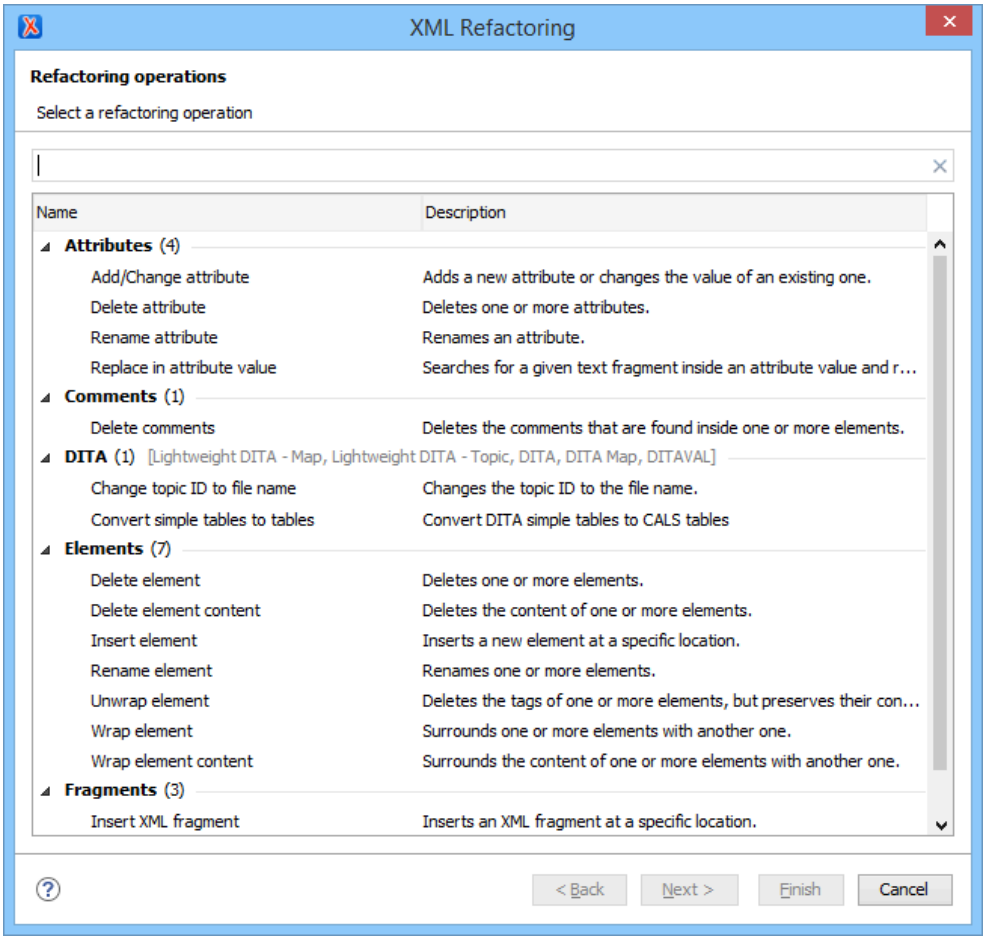
XML Refactoring Wizard

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.

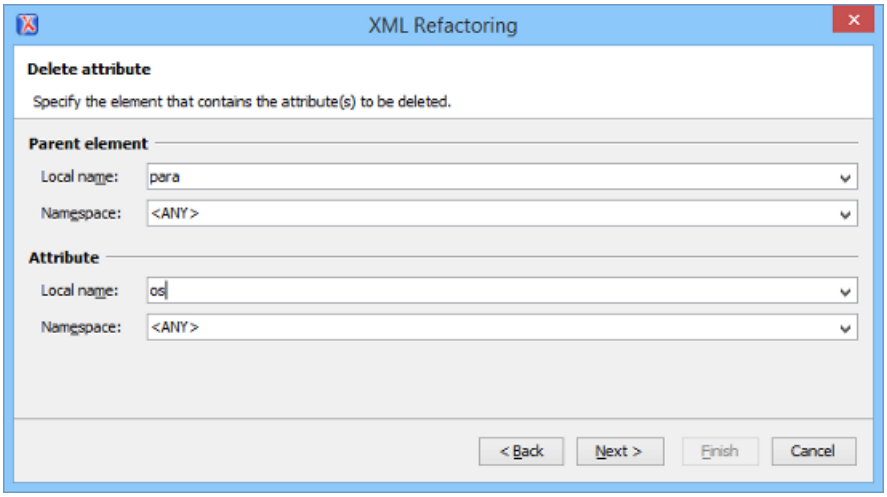
Figure 93. XML Refactoring Wizard



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.

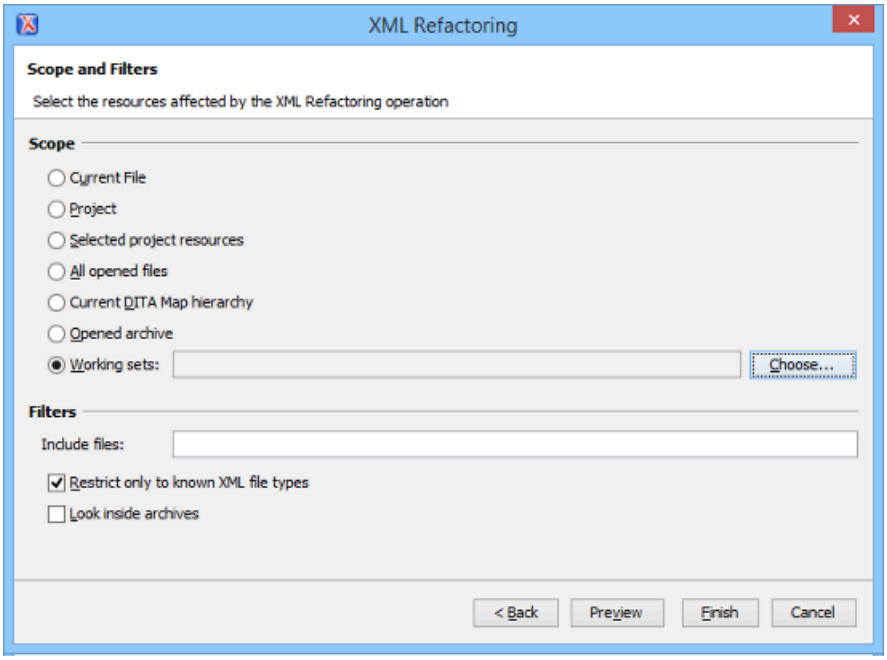
Figure 94. XML Refactoring 2nd Wizard Page (Delete Attribute Operation)



Scope and Filters

The last wizard page allows you to select the set of files that represent the input of the operation.

Figure 95. XML Refactoring - Scope and Filters Wizard Page



Scope section

In the **Scope** section, you can select from predefined resource sets (such as the current file, your whole project, the current *DITA map (on page 1324)* hierarchy for DITA projects, etc.) or you can define your own set of resources by creating a *working set (on page 1329)*.

Filters

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 selected, only resources with a known XML file type will be affected by the operation.

Preview

You can use the **Preview** button to open a comparison panel where you can review all the changes that will be made by the refactoring operation before applying the changes.

Finish

After clicking the **Finish** button, the operation will be processed and Oxygen XML Developer Eclipse plugin provides no automatic means for reverting the operations. Any **Undo** action will only revert changes on the current document.

Troubleshooting: If an operation fails, a notification will be displayed in the **Results** panel with some information about the error. For example, if the operation was invoked on a read-only resource, the error will indicate that a read-only file cannot be converted.

Tip: If an operation takes longer than expected you can use the **Stop** button in the progress bar to cancel the operation.

Restriction: XML refactoring operations cannot preserve CDATA sections. If your document contains XML CDATA sections, the refactoring operations will convert them to plain text nodes.

Built-in Refactoring Operations

The XML Refactoring tool includes a variety of built-in 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 Eclipse 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 built-in 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

Adds the attribute to all instances of the specified parent element.

Change the value in the parent elements where the attribute already exists

Replaces the value of the already existing attribute in all instance of the specified parent element.

Both

Adds the attributes to the instances where it is missing and replaces the value in instances where the attribute already exists.

Convert attribute to element

Use this operation to convert a specified attribute to an element. This operation allows you to specify the following parameters:

Parent element section

Element

The parent element of the attribute to be converted, 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.

New element section

Local name

The local name of the new element.

Namespace

The namespace of the new element.

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) that will have comments 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.

Convert CALS tables to simple tables

Use this operation to convert DITA CALS tables to simple tables.

Convert conrefs to conkeyrefs

Use this operation to convert `@conref` attributes to `@conkeyref` attributes.

Convert simple tables to CALS tables

Use this operation to convert DITA simple tables to CALS tables.

Convert Nested Topics to New Topics

Use this operation on topics that contain nested `<topic>` elements to convert each nested topic to a new topic.

Convert Sections to New Topics

Use this operation on topics that contain multiple sections to convert each section to a new topic.

Convert to Concept

Use this operation to convert a DITA topic (of any type) to a DITA Concept topic type (for example, Topic to Concept).

Convert to General Task

Use this operation to convert a DITA topic (of any type) to a DITA General Task topic type (for example, Task to General Task).

Convert to Reference

Use this operation to convert a DITA topic (of any type) to a DITA Reference topic type (for example, Topic to Reference).

Convert to Task

Use this operation to convert a DITA topic (of any type) to a DITA Task topic type (for example, Topic to Task).

Convert to Topic

Use this operation to convert a DITA topic (of any type) to a DITA Topic (for example, Task to Topic).

Convert to Troubleshooting

Use this operation to convert a DITA topic (of any type) to a DITA Troubleshooting topic type (for example, Topic to Troubleshooting).

Rename Key

Use this operation to rename a key. It also updates all references to it. Note that it does not work on DITA 1.3 key scopes.

All of these DITA refactoring actions allow you to choose a scope for the operation and some filters:

Scope

Select from a variety of options to define the scope that will have resources affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, **Current DITA map hierarchy**, 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.

Add Blue DOCTYPE - NISO JATS Publishing 1.1

Use this operation to add a JATS 'Blue' 1.1 DOCTYPE declaration.

Normalize IDs

Use this operation to normalize assigned IDs and assigned IDs to elements that are missing them.

All of these JATSKit refactoring actions allow you to choose a scope for the operation and some filters:

Scope

Select from a variety of options to define the scope for the resources that 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 *Publishing Template*

These operations are for those who use *Oxygen Publishing Templates* for WebHelp Responsive output customization.

Migrate HTML Page Layout Files to v21

Use this operation to convert custom HTML page layout files *(on page 830)* that are included in a custom Publishing Template that was created in Oxygen XML Developer Eclipse plugin version 20.0 or 20.1 so that they will be compatible with Oxygen XML Developer Eclipse plugin version 21.0.

Update HTML Pages

⚠ **Attention:** This operation is only used by Oxygen XML Developer Eclipse plugin and should not be used manually.

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** *(on page 133)* 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

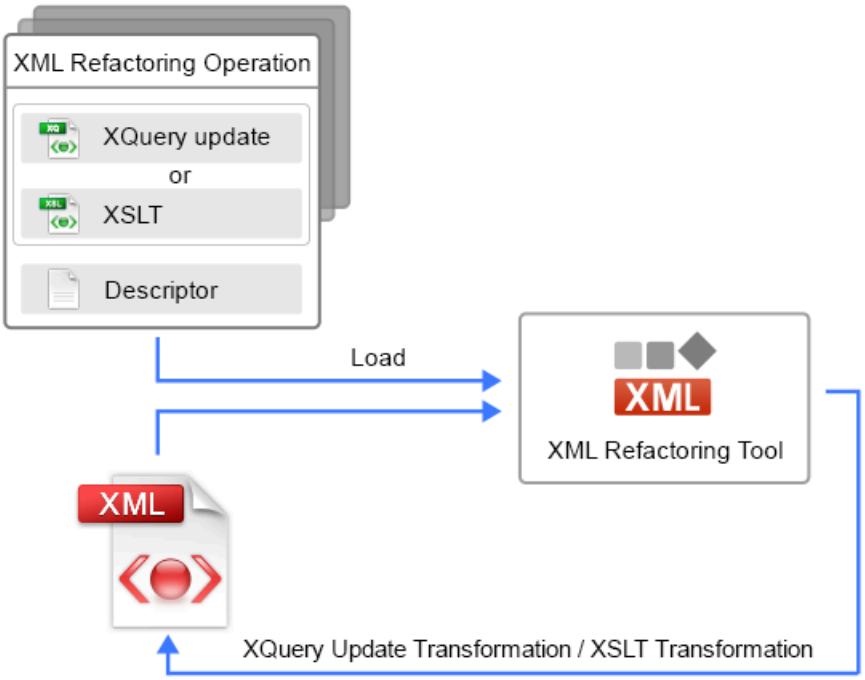
While Oxygen XML Developer Eclipse plugin includes a variety of built-in XML refactoring operations to help you accomplish particular tasks, you can also create custom operations according to your specific needs. For example, you could create a custom refactoring operation to convert an attribute to 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 Eclipse 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 96. 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 *(on page 326)*, along with the built-in operations.

After the user chooses an operation and specifies its parameters, Oxygen XML Developer Eclipse 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.

    Note: This can be changed using Saxon extension functions in XSLT *(on page 352)*.
  - 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 applied on the root element and the nodes inside it. However, as a work around to this limitation, you can use Saxon extension functions and the XSLT stylesheet method *(on page 352)* to implement the new custom XML refactoring operation.

Creating a Custom Refactoring Operation

To create a custom refactoring operation, follow these steps:

1. Create an XQuery Update script *(on page 344)* or XSLT stylesheet file *(on page 348)*.
2. Create an **XML Refactoring Operation Descriptor** file contains the path to the XQuery Update script *(on page 346)* or XSLT stylesheet *(on page 350)*.
3. Store both files in one of the locations that Oxygen XML Developer Eclipse plugin *(on page 354)* 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 *(on page 326)*.

Related Information:

- Storing and Sharing Refactoring Operations *(on page 354)*

Custom Refactoring Script

The first step in creating a custom refactoring operation is to create an XQuery Update script *(on page 344)* or XSLT stylesheet *(on page 348)* 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 the XQuery method example *(on page 344)* or XSLT method example *(on page 348)* to help you with the content.

There are cases when it is necessary to add parameters in the XQuery script *(on page 344)* or XSLT stylesheet *(on page 348)*. 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 Eclipse plugin. Thus, if you want to create or edit an XQuery Update script you have to enable this mechanism by creating an

XQuery transformation scenario *(on page 754)* and choose **Saxon EE** as the transformation engine. Also, you need to make sure the **Enable XQuery update** option is selected in the Saxon processor advanced options *(on page 711)*.

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 *(on page 133)*.

The next step in creating a custom refactoring operation is to create an **XML Refactoring Operation Descriptor** file contains the path to the XQuery Update script *(on page 346)* or XSLT stylesheet *(on page 350)*.

Related Information:

XQuery Update Script for Creating a Custom Operation *(on page 344)*

XSLT Stylesheet for Creating a Custom Operation *(on page 348)*

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 *(on page 344)* or XSLT stylesheet *(on page 348)* 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 *(on page 327)* 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 *(on page 327)*.

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 Eclipse 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** *(on page 133)* 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 Eclipse 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 (on page 1328)* 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 Eclipse 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 Eclipse 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" useCurrentContext="true">
    <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>
```

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" useCurrentContext="true">
    <localName label="Name" name="attribute_localName">
        <description>The name of the attribute to be converted.</description>
    </localName>
    <namespace label="Namespace" name="attribute_namespace" allowsAny="true">
        <description>Namespace of the attribute to be converted.</description>
    </namespace>
</attributeParameter>
```

Note: All built-in operations are loaded from the *`[OXYGEN_INSTALL_DIR]/refactoring`* folder.

Related Information:

Example of an Operation Descriptor File with an XSLT Stylesheet *(on page 350)*

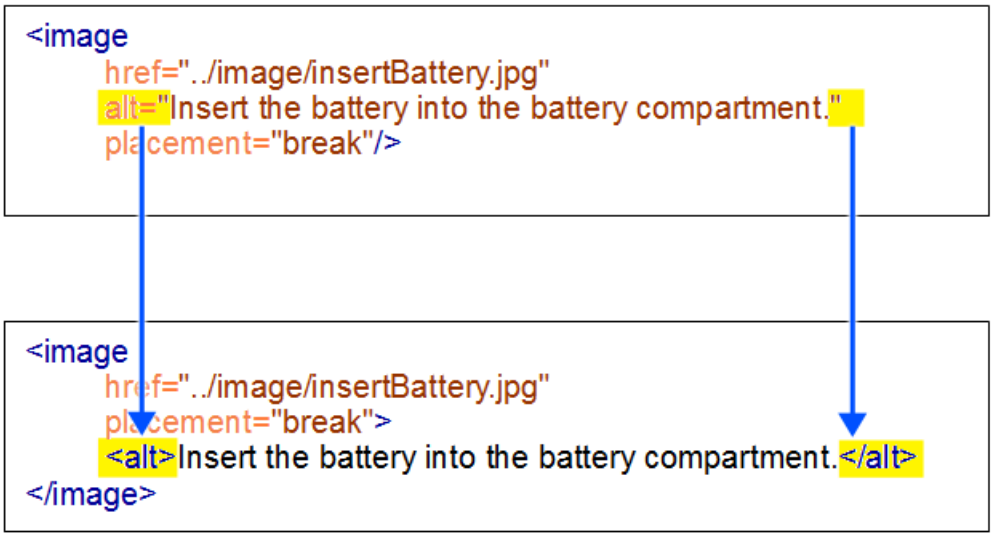
Example of an Operation Descriptor File with an XQuery Update script *(on page 346)*

XQuery Update Script for Creating a Custom Operation

To demonstrate creating a custom operation, suppose that you have a task where you 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, the 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 97. Example: Custom XML Refactoring Operation



An XQuery Update script can be used to implement the new custom XML refactoring operation. The second requirement is an XML Refactoring operation descriptor file *(on page 346)* that contains the path to the XQuery Update script.

Restriction: There is a limitation to using an XQuery script in that *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. However, as a work around to this limitation, you can use Saxon extension functions and the XSLT stylesheet method *(on page 352)* to implement the new custom XML refactoring operation.

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 (on page 1328)* of the new element to be inserted and inserts it as the first child of the parent element.

```
(:
    XQuery document used to implement 'Convert attribute to element'
      operation from XML Refactoring tool.
:)
declare namespace output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method   "xml";
declare option output:indent   "no";
(: Local name of the attribute's parent element. :)
declare variable $element_localName as xs:string external;
(: Namespace of the attribute's parent element. :)
declare variable $element_namespace as xs:string external;
(: The local name of the attribute to be converted :)
declare variable $attribute_localName as xs:string external;
(: The namespace of the attribute to be converted :)
declare variable $attribute_namespace as xs:string external;
(: Local name of the new element. :)
declare variable $new_element_localName as xs:string external;
(: Namespace of the new element. :)
declare variable $new_element_namespace as xs:string external;
(: Convert attribute to element:)
for $node in //*
(: Find the attribute to convert :)
let $attribute :=
    $node/@*[local-name() = $attribute_localName and
    ($attribute_namespace = '<ANY>' or $attribute_namespace = namespace-uri())]

(: Compute the prefix for the new element to insert :)
let $prefix :=
    for $p in in-scope-prefixes($node)
      where $new_element_namespace = namespace-uri-for-prefix($p, $node)
return $p
(: Compute the qname for the new element to insert :)
let $new_element_qName :=
    if (empty($prefix) or $prefix[1] = '') then $new_element_localName
```

```
   else $prefix[1] || ':' || $new_element_localName

where ('<ANY>' = $element_localName or local-name($node) = $element_localName)
  and
($element_namespace = '<ANY>' or $element_namespace = namespace-uri($node))

 return
   if (exists($attribute)) then
     (insert node element {QName($new_element_namespace, $new_element_qName)}
     {string($attribute)} as first into $node,
     delete node $attribute)
     else ()
```

Example of an Operation Descriptor File That References the XQuery Script for Creating a Custom Operation to *Convert an Attribute to an Element*

After you have developed the XQuery script, 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>
```

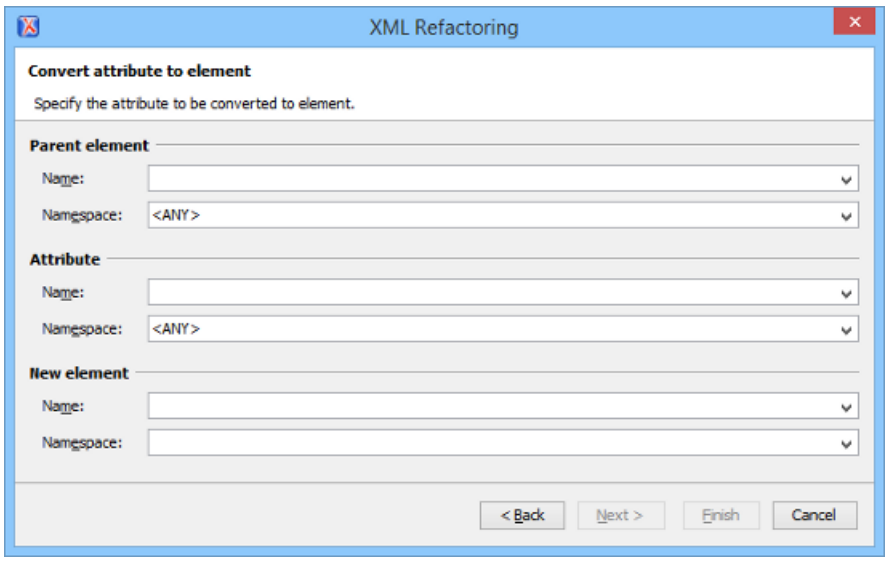
```
    <section label="Attribute">
     <attributeParameter dependsOn="elemID">
      <localName label="Name" name="attribute_localName">
       <description>Name of the attribute to be converted.</description>
      </localName>
     <namespace label="Namespace" name="attribute_namespace" allowsAny="true">
       <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>
```

Results

After you have created these files, copy them into a folder scanned by Oxygen XML Developer Eclipse plugin when it loads the custom operation *(on page 354)*. 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 the example of the custom operation to convert an attribute to an element:

Figure 98. Example: XML Refactoring Wizard for a Custom Operation

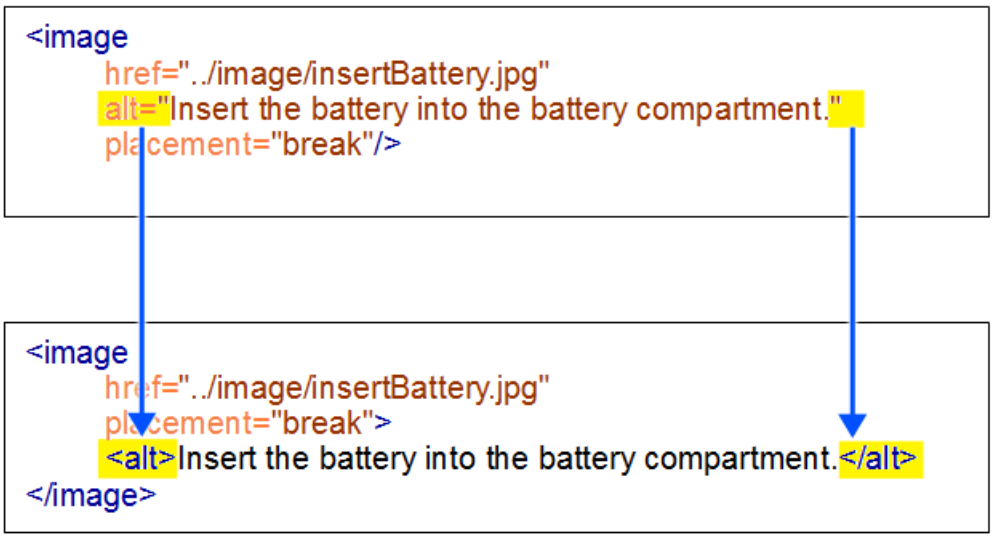


XSLT Stylesheet for Creating a Custom Operation

To demonstrate creating a custom operation, suppose that you have a task where you 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, the 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 99. Example: Custom XML Refactoring Operation



An XSLT stylesheet can be used to implement the new custom XML refactoring operation. The second requirement is an XML Refactoring operation descriptor file *(on page 350)* that contains the path to the XSLT stylesheet.

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 (on page 1329)* set in the XML Refactoring *framework (on page 1325)*.

Example of an Operation Descriptor File That References the XSLT Stylesheet for Creating a Custom Operation to *Convert an Attribute to an Element*

After you have developed the 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"/>
```

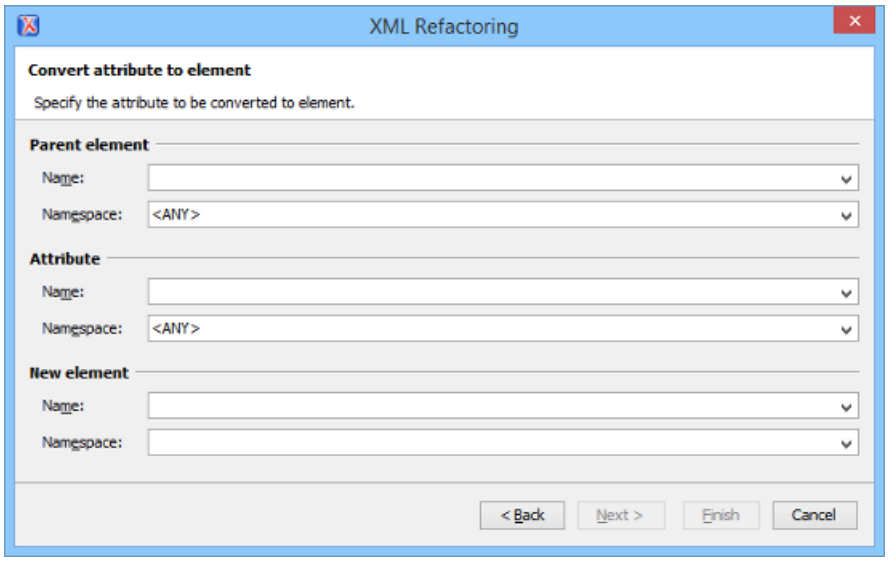
The code exemplified above and other refactoring examples can be found on the DITA Refactoring GitHub sample project.

Results

After you have created these files, copy them into a folder scanned by Oxygen XML Developer Eclipse plugin when it loads the custom operation *(on page 354)*. 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 the example of the custom operation to convert an attribute to an element:

Figure 100. Example: XML Refactoring Wizard for a Custom Operation



Using Saxon Extension Functions to Allow Custom Refactoring Operations to Read and Modify Content Outside the Root Node

One advantage to using an XSLT stylesheet is that there is limitation when using an XQuery Update script *(on page 344)* in that refactoring operations can only be performed on *comments* or *processing instructions* that are inside the root element. Thus, using the XQuery method, *comments* or *processing instructions* that are in any node before or after the root element cannot be modified by an XML Refactoring operation.

The XSLT stylesheet method offers a work-around to this limitation through the use of some Saxon extension functions.

To illustrate the use of these functions, consider the following sample XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- comment before root -->
<?pi before root ?>
<root>
```

```
    <child></child>
</root>
<!-- comment after root -->
<?pi after root ?>
```

The following Saxon extension functions can be used to read and modify content outside the root node:

Note: They belong to the `http://www.oxygenxml.com/ns/xmlRefactoring/functions` namespace.

- **get-content-after-root()** - Returns the content after root as `xs:string`.

  For the XML above, the call of this function will return the following string value:

  ```
  <!-- comment after root -->
  <?pi after root ?>
  ```

- **set-content-after-root(xs:string)** - Updates the content that will be serialized in the refactored document after the root node.

  The function call `set-content-after-root('<!-- Inserted comment -->')` will result in replacing the nodes after the root element with the comment passed as string argument. The XML document will be modified as follows:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <!-- comment before root -->
  <?pi before root ?>
  <root>
      <child></child>
  </root><!-- Inserted comment -->
  ```

- **get-content-before-root()** - Returns the content before root as `xs:string`.

  For the XML above, the call of this function will return the following string value:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <!-- comment before root -->
  <?pi before root ?>
  ```

- **set-content-before-root(xs:string)** - Updates the content that will be serialized in the refactored document after the root node.

  The function call `set-content-before-root('<!-- Inserted comment -->')` will result in replacing the nodes before the root element with the comment passed as string argument. The XML document will be modified as follows:

  ```
  <!-- Inserted comment --><root>
      <child></child>
  </root>
  <!-- comment after root -->
  <?pi after root ?>
  ```

XSLT Example:

To process content after the root node, the XSLT would look like this:

```
<?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:xrf="http://www.oxygenxml.com/ns/xmlRefactoring/functions" version="3.0">
    <xsl:template match="/">
        <!-- The comment content that will be inserted after the root element -->
        <xsl:variable name="commentAsText"><!-- COMMENT ADDED FROM XR OPERATION-->
        </xsl:variable>
        <!-- Retrieve the content after the root element as is -->
        <xsl:variable name="after-root-content" as="xs:string"
                      select="xrf:get-content-after-root()"/>
        <xsl:variable name="processedContent"
                      select="concat($after-root-content, $commentAsText)"/>

        <!-- Update the content after the root element -->
        <xsl:value-of select="xrf:set-content-after-root($processedContent)"/>
        <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="node() | @*">
        <xsl:copy>
            <xsl:apply-templates select="node() | @*"/>
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

Note: The above XSLT retrieves the nodes after the root element as string, appends a new comment, and then sets back the updated content into the XML document.

Storing and Sharing Refactoring Operations

Oxygen XML Developer Eclipse 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. To associate the parent directory to the *framework*, you need to add it in the **Classpath** tab of the **Document type** configuration dialog box *(on page 55)*.
- A folder that you specify in the **Load additional refactoring operations from** text box *(on page 127)* in the **XML Refactoring** preferences page *(on page 127)*.
- The `refactoring` folder from the Oxygen XML Developer Eclipse plugin installation directory (*`[OXYGEN_INSTALL_DIR]/refactoring/`*).

Sharing Custom Refactoring Operations

The purpose of Oxygen XML Developer Eclipse 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 *framework (on page 1325)* or projects.

After storing custom operations, you can share them with other users by sharing the resources.

Localizing XML Refactoring Operations

Oxygen XML Developer Eclipse 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 `possibleValues` for each `parameter`.

Translated refactoring information uses the following form:

```
${i18n(translation_key)}
```

Oxygen XML Developer Eclipse 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 *(on page 36)*, 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 Eclipse plugin installation directory (*[OXYGEN_INSTALL_DIR]*`/refactoring/i18n`).

Example: 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>
```

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 is 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 (on page 1323)* 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.

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 (on page 1323)* document. The *canonical (on page 1323)* form of an XML document is physical representation of the document produced by the method described in this specification. The *XML canonicalization (on page 1323)* method is the algorithm defined by this specification that generates the canonical form of a given 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 (on page 1323)* form of an XML document or document subset allows the signature digest calculations to be oblivious to changes in the original document's physical representation. During signature generation, the digest is computed over the *canonical* form of the document. The document is then transferred to the relying party, which validates the signature by reading the document and computing a digest of the *canonical* form of the received document. The equivalence of the digests computed by the signing and relying parties (hence, the equivalence of the *canonical* forms that they were computed for) 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 Eclipse plugin:

- **Canonical XML (or Inclusive XML Canonicalization)** (XMLC14N) - Used for XML where the context doesn't 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 standpoint 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 XML Canonicalization** (EXCC14N) - Designed for *canonicalization* where the context might change.

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 (on page 1323)* 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 the delimiter for comments outside the document element.

The three operations. **Canonicalize** *(on page 359)*, **Sign** *(on page 360)*, and **Verify Signature** *(on page 362)*, are available from the **Source** submenu when invoking the contextual menu in **Text** mode or from the **XML Tools** menu.

Related Information:

Certificates *(on page 358)*

Canonicalizing Files *(on page 359)*

Signing Files *(on page 360)*

Verifying Signature *(on page 362)*

Example of How to Digitally Sign XML Files or Content *(on page 362)*

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 (on page 1326)*.

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*.

Oxygen XML Developer Eclipse plugin provides 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 *(on page 36)* and go to **XML > XML Signing Certificates**. This opens the certificates preferences page *(on page 127)*.

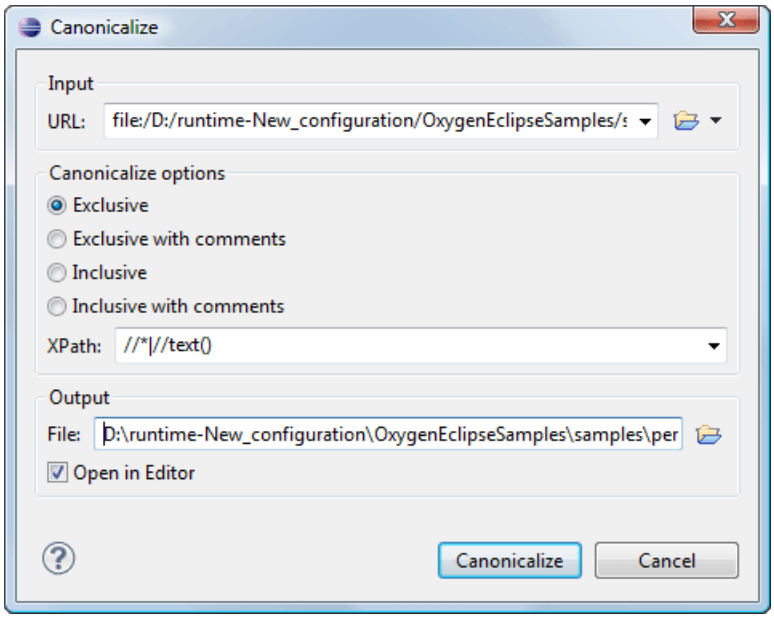
Related Information:

Digital Signatures Overview *(on page 356)*

Canonicalizing Files

You can select the *canonicalization (on page 1323)* 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 101. 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 (on page 1323)* 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 (on page 1323)* method is used.
- **Inclusive** - If selected, the inclusive (uncommented) *canonicalization (on page 1323)* 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 standpoint 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 (on page 1323)* 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 selected, the output file will be opened in the editor.

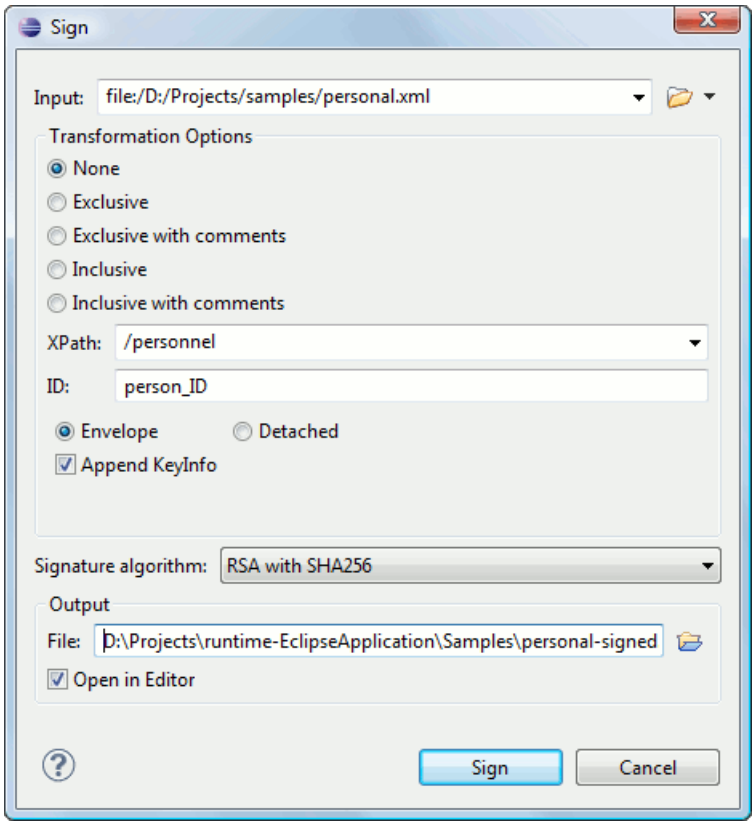
Related Information:

Digital Signatures Overview *(on page 356)*

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 102. Signature Settings Dialog Box



The following options are available:

Note: If Oxygen XML Developer Eclipse 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 *(on page 127)* where you can configure a valid certificate.

- **Input** - Available if the **Sign** action was selected from the **XML Tools** menu. Specifies the location of the input URL.
- **Transformation Options** - See the Digital Signature Overview *(on page 356)* section for more information about these options.
  - **None** - If selected, no *canonicalization (on page 1323)* algorithm is used.
  - **Exclusive** - If selected, the exclusive (uncommented) *canonicalization (on page 1323)* 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 (on page 1323)* method is used.
  - **Inclusive** - If selected, the inclusive (uncommented) *canonicalization (on page 1323)* 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 standpoint 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 (on page 1323)* 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 *(on page 356)* for more information.
- **Detached** - If selected, the *detached* signature is used. See the Digital Signature Overview *(on page 356)* for more information.
- **Append KeyInfo** - If this option is selected, 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 selected, the output file will be opened in Oxygen XML Developer Eclipse plugin.

Related Information:

Digital Signatures Overview *(on page 356)*

Verifying Signature *(on page 362)*

Example of How to Digitally Sign XML Files or Content *(on page 362)*

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 Information:

Digital Signatures Overview *(on page 356)*

Signing Files *(on page 360)*

Example of How to Digitally Sign XML Files or Content *(on page 362)*

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 Eclipse plugin includes a signature tool that allows you to digitally sign XML documents or specific content.

The Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse 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 *(on page 127)* 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** *(on page 361)*. 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 *(on page 361)* 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 356)*

Signing Files *(on page 360)*

Verifying Signature *(on page 362)*

Editing XSLT Stylesheets

Oxygen XML Developer Eclipse 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 *(on page 171)*.
- **Open and Edit XSLT files** - XSLT files can be opened and edited in the source editor (**Text** mode *(on page 222)*).
- **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 Eclipse plugin makes XSLT files more readable.

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 Eclipse plugin provides the support for defining the main module (or modules), allowing you to edit any of the imported/ included files in the context of the larger stylesheet structure.

You cat set a main XSLT stylesheet either using the *master files* support from the **Navigator** view *(on page 200)*, or using a validation scenario.

To set a *master file* using a validation scenario, add validation units that point to the main modules. Oxygen XML Developer Eclipse 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 *master file (on page 1326)* include:

- Correct validation of a module in the context of a larger stylesheet structure.
- *Content Completion Assistant (on page 1324)* displays all components valid in the current context.
- The **Outline** view *(on page 376)* displays the components collected from the entire stylesheet structure.

For more information about editing XSLT stylesheets in the *master files* context, watch our video demonstration:

https://www.youtube.com/embed/UZwg385RKNw

Related Information:

XSLT Resource Hierarchy/Dependencies View *(on page 382)*

XSLT Component Dependencies View *(on page 384)*

Validating XSLT Stylesheets

Numerous XSLT code quality assurance checks are done during automatic validation to help you keep your stylesheets valid and well formed. Oxygen XML Developer Eclipse plugin performs the validation of XSLT documents with the help of an XSLT processor that you can configure in the preferences pages *(on page 137)* according to the XSLT version.

For XSLT 1.0, the options are: Xalan, Saxon 6.5.5, Saxon 9.9.1.5 and a JAXP transformer specified by the main Java class. *(on page 137)* For XSLT 2.0, the options are: Saxon 9.9.1.5 and a JAXP transformer specified by the main Java class. *(on page 137)* For XSLT 3.0, the options are Saxon 9.9.1.5 and a JAXP transformer specified by the main Java class. *(on page 137)*

Creating a Validation Scenario for XSLT Stylesheets

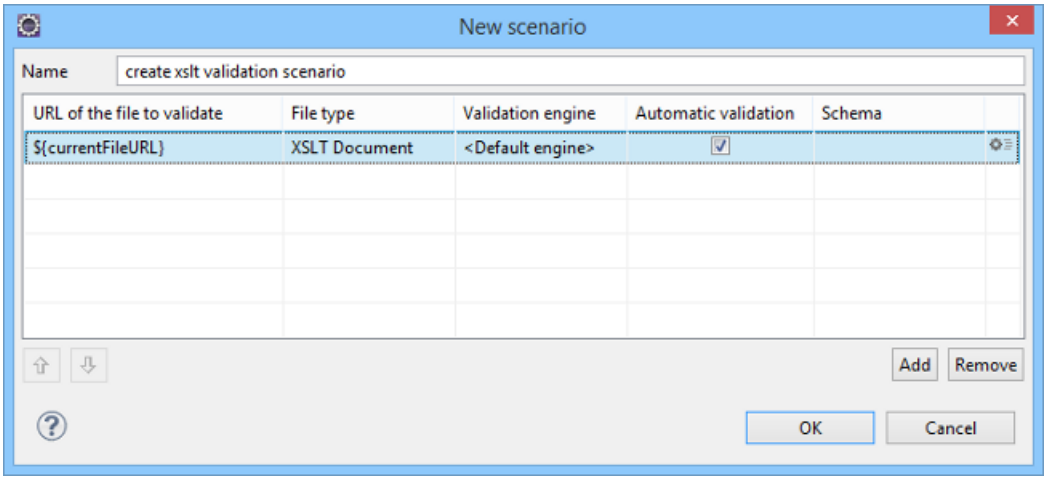
You can validate an XSLT document using the engine defined in the transformation scenario, or a custom validation scenario. If you choose to validate using the engine from transformation scenario, and a

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 *(on page 36)* and go to **XML > XSLT/FO/XQuery > XSLT**.

You can also create new validation scenarios or edit existing ones, and you can add *JARS (on page 1326)* 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 open in Oxygen XML Developer Eclipse 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 *(on page 192)*.
   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 click the **Edit** button. If you try to edit one of the *read-only* built-in scenarios, Oxygen XML Developer Eclipse plugin creates a customizable duplicate (you can also use the **Duplicate** button).
3. To add a new scenario, click the **New** button.
   The **New scenarios** dialog box is displayed. It lists all validation units of the scenario.

Figure 103. 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** *(on page 148)* 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 selected, 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 click the **Move up** or **Move down** buttons.
6. Click **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 click 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 Eclipse plugin preferences. After a custom validation engine is properly configured *(on page 82)*, 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 Eclipse plugin (Windows edition). It is associated to the XSL Editor type in Preferences page. *(on page 82)*
- **MSXML.NET** - included in Oxygen XML Developer Eclipse plugin (Windows edition). It is associated to the XSL Editor type in Preferences page. *(on page 82)*

Validating XSLT Stylesheets that Call Java Extensions

It is possible to validate an XSLT that calls Java extensions. This is achieved through a transformation scenario where the Java extensions are specified, and the default validation will be processed using the parameters defined in the transformation scenario.

To validate XSLT with Java extensions, follow this procedure:

1. Create an XSLT transformation scenario *(on page 732)* for your XSLT document (select **Configure Transformation Scenario(s)** action from the toolbar, then click **New**, and select **XSLT transformation**).
2. In the **New scenario** dialog box, click the **Extensions** button (in the **XSLT** tab), specify the Java extensions (JAR libraries) that are needed, and click **OK**.
3. Once you are finished configuring the transformation scenario, click **OK**, then select **Save and close**.
4. Use the **Validate** button on the toolbar **(or Alt + Shift + V, V)** and the default validation will detect and use the transformation scenario profile you just configured and saved.

Related Information:

Debugging XSLT that Call Java Extensions *(on page 1235)*

XSLT Quick Fix Support

The Oxygen XML Developer Eclipse plugin *Quick Fix* support *(on page 1328)* 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 Eclipse 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 104. Example of an Undefined XSLT Functions Quick Fix

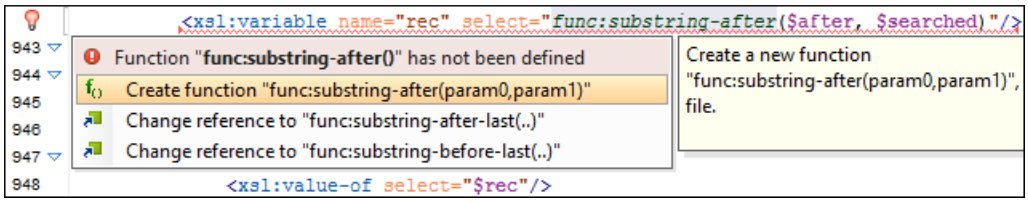
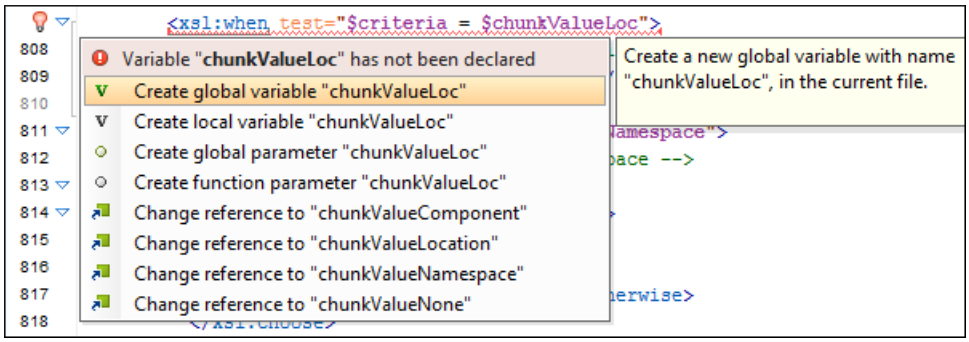


Figure 105. Example of an Undeclared XSLT Variables/Parameters Quick Fix



Oxygen XML Developer Eclipse 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 **Add parameter "paramName" in call-template** quick-fix is available. It 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.

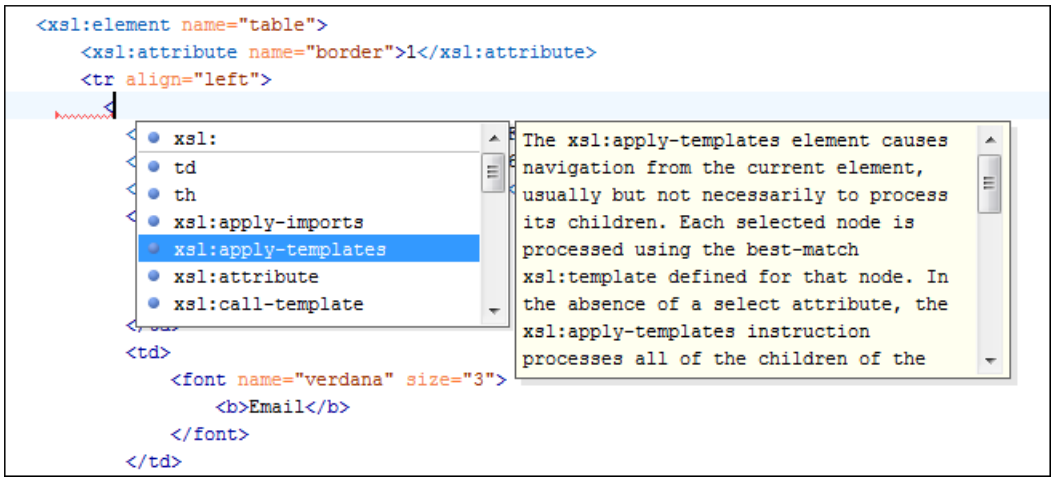
- **Character-map "chacterMap" has not been defined**, when the referenced character map declaration is not found. The following *Quick Fixes* are available:
  - **Create character-map "characterMapName"** - creates a new character map with the specified name, after the current top-level element from stylesheet.
  - **Change reference to "characterMapName"** - changes the referenced character map to an already defined one.

Content Completion in XSLT Stylesheets

The list of proposals offered by the *Content Completion Assistant (on page 1324)* in XSLT are context-sensitive and includes proposals that are valid at the current cursor position. It can be manually activated with the **Ctrl + Space (Command + Space on OS X)** shortcut.

You can enhance the list of proposals by specifying an additional schema. This schema is defined in the **Content Completion / XSLT** preferences *(on page 81)* page and can be any of the following: XML Schema, DTD, RELAX NG schema, or NVDL schema.

Figure 106. XSLT Content Completion Assistant



The feature is activated in **Text** mode in the following situations:

- After you enter 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 *(on page 79)* from the **Content Completion** preferences page.
- After typing a partial element or attribute name, you can manually 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 *(on page 238)* into stylesheets.

Note: For XSL and XSD resources, the *Content Completion Assistant* collects its components starting from the *master files (on page 1326)*. 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 *(on page 200)*.

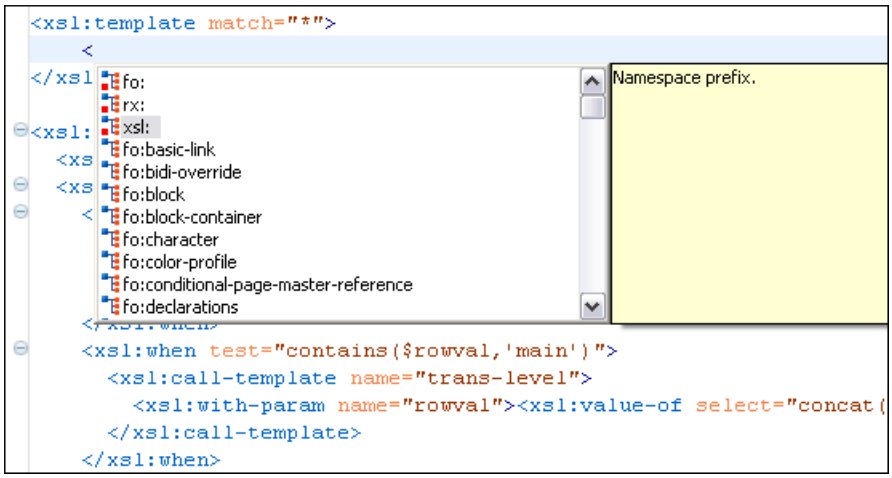
The extension functions included in the Saxon 6.5.5 and 9.9.1.5 transformation engines are presented in the content completion list only if the Saxon namespace (*http://saxon.sf.net* for XSLT version 2.0 / 3.0 or *http://icl.com/saxon* for XSLT version 1.0) is declared and one of the following conditions is true:

- The edited file has a transformation scenario that uses as transformation engine Saxon 6.5.5 (for XSLT version 1.0), Saxon 9.9.1.5 PE or Saxon 9.9.1.5 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 9.9.1.5 PE or Saxon 9.9.1.5 EE (for version 2.0 / 3.0).
- The validation engine specified in Options *(on page 137)* page is Saxon 6.5.5 (for version 1.0), Saxon 9.9.1.5 PE or Saxon 9.9.1.5 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 107. 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 Eclipse 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 (on page 1324)* provides all the features available in the XML editor *(on page 234)* and also adds some enhancements. In XPath expressions used in attributes of XSLT stylesheets (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 *(on page 380)*.

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 subordinates="one.worker"/>
    </person>
    <person id="one.worker">
        <name>
            <family>Worker</family>
```

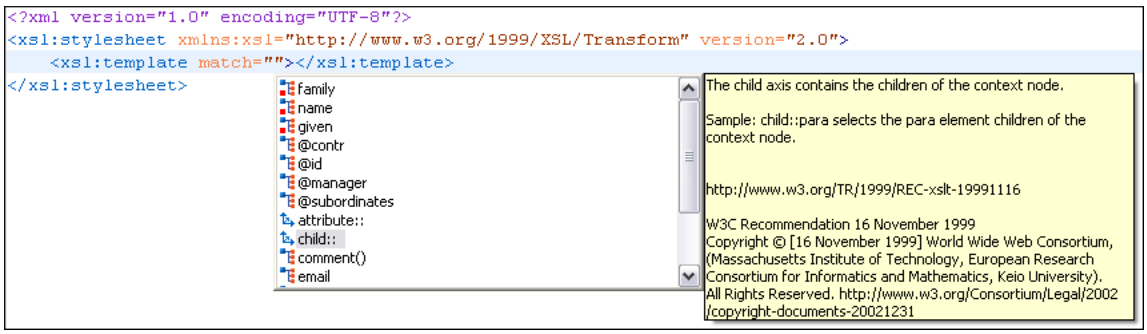
```
            <given>One</given>
        </name>
        <email>one@oxygenxml.com</email>
        <link manager="Big.Boss"/>
    </person>
</personnel>
```

If you enter an `<xsl:template>` element using the *Content Completion Assistant*, the following actions are triggered:

- The `@match` attribute is inserted automatically.
- The cursor is placed between the quotes.
- The XPath *Content Completion Assistant* automatically displays a 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.

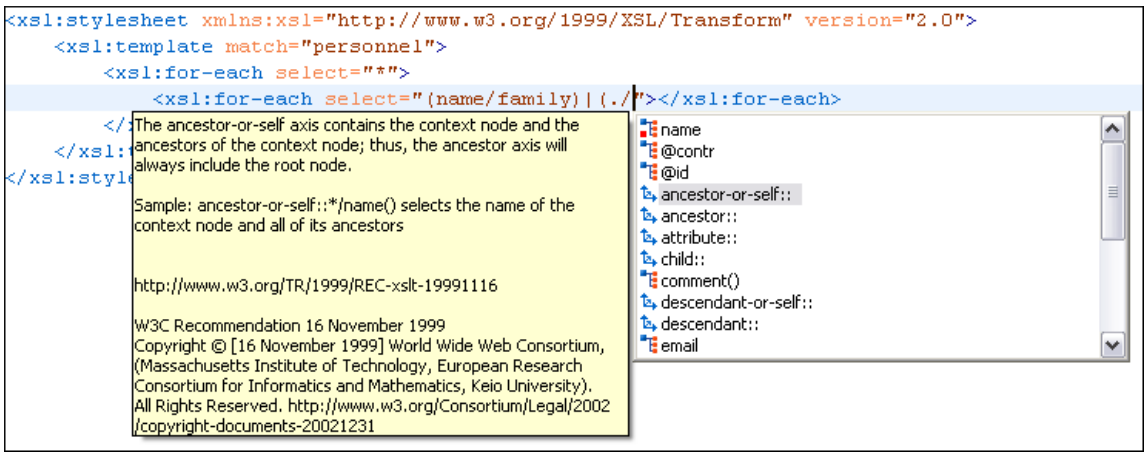
The set of XPath functions depends on the XSLT version declared in the root element `xsl:stylesheet`: 1.0, 2.0, or 3.0. Functions from other namespaces, such as maps, arrays, and math, are presented only if the namespaces are declared.

Figure 108. Content Completion in the `@match` Attribute



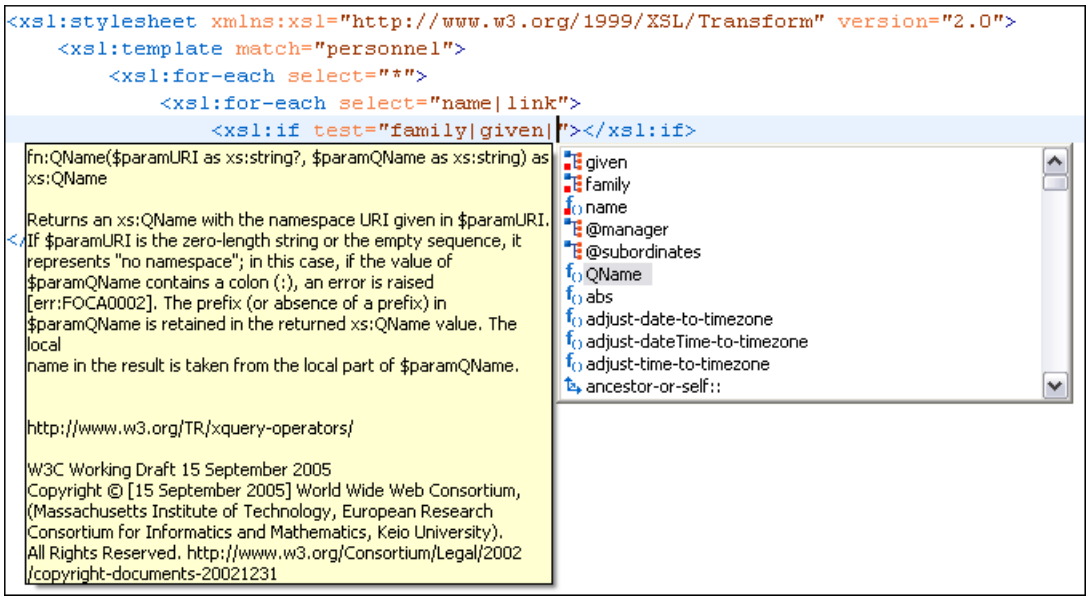
If the cursor is inside the `@select` attribute of an `<xsl:for-each>`, `<xsl:apply-templates>`, `<xsl:value-of>` or `<xsl:copy-of>` element the content completion proposals depend on the path obtained by concatenating the XPath expressions of the parent XSLT elements `<xsl:template>` and `<xsl:for-each>` as shown in the following figure:

Figure 109. Content Completion in the `@select` Attribute



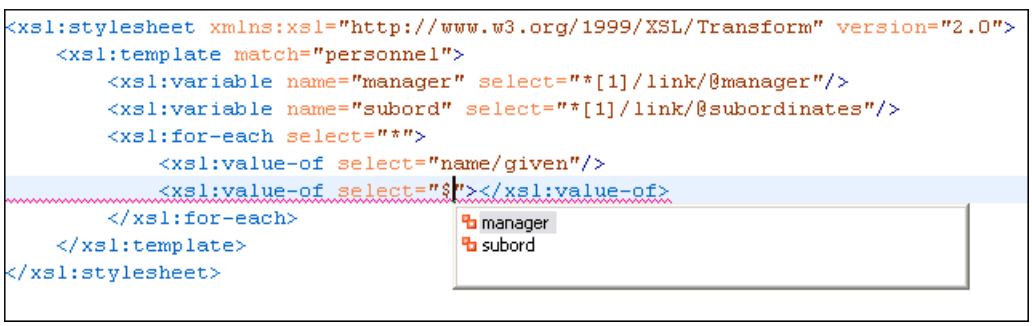
Also XPath expressions typed in the `@test` attribute of an `<xsl:if>` or `<xsl:when>` element benefit of the assistance of the content completion.

Figure 110. Content Completion in the `@test` Attribute



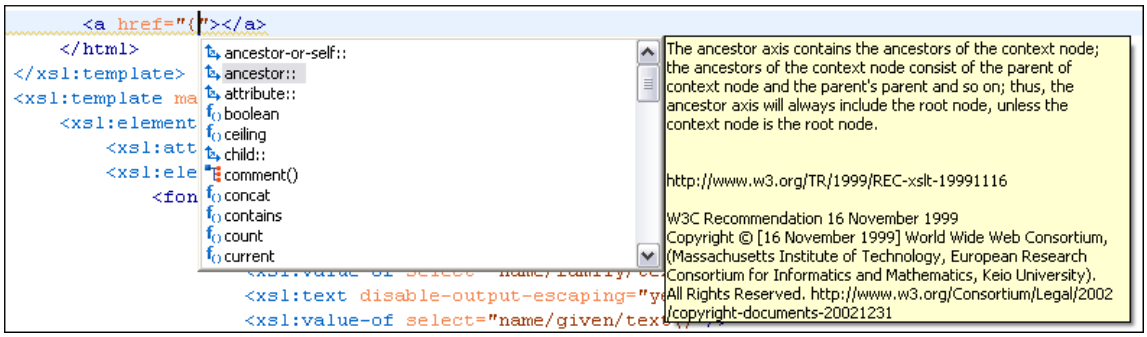
XSLT variable references are easier to insert in XPath expressions with the help of the content completion pop-up triggered by the `$` character, which signals the start of such a reference in an XPath expression.

Figure 111. 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 112. Content Completion in Attribute Value Templates



The time delay (configured in the **Content Completion** preferences page *(on page 79)*) is also applied for the content completion in XPath expressions.

Related Information:

Working with XPath Expressions *(on page 1119)*

Tooltip Helper for the XPath Functions Arguments

When editing the arguments of an XPath/XSLT function, Oxygen XML Developer Eclipse 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.

Examples:

If you want to concatenate the absolute values of two variables, named *v1* and *v2*:

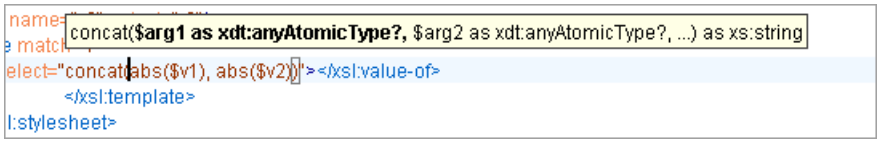
```
<xsl:template match="/">
    <xsl:value-of select="concat(abs($v1), abs($v2))"></xsl:value-of>
</xsl:template>
```

When moving the cursor before the first `abs` function, Oxygen XML Developer Eclipse 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).

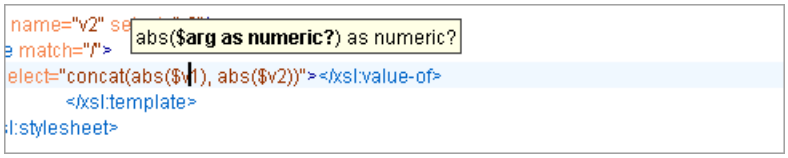
The function also takes other arguments that have the same type and returns a `xs:string`.

Figure 113. XPath Tooltip Helper - Identify the `concat` Function's First Argument



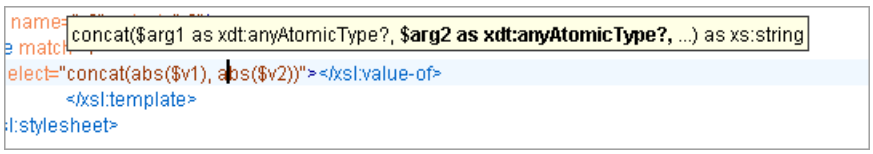
Moving the cursor on the first variable `$v1`, the editor identifies the `abs` as context function and shows its signature:

Figure 114. XPath Tooltip Helper - Identify the `abs` Function's Argument



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

Figure 115. XPath Tooltip Helper - Identify the `concat` Function's Second Argument



Note: The tooltip helper is also available in the **XPath Builder** view *(on page 1119)*.

Related Information:

Working with XPath Expressions *(on page 1119)*

Syntax Highlighting in XSLT

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for XSLT files, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
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.

Tip: Oxygen XML Developer Eclipse plugin also allows you to specify syntax highlighting colors for specific XML elements and attributes with specific namespace prefixes. This can be done in the **Editor > Syntax Highlight > Elements/Attributes by Prefix** preferences page *(on page 101)*.

Related Information:

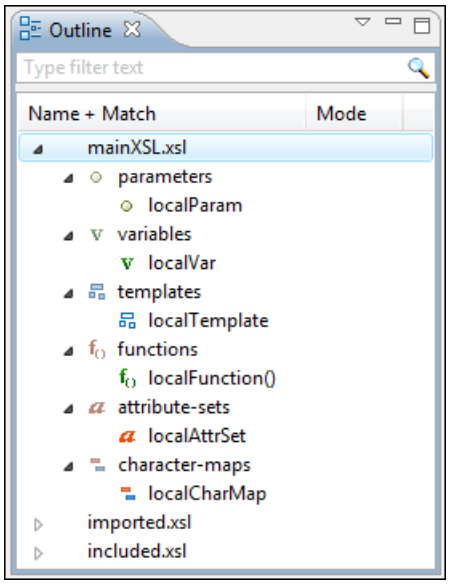
Customize Syntax Highlight colors *(on page 101)*

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 (on page 1326)*. 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 *(on page 200)*.

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

Figure 116. 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 *master file (on page 1326)*. 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 *(on page 144)*.

Configure displayed attributes

Displays the **XML Structured Outline** preferences page *(on page 144)*.

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 allows 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 *(on page 386)* 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 *(on page 386)* for more details.

Component Dependencies

Opens the **Component Dependencies** view *(on page 384)* that allows you to see the dependencies for the currently selected component.

Show Resource Hierarchy

Opens the **Resource Hierarchy/Dependencies** view *(on page 382)* that displays the hierarchy for the currently selected resource.

Show Resource Dependencies

Opens the **Resource Hierarchy/Dependencies** view *(on page 382)* that displays the dependencies of the currently selected resource.

Rename Component in

Renames the selected component. See XSLT Refactoring Actions *(on page 389)* 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 allows 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

Opens the **Component Dependencies** view *(on page 384)* that displays the dependencies of the currently selected component.

Rename Component in

Renames the currently selected component in the context of a scope that you define.

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Delete

Deletes the currently selected component.

Expand All

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

Collapse All

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

The stylesheet components information is presented on two columns: the first column presents the `@name` and `@match` attributes, the second column the `@mode` attribute. If you know the component name, match or mode, you can search it in the **Outline** view by typing one of these pieces of information in the filter text field from the top of the view or directly on the tree structure. When you type de component name, match or mode in the text field, you can switch to the tree structure using:

- Keyboard arrow keys
- **Enter** key
- **Tab** key
- **Shift-Tab** key combination

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

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

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

If no wildcards are specified, the string to search is used as a partial match.

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 Eclipse 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 Eclipse plugin has a predefined order of the groups in the **Outline** view:

- For location, the names of the files are sorted alphabetically. The file you are editing is located at the top of the list.
- For type, the order is: parameters, variables, templates, functions, set attributes, character-map.

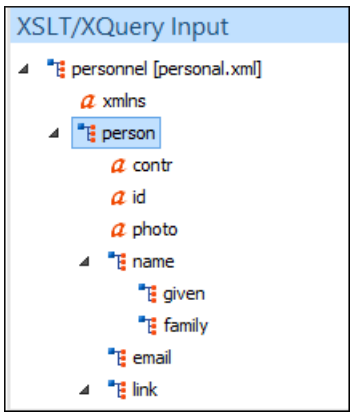
Note: When no grouping is available and the table is not sorted, Oxygen XML Developer Eclipse plugin sorts the components depending on their order in the document. Oxygen XML Developer Eclipse plugin also takes into account the name of the file that the components are part of.

XSLT Input View

The structure of the XML document associated to the edited XSLT stylesheet is displayed in a tree form in a view called the **XSLT Input** view. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu. The tree nodes represent the elements of the documents.

If you click a node in the **XSLT 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`, or `@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 117. XSLT Input View



Example:

For the following XML document:

```
<personnel>
```

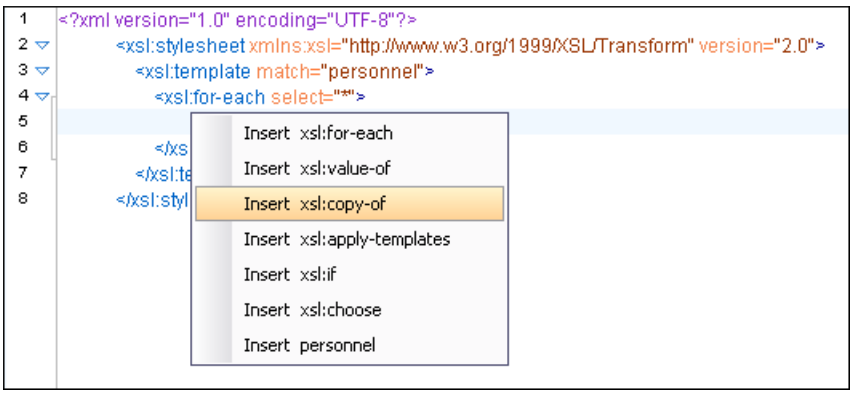
```
    <person id="Big.Boss">
        <name>
            <family>Boss</family>
            <given>Big</given>
        </name>
        <email>chief@oxygenxml.com</email>
        <link subordinates="one.worker"/>
    </person>
    <person id="one.worker">
        <name>
            <family>Worker</family>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



if you select **Insert xsl:copy-of** (for example), the resulting document will look like this:

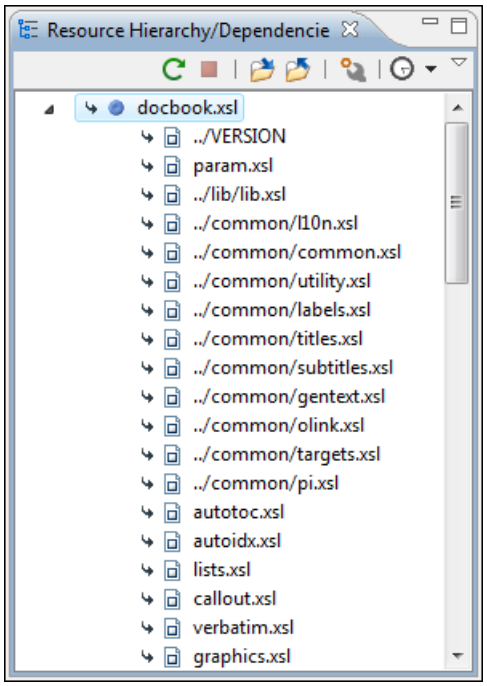
```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:template match="personnel">
        <xsl:for-each select="*">
            <xsl:copy-of  select="name/given"/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

XSLT Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view displays the hierarchy or dependencies for resources included in a stylesheet. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

If you want to see the hierarchy or dependencies of a stylesheet, select the desired stylesheet in the **Navigator** view *(on page 192)* and choose **Show Resource Hierarchy** or **Show Resource Dependencies** from the contextual menu.

Figure 118. Resource Hierarchy/Dependencies View



If you want to see the dependencies of a stylesheet, select the desired stylesheet in the **Navigator** view *(on page 192)* and choose **Resource Dependencies** from the contextual menu.

The following actions are available on the toolbar of 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 dependencies search scope

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 for a resource listed in the **Resource Hierarchy/Dependencies** view contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Go to reference

Opens the source document where the resource is referenced.

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 *(on page 200)*.

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 318)*

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 of the renamed resource(s)** - Select this option to update the references to the resource you are renaming. A **Preview** option is available that allows you to see what will be updated before selecting **Rename** to process the operation.

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)** - Select this option to update the references to the resource you are moving, in accordance with the new location and name. A **Preview** option is available that allows you to see what will be updated before selecting **Move** to process the operation.

XSLT Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected component. This is helpful if you want to see where components are used in the entire hierarchy. For example, if you want to find all the references where a given component is used.

If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

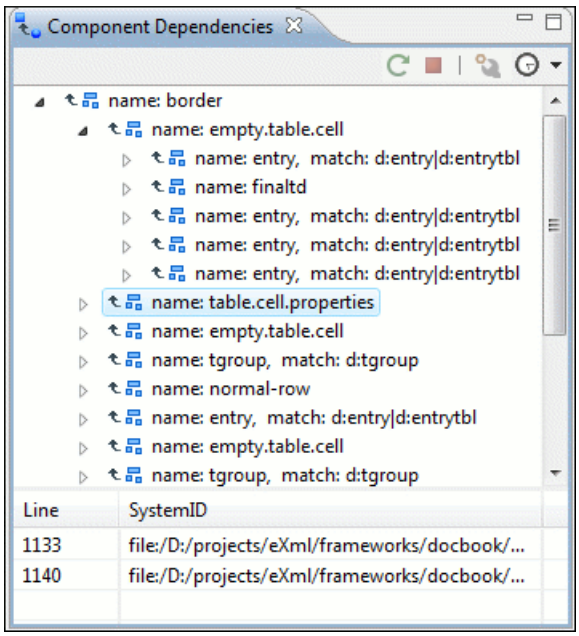
To see the dependencies of an XSLT component:

1. Right-click the desired component in the editor or **Outline** view.
2. Select the **Component Dependencies** action from the contextual menu.

The action is available for all named components (templates, variables, parameters, attribute sets, keys, functions, outputs).

If a component contains multiple references, a small table is displayed at the bottom of the view that contains all the references. When a recursive reference is encountered, it is marked with a special icon .

Figure 119. Component Dependencies View



The **Component Dependencies** view includes the following toolbar actions:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependency computation.

Configure

Allows you to choose the search scope for computing the dependencies structure. This is helpful for making sure all imported/included resources are computed.

History

Allows you to select from a list of the most recently used dependency computations.

In addition, the following actions are available in the contextual menu:

Go to First Reference

Selects the first reference of the currently selected component in the dependencies tree.

Go to Component

Shows the definition of the currently selected component in the dependencies tree.

Related Information:

Search and Refactor Operations Scope *(on page 318)*

Highlight Component Occurrences

When a component (for example variable or named template) is found at current cursor position, Oxygen XML Developer Eclipse 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 Eclipse 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 *(on page 36)* 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 *(on page 248)*.

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 currently 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:

Go to 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 318)*

XSLT Stylesheet Component Documentation Support

Oxygen XML Developer Eclipse plugin offers built-in support for documenting XSLT stylesheets. If the expanded *QName (on page 1328)* 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 Eclipse 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 *(on page 81)*.

Content Completion

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).

Adding Documentation Blocks

In **Text** mode, to add documentation blocks, press **Ctrl + Alt + D (Command + Alt + D on OS X)** or select **Add component documentation** from 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: Documentation Block Using Oxygen XML Developer Eclipse plugin Built-in Schema

```
<xd:doc>
  <xd:desc>
    <xd:p>Search inside parameter <xd:i>string</xd:i>
        for the last occurrence of parameter
    <xd:i>searched</xd:i>. The substring starting from the 0 position
        to the identified last occurrence will be returned.
    <xd:ref name="f:substring-after-last" type="function"
        xmlns:f="http://www.oxygenxml.com/doc/xsl/functions">See also
    </xd:ref>
    </xd:p>
  </xd:desc>
```

```
  <xd:param name="string">
    <xd:p>String to be analyzed</xd:p>
    </xd:param>
  <xd:param name="searched">
    <xd:p>Marker string. Its last occurrence will be identified</xd:p>
    </xd:param>
  <xd:return>
    <xd:p>A substring starting from the beginning of <xd:i>string</xd:i>
        to the last occurrence of <xd:i>searched</xd:i>.
        If no occurrence is found an empty string will be returned.
    </xd:p>
  </xd:return>
</xd:doc>
```

XSLT Documentation Links

Oxygen XML Developer Eclipse plugin includes support for links inside XSLT documentation blocks. Using a construct like `<xd:a docid="user-defined-id">TEXT</xd:a>` will cause the browser to scroll to the particular anchor (the defined ID) in the current document. Using a construct like `<xd:a href="http://www.my-web-site">TEXT</xd:a>` or `<xd:a href="local-file-path/filename">TEXT</xd:a>` will open the referenced link in a new tab.

Example: Documentation Links

```
<xd:doc xmlns:xd="http://www.oxygenxml.com/ns/doc/xsl" id="thisDoc">
   <xd:desc>
      <xd:p>
         <xd:ref name="test" type="variable">My test variable</xd:ref>
            <xd:a docid="thisDoc">Link to this documentation, see
the the id="thisDoc" above</xd:a>
            <xd:a docid="otherDocID" href="included.xsl">Link to
otherDocID defined in included.xsl</xd:a>
      </xd:p>
   </xd:desc>
</xd:doc>
```

Related Information:

Generating Documentation for an XSLT Stylesheet *(on page 398)*

XSLT 3.0 Text Value Templates

Oxygen XML Developer Eclipse plugin offers built-in support for XSLT 3.0 *Text Value Templates*, including content completion to present the variables, functions, and parameters from the current context and syntax highlighting.

A text node in the stylesheet is treated as a *text value template* if the following things are true:

- It is part of a sequence constructor or a child of an `<xsl:text>` instruction.
- There is an ancestor element with an `@[xsl:]expand-text` attribute and on the innermost ancestor element that has such an attribute, the value of the attribute is `yes`.

Example:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
expand-text="yes"
version="3.0">
  <xsl:param name="seq" as="xs:string*" select="'c', 'a', 'b', 'z'"/>
  <xsl:template name="main">
    {sort($seq)}
  </xsl:template>
</xsl:stylesheet>
```

For more information, see: W3C XSLT Specifications: Text Value Templates.

Related Information:

Content Completion in XPath Expressions *(on page 370)*

XSLT Refactoring Actions

Oxygen XML Developer Eclipse 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 the `<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. You can also use parts of an XPath expression to create the new functions.

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. Click the **Choose** button to select the destination stylesheet file. Oxygen XML Developer Eclipse 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. Click 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:attributes to attributes

Converts `<xsl:attribute>` elements to inline attributes for elements outside the XSL namespace. For example, the following element: It is the reverse of the **Convert attributes to xsl:attributes** action with the following limitations:

- The `<xsl:attribute>` element is "text only".
- The `<xsl:attribute>` element has a single `<xsl:text>` child element.
- The `<xsl:attribute>` element has a single `<xsl:value-of>` child element. In this case, the value of the attribute will be the XPath expression from the `@select` attribute surrounded by curly brackets (*text value template*).

```
<person>
  <xsl:attribute name="id">john.doe</xsl:attribute>
  <xsl:attribute name="email"><xsl:text>john.doe@example.com</xsl:text>
  </xsl:attribute>
  <xsl:attribute name="manager"><xsl:value-of select="person[@id='boss']/name"/>
  </xsl:attribute>
</person>
```

is converted to:

```
<person id="john.doe" email="john.doe@example.com" manager="{person[@id='boss']/name}"
/>
```

Convert xsl:if into xsl:choose/xsl:when

Converts one or more `<xsl:if>` element blocks into one or more `<xsl:when>` blocks surrounded by an `<xsl:choose>` element. If it is invoked on a selection, the selection must contain a well-formed fragment. If there is no selection, the `<xsl:if>` element that surrounds the content at the current cursor position is converted.

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.

Convert xsl:choose/xsl:when into xsl:if

Converts each `<xsl:when>` block into an `<xsl:if>` block. For the `<xsl:otherwise>` branch, it also adds an *and* statement to each negated form of the conditions. For example, the following block:

```
<xsl:choose>
    <xsl:when test="c1">
        <!-- XSLT statement 1 -->
    </xsl:when>
    <xsl:when test="c2">
        <!-- XSLT statement 2 -->
    </xsl:when>
    <xsl:when test="c3">
        <!-- XSLT statement 3 -->
    </xsl:when>
    <xsl:otherwise>
        <!-- XSLT "otherwise" statement-->
    </xsl:otherwise>
</xsl:choose>
```

is converted to:

```
<xsl:if test="c1">
    <!-- XSLT statement 1 -->
</xsl:if>
<xsl:if test="c2">
    <!-- XSLT statement 2 -->
</xsl:if>
<xsl:if test="c3">
    <!-- XSLT statement 3 -->
</xsl:if>
<xsl:if test="not(c1) and not(c2) and not(c3)">
    <!-- XSLT "otherwise" statement-->
</xsl:if>
```

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 Eclipse 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 Eclipse plugin allows you to edit the name of the variable.

Note: Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin allows you to edit the name of the parameter.

  Note: Oxygen XML Developer Eclipse 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.

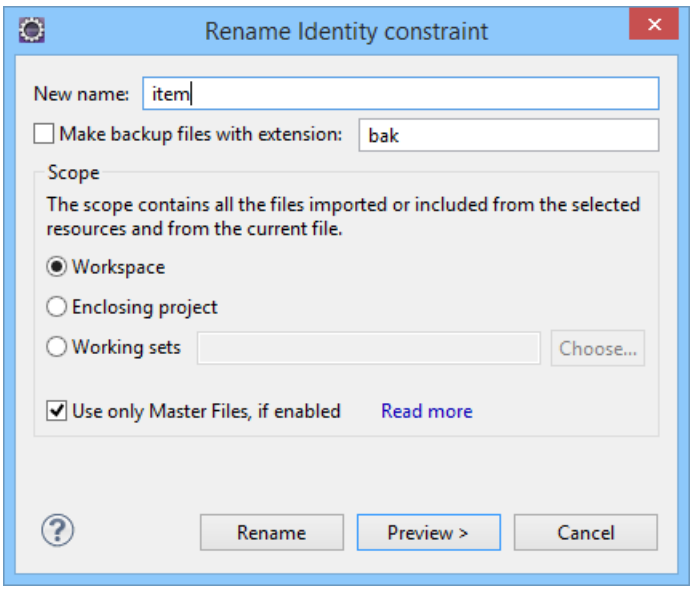
Rename Component

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 120. Rename Identity Constraint Dialog Box



Note: Many of these refactoring actions are also proposed by the *Quick Assist* support *(on page 394)*.

For more information about XSLT refactoring, watch our video demonstration:

https://www.youtube.com/embed/4ir5XWyp8Zo

XSLT Quick Assist Support

The *Quick Assist* support *(on page 1328)* 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.

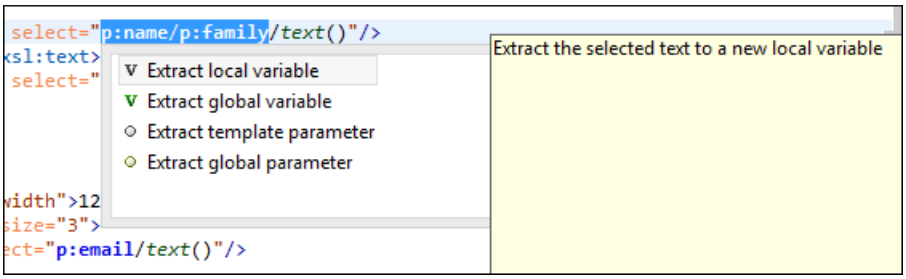
  Extract template parameter

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 121. 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.

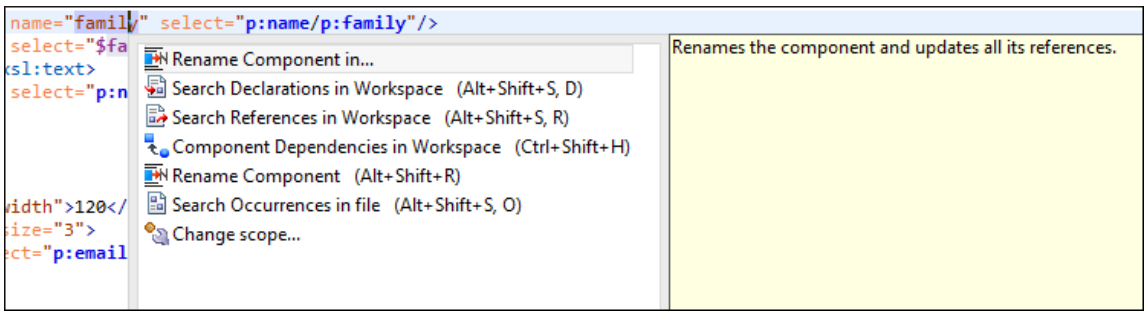
  Rename Component

  Allows you to rename the current component in-place.

  Search Occurrences

  Searches all occurrences of the component within the current file.

Figure 122. XSLT Quick Assist Support - Component Actions



Related Information:

Component Dependencies View *(on page 384)*

XSLT Hierarchy View *(on page 382)*

XSLT Refactoring Actions *(on page 389)*

Search and Refactor Operations Scope *(on page 318)*

XSLT Unit Test (XSpec)

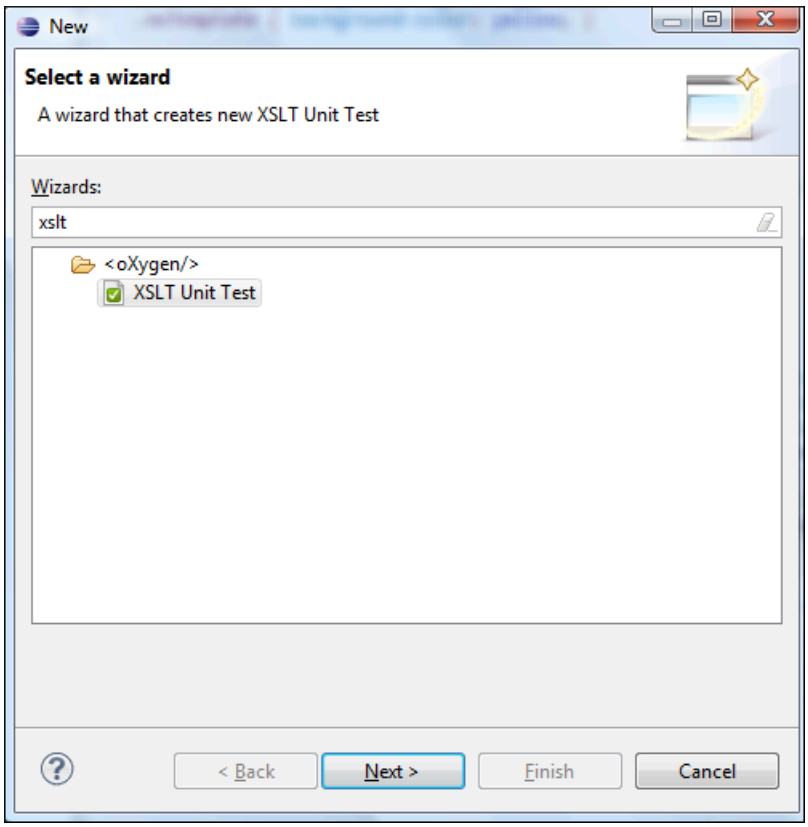
XSpec is a behavior driven development (BDD) *framework* for XSLT, XQuery, and Schematron. XSpec consists of syntax for describing the behavior of your XSLT, XQuery, or Schematron code, and some code that enables you to test your code against those descriptions.

Creating an XSLT Unit Test

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 *(on page 192)*. Oxygen XML Developer Eclipse 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.

When you create an XSpec document based on an XSL document, Oxygen XML Developer Eclipse plugin uses information from the validation and transformation scenarios associated with the XSL file. From the transformation scenario Oxygen XML Developer Eclipse plugin uses extensions and properties of Saxon 9.9.1.5, improving the Ant scenario associated with the XSpec document.

Figure 123. New XSLT Unit Test Wizard



Running an XSLT Unit Test

To run a Unit Test, open the XSpec file in an editor and click **Apply Transformation Scenario(s)** on the main toolbar. This will run the built-in **Run XSpec Test** transformation scenario that is defined in the XSpec *framework (on page 1325)*.

Testing a Stylesheet

An XSpec file contains one or more test scenarios. You can test a stylesheet in one of the following ways:

- Test an entire stylesheet.

  Testing is performed in a certain context. You can define a context as follows:
  - Inline context, building the test based on a string.

```
<x:scenario label="when processing a para element">
    <x:context>
        <para>...</para>
    </x:context>
    ...
</x:scenario>
```

  - Based on an external file, or on a part of an external file extracted with an XPath expression.

```
<x:scenario label="when processing a para element">
    <x:context href="source/test.xml" select="/doc/body/p[1]" />
    ...
```

```
</x:scenario>
```

- Test a function.

```
<x:scenario label="when capitalising a string">
  <x:call function="eg:capital-case">
    <x:param select="'an example string'" />
    <x:param select="true()" />
  </x:call>
  ...
</x:scenario>
```

- Test a template with a name.

```
<x:scenario label="when creating a table">
  <x:call template="createTable">
    <x:param name="nodes">
      <value>A</value>
      <value>B</value>
    </x:param>
    <x:param name="cols" select="2" />
  </x:call>
</x:scenario>
```

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.

Adding a Catalog to an XSpec Transformation

If your XSLT needs a catalog, you can add one to the XSpec transformation by doing one of the following:

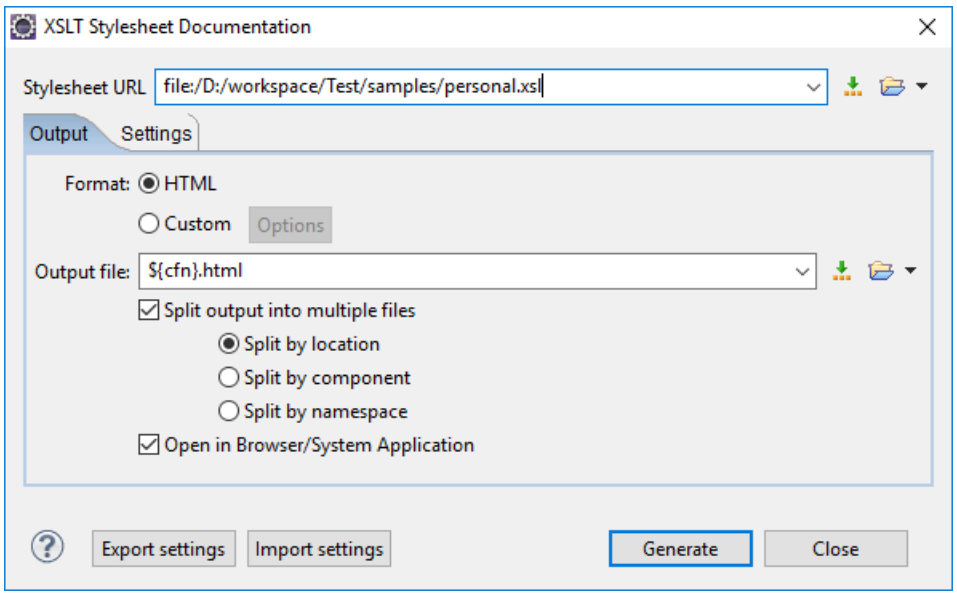
- If you are using a project *(on page 191)* in Oxygen XML Developer Eclipse plugin, create a `catalog.xml` file in the project directory. This catalog will then be loaded automatically.
- Edit *(on page 762)* the **Run XSpec Test** transformation scenario, go to the **Parameters** tab *(on page 731)*, and set the value of the `catalog` parameter to the location of your catalog file.

Generating Documentation for an XSLT Stylesheet

You can use Oxygen XML Developer Eclipse 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 *(on page 403)*, 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 *(on page 192)*.

Figure 124. 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:

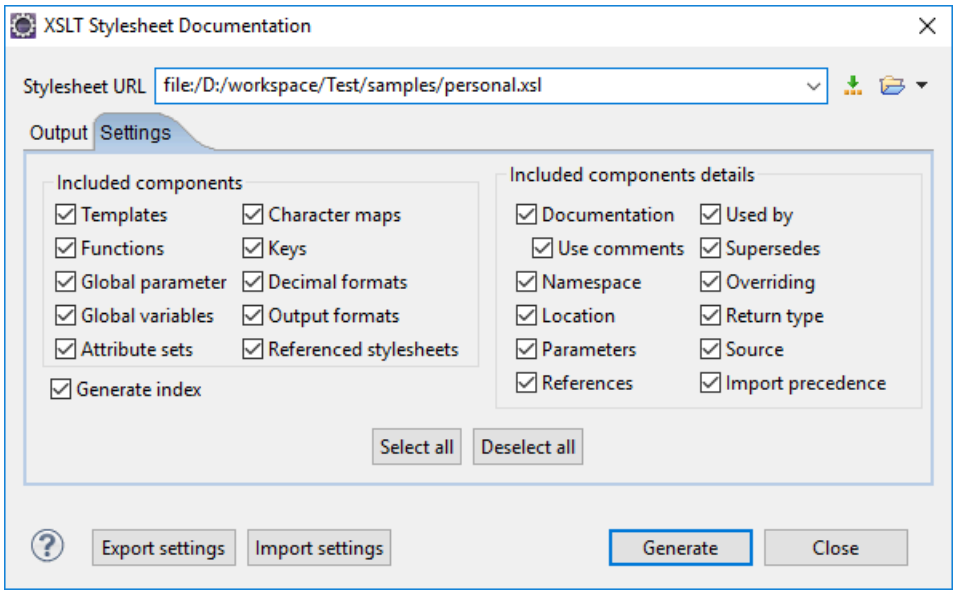
- **Format** - Allows you to choose between the following formats:
  - **HTML** - The documentation is generated in HTML output format *(on page 401)*.
  - **Custom** - The documentation is generated in a custom output format *(on page 403)*, allowing you to control the output. Click the **Options** button to open a **Custom format options** dialog box *(on page 404)* 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 125. 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 Eclipse 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 (on page 1325)*. 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 *(on page 403)* instead of the built-in HTML format *(on page 401)* 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, if you need the exported settings file for generating the documentation from the command-line interface).

Import settings - Reloads the settings from the exported file.

Generate - Use this button to generate the XSLT documentation.

Tip: This function can be executed from an automated command-line script, but it requires an additional **Oxygen XML Scripting License**. For more details, see https://www.oxygenxml.com/oxygen_scripting.html.

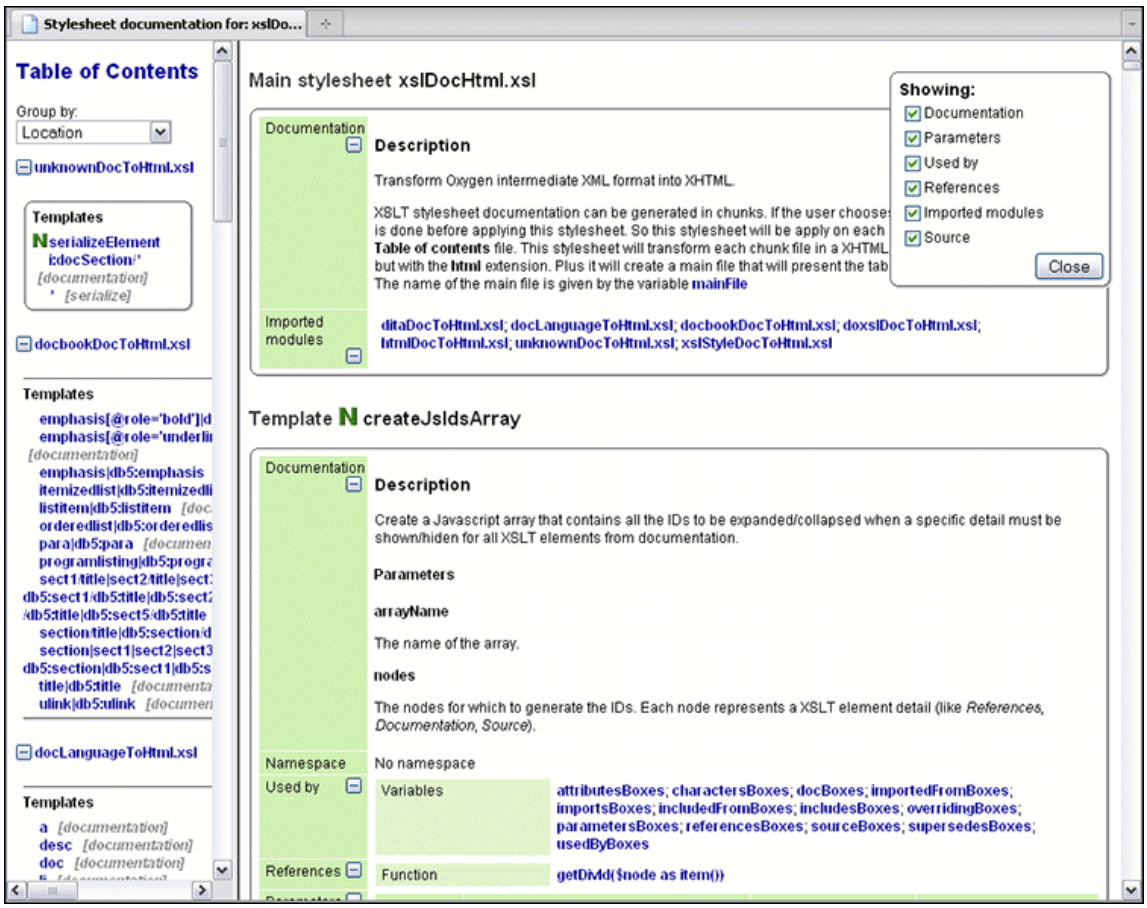
Related Information:

XSLT Stylesheet Component Documentation Support *(on page 387)*

Generate XSLT Documentation in HTML Format

When using the XSLT Stylesheet Documentation dialog box *(on page 398)* to generate XSLT documentation in HTML format, it is presented in a visual diagram style with various sections, hyperlinks, and options.

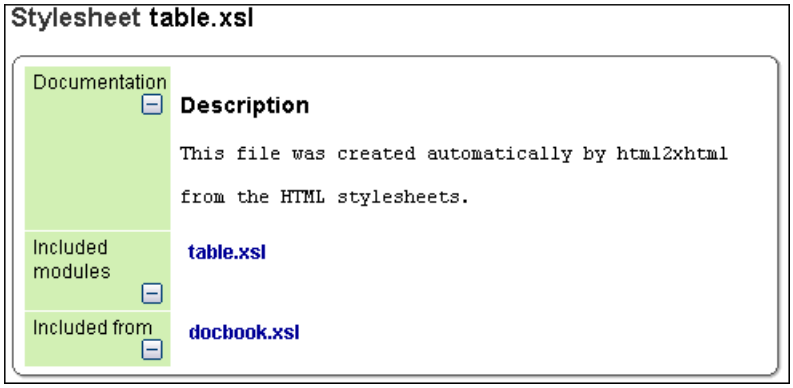
Figure 126. XSLT Stylesheet Documentation Example



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>` elements 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.

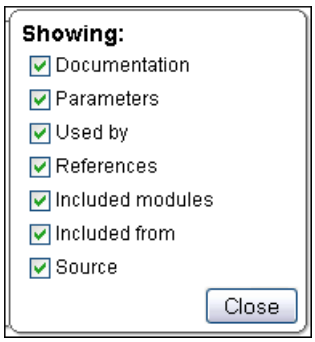
Figure 127. Information About an XSLT Stylesheet



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.

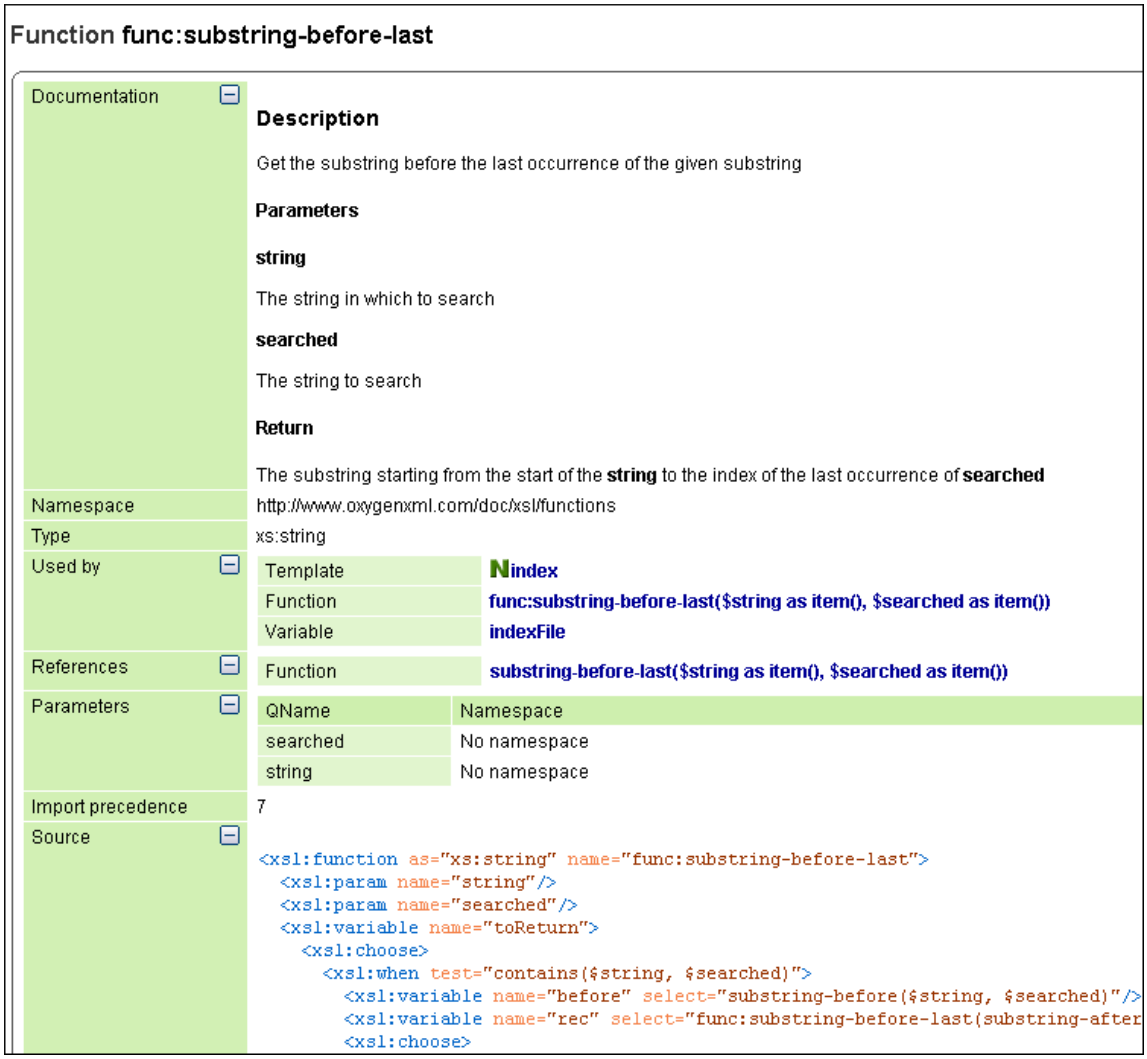
After the documentation is generated, you can collapse or expand details for some stylesheet XSLT elements by using the **Showing** options or the **Collapse** or **Expand** buttons.

Figure 128. Showing Options



For each element included in the documentation, the section presents the element type followed by the element name (value of the `@name` or `@match` attribute for match templates).

Figure 129. Documentation for an XSLT Element



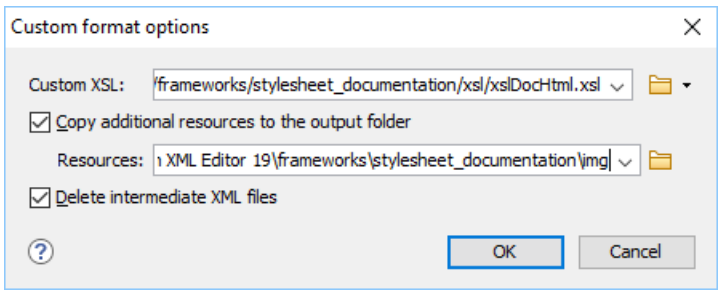
Generate XSLT Documentation in a Custom Format

XSLT stylesheet documentation can be also generated in a custom format. You must write your custom 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 built-in HTML, PDF, and DocBook formats. These stylesheets are available in *`[OXYGEN_INSTALL_DIR]`*`/frameworks/stylesheet_documentation/xsl`.

To generate XSLT documentation in a custom format:

1. Select **Tools > Generate Documentation > XSLT Stylesheet Documentation** to open the XSLT Stylesheet Documentation dialog box *(on page 398)*.
2. Select **Custom** for the **Format** and click the **Options** button.
3. In this next dialog box, specify your own stylesheet to transform the intermediary XML generated in the documentation process.
4. You can also choose to copy additional resources into the output folder or choose whether or not to keep the intermediate XML files created during the documentation process.
5. Click **OK** to close this dialog box and then click **Generate**.

Figure 130. Custom Format Options Dialog Box



Compiling an XSL Stylesheet for Saxon

As of Saxon 9.7, it is possible to export a compiled form of a stylesheet as an XML file (called a *stylesheet export file* or SEF). Oxygen XML Developer Eclipse plugin includes a simple tool called **Compile XSL Stylesheet for Saxon** (found in the **XML Tools** menu) that does this for you.

Use-Cases for a Stylesheet Export File (SEF)

- **Use Saxon-JS to run transformations in a browser** - A *stylesheet export file* (SEF) is needed if you want to use the Saxon-JS product to run transformations in a browser, as in the following example:

```
<script type="text/javascript" src="SaxonJS/SaxonJS.min.js"></script>
<script>
    window.onload = function() {
      SaxonJS.transform({
         stylesheetLocation: "books.sef",
         sourceLocation: "books.xml"
         });
```

```
    }
</script>
```

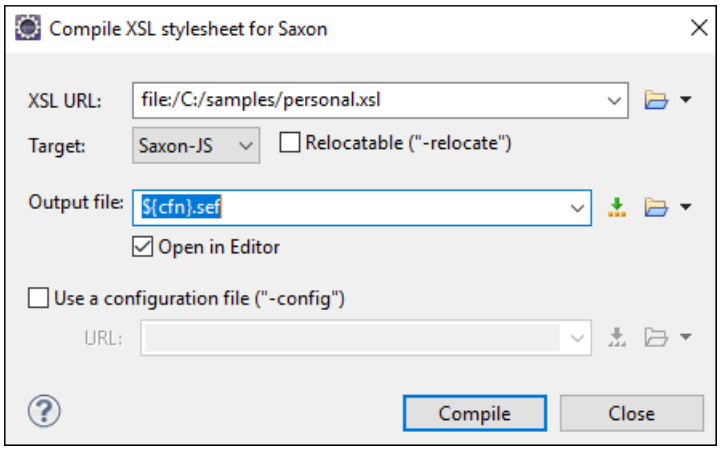
- **Use SEF to run transformations in Oxygen XML Developer Eclipse plugin** - You can also use a *stylesheet export file* (SEF) in Oxygen XML Developer Eclipse plugin to apply an XSLT transformation over an XML file. This requires **Saxon-EE** or **Saxon-PE** versions of the Saxon product and you must select one of those two versions for the **Target** when you configure the SEF file *(on page 405)*. When configuring the XSLT transformation, you will specify the SEF file in the **XSL URL** field *(on page 692)*.

Compiling an SEF File

The **Compile XSL Stylesheet for Saxon** tool can be found in the **XML Tools** menu and it compiles a specified stylesheet as an XML file (*stylesheet export file* with a file extension of `.sef`).

Selecting this tool opens the **Compile XSL Stylesheet for Saxon** dialog box that allows you to configure some options for conversion.

Figure 131. Compile XSL Stylesheet for Saxon Dialog Box



This dialog box includes the following options:

XSL URL

Allows you to select URL of the source XSL stylesheet. You can specify the URL by using the text field, the history drop-down, or the browsing actions in the **Browse** drop-down list.

Target

Allows you to select the type of Saxon product that the export file will be used with. You can choose **Saxon-JS**, **Saxon-EE**, **Saxon-PE**, or **Saxon-HE**.

Relocatable

Can be used to control the Saxon `-relocate` parameter. You can select this option to produce a *relocatable* export package (SEF) that can be deployed to a different location, with a different base URI.

Output file

You can specify the path where the output file will be saved by entering it in the text field, using the **Insert Editor Variables** button, or using the browsing actions in the **Browse** drop-down list.

Open in Editor

Select this option to open the resulting *stylesheet export file* in the main Oxygen XML Developer Eclipse plugin editing pane.

Use a configuration file ("-config")

Select this option if you want to use a Saxon 9.9.1.5 configuration file that will be executed for the XSLT transformation and validation processes. You can specify the path to the configuration file by entering it in the **URL** field, or by using the **Insert Editor Variables** button, or using the browsing actions in the **Browse** drop-down list.

Compile

Use this button to generate the *stylesheet export file* according the options selected in this dialog box.

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 Eclipse 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** ***(on page 446)*** - Allows you to edit XML Schema files in a source editing mode.
- **Grid editing mode** ***(on page 168)*** - Displays XML Schema files in a structured spreadsheet-like grid.
- **Design editing mode** ***(on page 169)*** - 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 303)*.

For more information about editing XML Schemas, see our video demonstration:

https://www.youtube.com/embed/vz1eIZELQgc

Related Information:

Associating a Schema to XML Documents *(on page 303)*

Working with the XML Schema Design Mode (XML Schema Diagram Editor)

This section includes topics that describe how to work with XML Schema documents in **Design** mode, including various features, actions that are available, and much more.

The **Design** mode in Oxygen XML Developer Eclipse plugin provides a simple and expressive XML Schema diagram editor for working with XML Schema documents. The schema diagram helps both the content authors who want to understand a schema and schema designers who develop complex schemas.

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

For more information about designing XML Schemas, watch our video demonstration:

https://www.youtube.com/embed/vz1eIZELQgc

Navigation in the XML Schema Design Mode

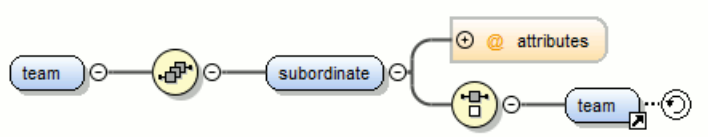
The following editing and navigation features work for all types of schema components in the XML Schema **Design** mode:

- 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 *(on page 450)*:
  - **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 **Go to Definition** action.
- You can expand and see the contents of the imports/includes/redefines in the diagram. To edit components from other schemas, the schema for each component will be opened as a separate file in Oxygen XML Developer Eclipse plugin.

Tip: If an XML Schema referenced by the currently open 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 132. Recursive Reference

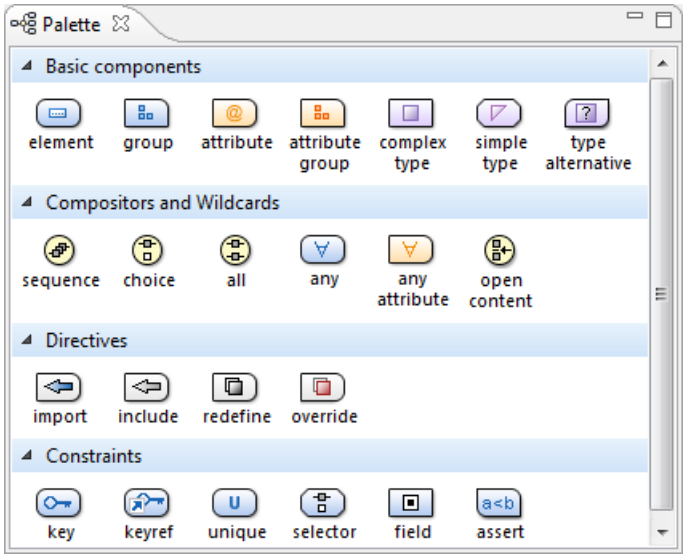


XML Schema Palette View (Available in Design Mode)

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 *(on page 126)*. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

Figure 133. Palette View

Figure 134. 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 *(on page 84)* (default color is red).

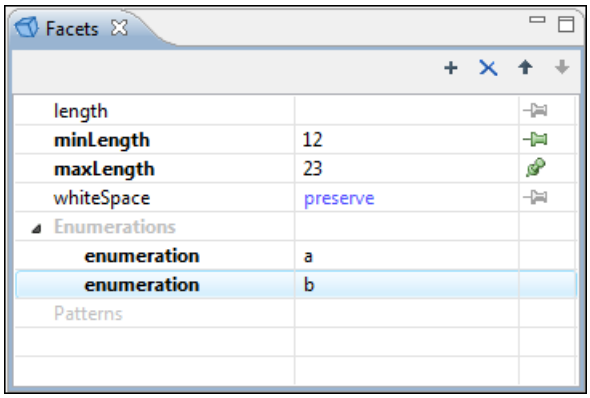
For more information about the Schema palette, watch our video demonstration:

https://www.youtube.com/embed/KalHUXmpuAA

XML Schema Facets View (Available in Design Mode)

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 by selecting it from the **Window > Show View** menu.

Figure 135. 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 currently 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 *(on page 84)*.

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 *(on page 480)* 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 click the **Pin** button.

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 *(on page 450)* 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 the **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 *(on page 88)*.

When editing references, you can choose from a list of available components. A component from an imported schema whose 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 currently edited schema.

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

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

Related Information:

Searching and Refactoring Actions in XML Schemas *(on page 458)*

XML Schema Component Dependencies View *(on page 456)*

XML Schema Resource Hierarchy / Dependencies View *(on page 454)*

Generating Sample XML Files *(on page 461)*

Schema Design Preferences *(on page 87)*

Contextual Menu Actions in the Design Mode

The contextual menu of the **Design** mode includes the following actions:

Go to Definition (Ctrl + Shift + Enter)

Shows the definition for the currently 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 *(on page 452)* and the **Facets** view *(on page 409)*. 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 *(on page 452)* for the schema or by double-clicking the schema component.

Edit Annotations

Allows you to edit the annotation for the selected schema component in the **Edit Annotations** dialog box. You can perform the following operations in the dialog box:

- **Edit all `appinfo/documentation` items for a specific annotation** - All `appinfo/documentation` items for a specific annotation are presented in a table and can be easily edited. Information about an annotation item includes: type (`documentation/appinfo`), content, source (optional, specify the source of the `documentation/appinfo` element) and `xml:lang`. The content of a `documentation/appinfo` item can be edited in the **Content** area below the table.
- **Insert/Insert before/Remove `documentation/appinfo`** - The **Add** button allows you to insert a new annotation item (`documentation/appinfo`). You can add a new item before the item selected in table by pressing the **Insert Before** button. Also, you can delete the selected item using the **Remove** button.
- **Move items up/down** - Do this by using 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 also presented in the diagram below the referenced component. To edit the annotations, use the **Edit Annotations** action from the contextual menu. 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.

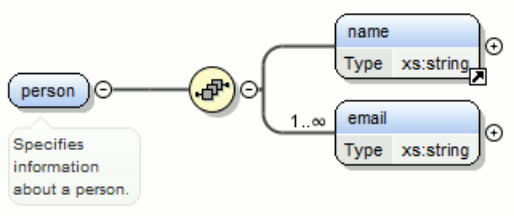
Change XML Schema Version

Use this action to change the XML Schema version of the current document.

Extract Global Element

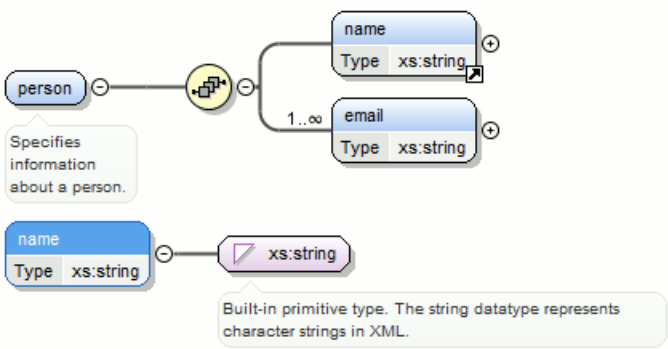
This action is available for local elements. A local element is made global and is replaced with a reference to the global element. The local element properties that are also valid for the global element declaration are kept.

Figure 136. Extracting a Global Element



If you use the **Extract Global Element** action on a `<name>` element, the result is:

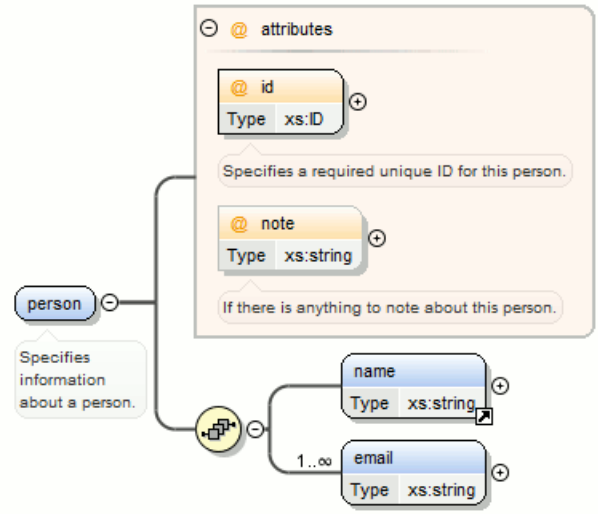
Figure 137. Extracting a Global Element on a `<name>` Element



Extract Global Attribute

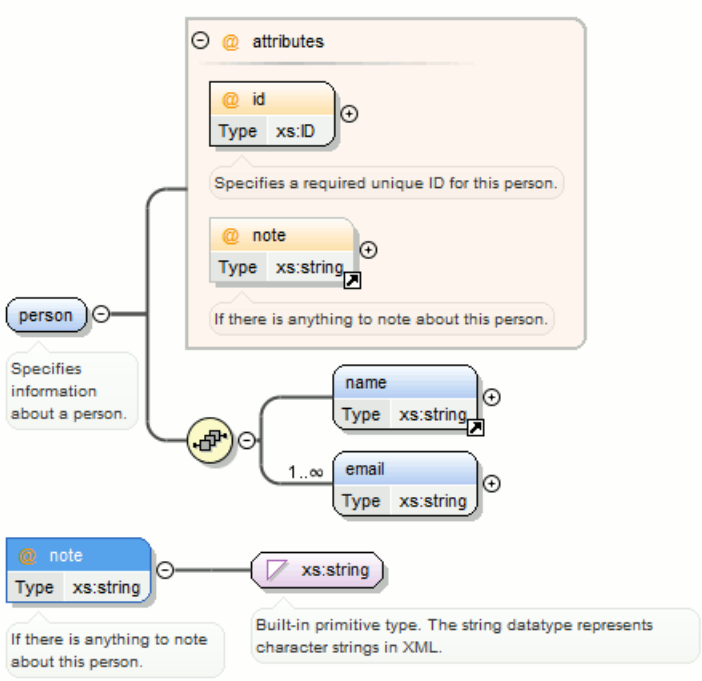
This action is available for local attributes. A local attribute is made global and replaced with a reference to the global attribute. The properties of local attribute that are also valid in the global attribute declaration are kept.

Figure 138. Extracting a Global Attribute



If you use the **Extract Global Attribute** action on a `@note` attribute, the result is:

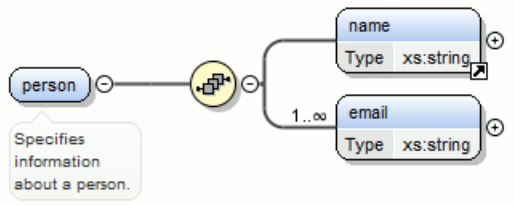
Figure 139. Extracting a Global Attribute on a `@note` Attribute



Extract Global Group

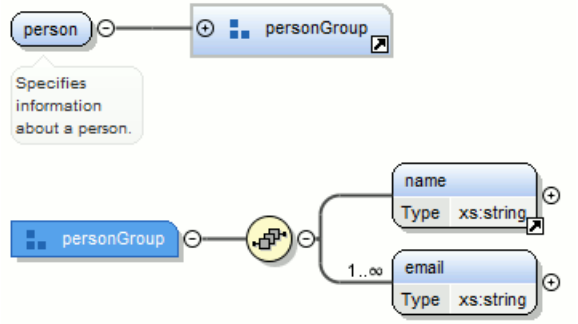
This action is available for compositors (sequence, choice, all). This action extracts a global group and makes a reference to it. The action is available only if the parent of the compositor is not a group.

Figure 140. Extracting a Global 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:

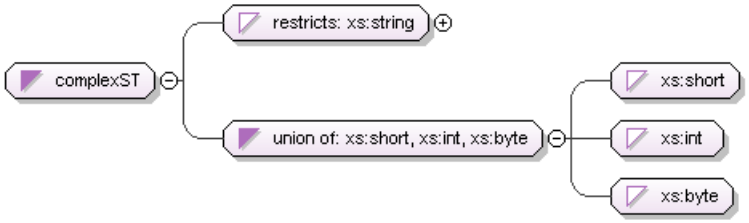
Figure 141. Extracting a Global Group on a `<sequence>` Element



Extract Global Type

This action is 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 142. 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 143. Extracting a Global Simple Type on a `union` Component

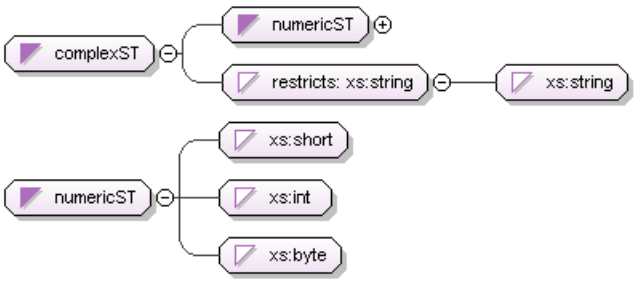
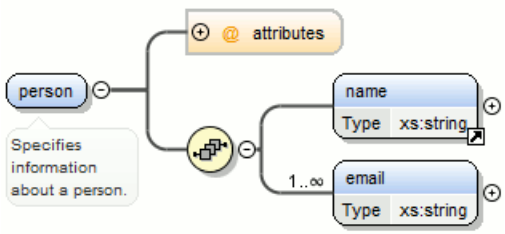
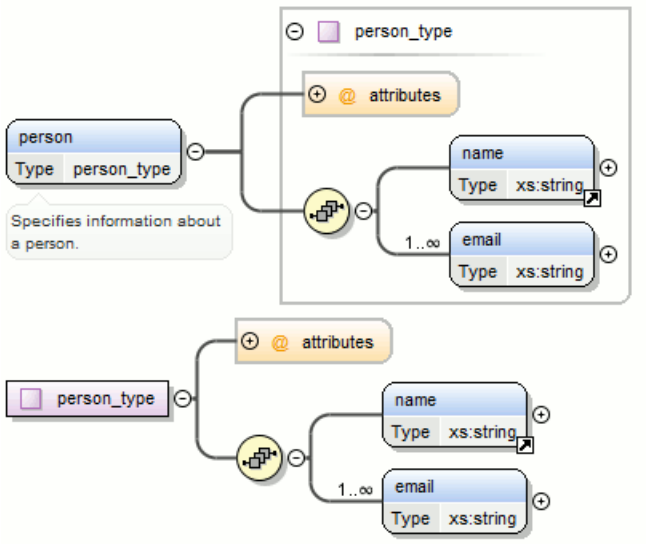


Figure 144. Extracting a Global Complex Type



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

Figure 145. Extracting a Global Complex Type on a `<person>` Element



Rename Component in

Renames the selected component.

Cut Ctrl + X (Command + X on OS X)

Cuts the selected component(s).

Copy Ctrl + C (Command + C on OS X)

Copies the selected component(s) to the clipboard.

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)

Pastes the component(s) from the clipboard as children of the selected component.

Paste as Reference

Creates references to the copied component(s). If not possible, a warning message is displayed.

Remove Delete

Removes 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

Opens the **Component Dependencies** view *(on page 456)* that allows you to see the dependencies for the currently selected component.

Show Resource Hierarchy

Opens the **Resource Hierarchy / Dependencies** view *(on page 454)* that allows you to see the hierarchy for the currently selected resource.

Show Resource Dependencies

Allows you to see the dependencies for the currently 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

Saves the diagram as image, in JPEG, BMP, SVG or PNG format.

Generate Sample XML Files

Generates XML files using the current opened schema. The selected component is the XML document root. See more in the Generate Sample XML Files *(on page 461)* section.

Flatten Schema

Recursively adds the components of included Schema files to the main one. It also flattens every imported XML Schema from the hierarchy.

Options

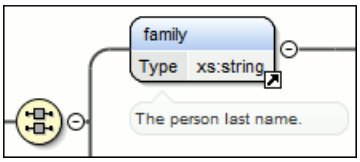
Opens the Schema preferences page *(on page 87)*.

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 146. Example: Required Component



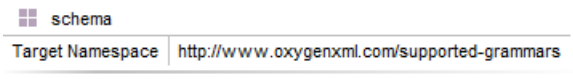
- A thin line to identify a connection with an optional component (in the following image, `<email>` is an optional element).

Figure 147. Example: Optional Component

The topics in this section provide details about all of the available components and their symbols as they appear in an XML schema diagram.

xs:schema

Figure 148. 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 *block* attribute of *xs:element* and *xs:complexType*	#all, extension, restriction, substitution, restriction extension, restriction substitution, extension substitution, restriction extension substitution, [Empty]
Final Default	Default value of the *final* attribute of *xs:element* and *xs:complexType*	#all, restriction, extension, restriction extension, [Empty]
Default Attributes	Specifies a set of attributes that apply to every complex Type in a schema document	Any
XPath Default Namespace	The default namespace used when the XPath expression is evaluated	##defaultNamespace, ##targetNamespace, ##local
Version	Schema version	Any token
ID	The schema ID	Any ID
Component	The edited component name	Not editable property
SystemID	The schema system ID	Not editable property

xs:element

Figure 149. 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 (on page 1328)* 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

Table 5. *xs:element* Properties (continued)

Property Name	Description	Possible Values	Mentions
			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

Table 5. *xs:element* Properties (continued)

Property Name	Description	Possible Values	Mentions
Min Occurs	Minimum number of occurrences of the element	A numeric positive value. Default value is 1	Only for references/local elements
Max Occurs	Maximum number of occurrences of the element	A numeric positive value (default value is 1)	Only for references/local elements
Substitution Group	Qualified name of the head of the substitution group 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
Nillable	When this attribute is set to true, the element can be declared as nil using an *xsi:nil* attribute in the instance documents	true/false	For global elements and element references
Target Namespace	Specifies the target namespace for local element and attribute declarations. The namespace URI may be different from the schema target namespace. This property is available for local elements only.	Not editable property	For all elements
Block	Controls 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	#all, restriction, extension,substitution, extension restriction, extension substitution, restriction	For global elements and ele-

Table 5. *xs:element* Properties (continued)

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

xs:attribute

Figure 150. 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 (on page 1328)* for references	For all local or global attributes. If missing, will be displayed as '[attribute]' in the diagram.
Is Reference	When set, the local attribute is a reference	true/false	For local attributes
Type	Qualified name of a simple type	All global simple types and built-in simple types. In addition another 3 proposals are present: [anonymous restriction], [anonymous list], [anonymous union] for creating anonymous simple types more easily.	For all attributes. For references, the type is set to the referenced attribute.
Default	Default value. When specified, an attribute is added by the schema processor (if it is missing from the instance XML document) and it is given this value. The default and 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	unqualified/qualified	For local attributes

Table 6. *xs:attribute* Properties (continued)

Property Name	Description	Possible Value	Mentions
	XML document). The default value for this attribute is specified by the *attributeFormDefault* attribute of the *xs:schema* document element.		
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 151. The *xs:attributeGroup* Component

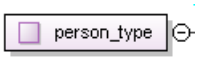
Defines an attribute group to be used in complex type definitions. See more info at http://www.w3.org/TR/xmlschema11-1/#element-attributeGroup.

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 152. 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 infoset contributions for elements.
- Limiting the ability to derive additional types from a given complex type.
- Controlling the permission to substitute, in an instance, elements of a derived type for elements declared in a content model to be of a given complex type.

Tip: A complex type 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
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

Table 8. *xs:complexType* Properties (continued)

Property Name	Description	Possible Values	Mentions
Abstract	When set to *true*, this complex type cannot be used directly in the instance documents and needs to be substituted using an *xsi:type* attribute	true/ false	For global and anonymous complex types
Block	Controls if a substitution (either through a *xsi:type* or substitution groups) can be performed for a complex type, which is an extension or a restriction of the current complex type. This attribute can only block such substitutions (it cannot "unblock" them), which can also be blocked in the element definition. The default value is defined by the *blockDefault* attribute of *xs:schema*.	all, extension, restriction, extension restriction, [Empty]	For global complex types
Final	Controls whether the complex type can be further derived by extension or restriction to create new complex types	all, extension, restriction, extension restriction, [Empty]	For global complex types
Default Attributes Apply	The *schema* element can carry a *defaultAttributes* attribute, which identifies an attribute group. Each *complexType* defined in the schema document then automatically includes that attribute group, unless this is overridden by the *defaultAttributesApply* attribute on the *complexType* element.	true/ false	This property is available only for XML Schema 1.1
ID	The component ID	Any ID	For all complex types
Component	The edited component name	Not editable property	For all complex types
Namespace	The component namespace	Not editable property	For all complex types

Table 8. *xs:complexType* Properties (continued)

Property Name	Description	Possible Values	Mentions
System ID	The component system ID	Not editable property	For all complex types

xs:simpleType

Figure 153. 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	A simple type category	restriction, list, or union	For all simple types
Base Type	A simple type definition component. Required if derivation method is	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

Table 9. *xs:simpleType* Properties (continued)

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

Table 9. *xs:simpleType* Properties (continued)

Name	Description	Possible Values	Scope
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 154. The *xs:alternative* Component

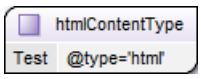


Table 10. *xs:alternative* Properties

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]
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	An XPath expression

Table 10. *xs:alternative* Properties (continued)

Name	Description	Possible Values
	ancestor elements are accessible in the XPath expression. When you edit this property, the content completion list of proposals offers XPath expressions.	
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 155. 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

Table 11. *xs:group* Properties (continued)

Property Name	Description	Possible Values	Mentions
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 156. 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 157. 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 158. 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 159. 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 160. 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 161. *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 162. *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 163. *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 164. 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.

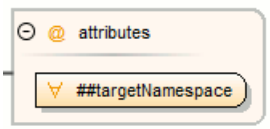
The graphical representation also contains the value for the *minOccurs* and *maxOccurs* properties (for 0..1 and 1..1 occurs the values are implied by the connector style) and the connectors to the wildcard are drawn using dotted lines if the wildcard is optional.

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
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 165. The *xs:anyAttribute* Component



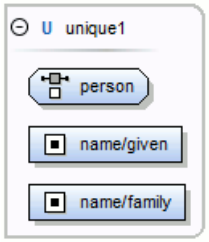
Enables the author to extend the XML document with attributes not specified by the schema. See more info at http://www.w3.org/TR/xmlschema11-1/#element-anyAttribute.

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 166. 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

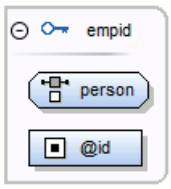
Property Name	Description	Possible Values
Name	The unique name (always required)	Any NCName
ID	The component ID	Any ID

Table 20. *xs:unique* Properties (continued)

Property Name	Description	Possible Values
Component	The edited component name	Not editable property
Namespace	The component namespace	Not editable property
System ID	The component system ID	Not editable property

xs:key

Figure 167. The *xs:key* Component



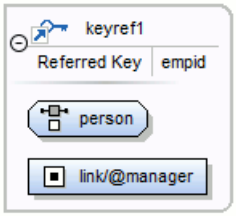
Specifies an attribute or element value as a key (unique, non-nullable and always present) within the containing element in an instance document. See more info at http://www.w3.org/TR/xmlschema11-1/#element-key.

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 168. 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
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 169. 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 170. 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 171. 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	##default-Namespace, ##target-Namespace, ##local
ID	Specifies the component ID	Any ID

Table 25. *xs:assert* Properties (continued)

Property Name	Description	Possible Values
Component	The edited component name	Not editable property
System ID	The component system ID	Not editable property

xs:openContent

Figure 172. 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

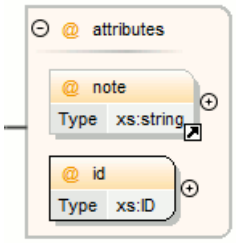
Note: This component is available for XML Schema 1.1 only. To change the version of the XML Schema, open the **Preferences** dialog box *(on page 36)* and go to **XML > XML Parser > XML Schema**.

Constructs Used to Group Schema Components

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

Attributes

Figure 173. Attributes Construct



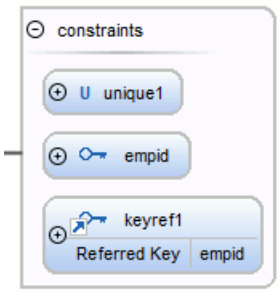
Groups all attributes and attribute groups belonging to a complex type.

Table 27. *attributes* Properties

Property Name	Description	Possible Values
Component	The element that has the attributes displayed	Not editable property
System ID	The component system ID	Not editable property

Constraints

Figure 174. Constraints Construct



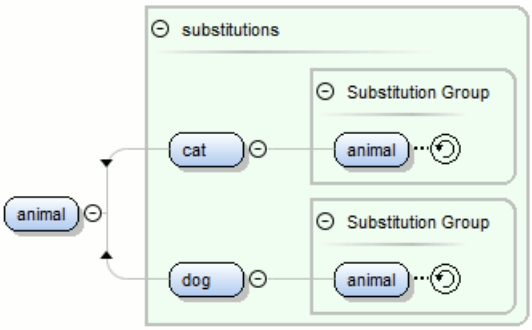
Groups all constraints (*xs:key (on page 440)*, *xs:keyRef (on page 440)*, or *xs:unique (on page 439)*) belonging to an element.

Table 28. *constraints* Properties

Property Name	Description	Possible Values
Component	The element that has the constraints displayed	Not editable property
System ID	The component system ID	Not editable property

Substitutions

Figure 175. Substitutions Construct



Groups all elements that can substitute the current element.

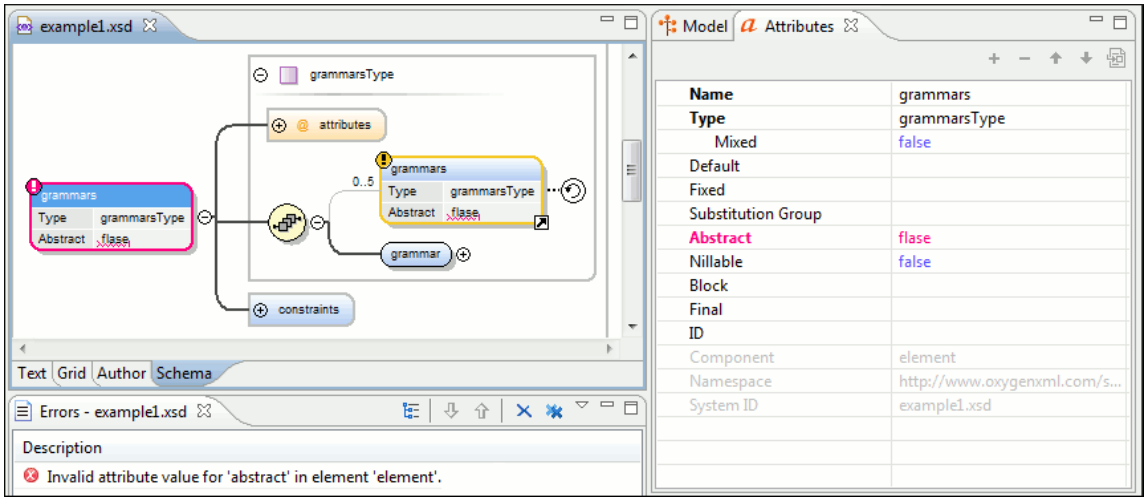
Table 29. *substitutions* Properties

Property Name	Description	Possible Values
Component	The element that has the substitutions 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 Eclipse plugin XML documents validation *(on page 277)* capability.

Figure 176. XML Schema Validation



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

- In the Attributes View *(on page 452)*.
- 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 *(on page 409)*.

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 *(on page 84)*. 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 *(on page 312)* 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 currently 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.

Editing XML Schema in Text Editing Mode

The Oxygen XML Developer Eclipse 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 *(on page 458)*. The Outline view has two display modes: the standard outline *(on page 240)* mode and the components *(on page 450)* mode.

A diagram of the XML Schema can be presented side by side with the text. To activate the diagram presentation, select the *Show Full Model XML Schema diagram* option *(on page 88)* in the **Diagram** preferences page *(on page 88)*.

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 Eclipse 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 *(on page 200)*, or using a validation scenario.

To set a *master file* using a validation scenario, add validation units that point to the main schemas. Oxygen XML Developer Eclipse 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 a *master file (on page 1326)* include:

- Correct validation of a module in the context of a larger schema structure.
- *Content Completion Assistant (on page 1324)* displays all the referable components valid in the current context. This include components defined in modules other than the currently edited one.
- The **Outline** view *(on page 450)* 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 *(on page 36)*, go to **Editor > Document Checking**, and deselect the **Enable automatic validation** option *(on page 84)*.

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 Eclipse 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.

Validation of an XML Schema containing a type definition with a `@minOccurs` or `@maxOccurs` attribute having a value larger than 256 limits the value to 256 and issues a warning about this restriction in the Message panel at the bottom of the Oxygen XML Developer Eclipse plugin window. Otherwise, for large values of the `@minOccurs` and `@maxOccurs` attributes, the validator fails with an **OutOfMemory** error that might make Oxygen XML Developer Eclipse plugin unusable without restarting the entire application.

Important: If the schema imports only a namespace without specifying the schema location and a catalog is set up *(on page 312)* to map the namespace to a certain location, both validation and the schema components will correctly identify the imported schema.

Related Information:

Validating XML Documents Against a Schema *(on page 279)*

Embedding Schematron Rules in XML Schema or RELAX NG *(on page 595)*

Validation Scenario *(on page 287)*

Associating a Schema to XML Documents *(on page 303)*

Presenting Validation Errors in Text Mode *(on page 281)*

Quick Fixes for DTD, XSD, and Relax NG Errors

Oxygen XML Developer Eclipse plugin offers *Quick Fixes (on page 1328)* 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 Eclipse 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 303)*

Content Completion in XML Schema

The intelligent *Content Completion Assistant (on page 1324)* allows you to quickly identify and insert elements, attributes, and attribute values that are valid in the current 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 *(on page 36)*, go to **Editor > Content Completion**, and deselect the **Enable content completion** option *(on page 78)*.

When active, the *Content Completion Assistant* displays a list of context-sensitive proposals valid at the current cursor position. It can be manually activated with the **Ctrl + Space (Command + Space on OS X)** shortcut. 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 proposal 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 *(on page 484)*, Oxygen XML Developer Eclipse plugin populates the proposals list with information taken either from XML Schema 1.0 or 1.1.

Oxygen XML Developer Eclipse 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>` or `<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 *(on page 81)* preferences page.

Syntax Highlighting in XML Schema

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for XML Schema files, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
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 **XSD** tab in the **Preview** pane to see the effects of your changes.

Tip: Oxygen XML Developer Eclipse plugin also allows you to specify syntax highlighting colors for specific XML elements and attributes with specific namespace prefixes. This can be done in the **Editor > Syntax Highlight > Elements/Attributes by Prefix** preferences page *(on page 101)*.

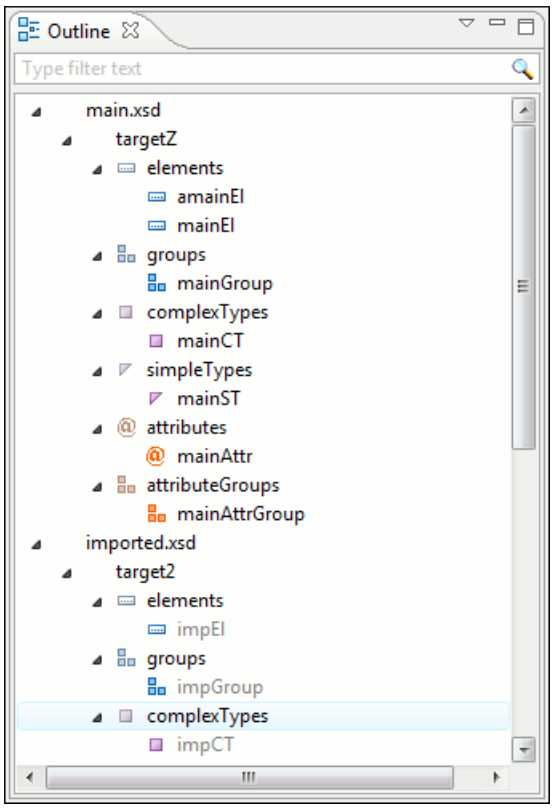
Related Information:

Customize Syntax Highlight colors *(on page 101)*

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 by selecting it from the **Window > Show View** menu.

Figure 177. 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 *master files (on page 1326)*. 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 *master files (on page 1326)*) 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

Opens the **Component Dependencies** view *(on page 456)* that allows you to see the dependencies for the currently selected component.

Show Resource Hierarchy

Opens the **Resource Hierarchy / Dependencies** view *(on page 318)* that allows you to see the hierarchy for the currently selected resource.

Show Resource Dependencies

Opens the **Resource Hierarchy / Dependencies** view *(on page 318)* that allows you to see the dependencies for the currently selected resource.

Rename Component in

Renames the selected component.

Generate Sample XML Files

Generate XML files using the currently open 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 (such as `*` or `?`) 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 458)*

XML Schema Component Dependencies View *(on page 456)*

XML Schema Resource Hierarchy / Dependencies View *(on page 454)*

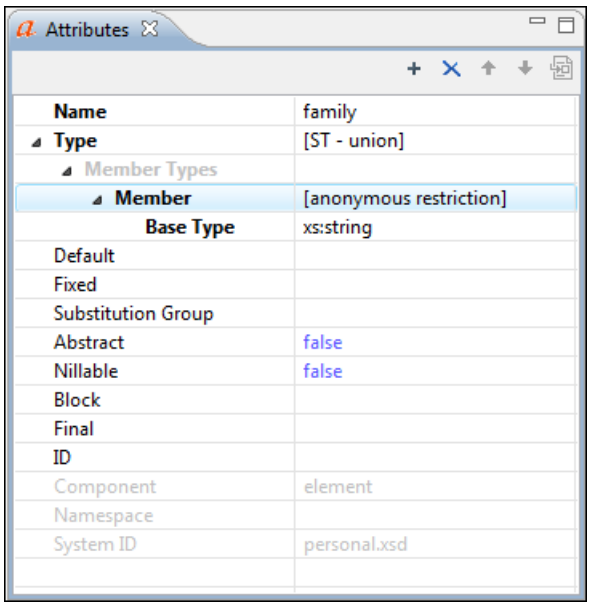
Generating Sample XML Files *(on page 461)*

Editing Relax NG Schema in the Master Files Context *(on page 526)*

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 by selecting it from the **Window > Show View** menu.

Figure 178. 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 currently 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 *(on page 84)*.

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 *(on page 446)* 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.

Go to Definition

Shows the definition for the selected type.

Show Facets

Allows you to edit the facets for a simple type.

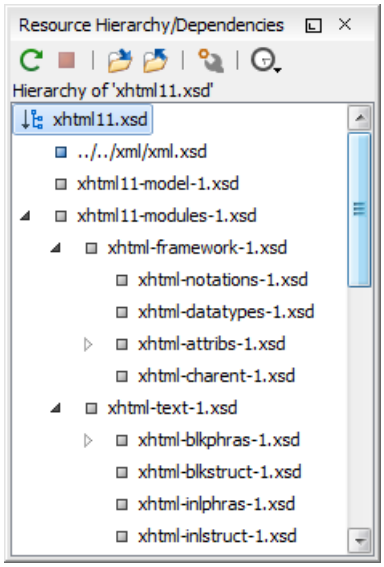
XML Schema Resource Hierarchy / Dependencies View

The **Resource Hierarchy / Dependencies** view displays the hierarchy or dependencies for resources included in an XML Schema. If the view is not displayed, it can be opened by selecting it 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.)

If you want to see the hierarchy or dependencies of an XML schema, select the desired schema in the **Navigator** view *(on page 192)* and choose **Show Resource Hierarchy** or **Show Resource Dependencies** from the contextual menu.

Figure 179. Resource Hierarchy/Dependencies View



The following actions are available on the toolbar of 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 dependencies search scope

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 for a resource listed in the **Resource Hierarchy/Dependencies** view contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Go to reference

Opens the source document where the resource is referenced.

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 *(on page 200)*.

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 *(on page 456)*.

Related Information:

Search and Refactor Operations Scope *(on page 318)*

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 of the renamed resource(s)** - Select this option to update the references to the resource you are renaming. A **Preview** option is available that allows you to see what will be updated before selecting **Rename** to process the operation.

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)** - Select this option to update the references to the resource you are moving, in accordance with the new location and name. A **Preview** option is available that allows you to see what will be updated before selecting **Move** to process the operation.

XML Schema Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected component. This is helpful if you want to see where components are used in the entire hierarchy. For example, if you want to find all the references where a given component is used.

If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

To see the dependencies of an XML Schema component:

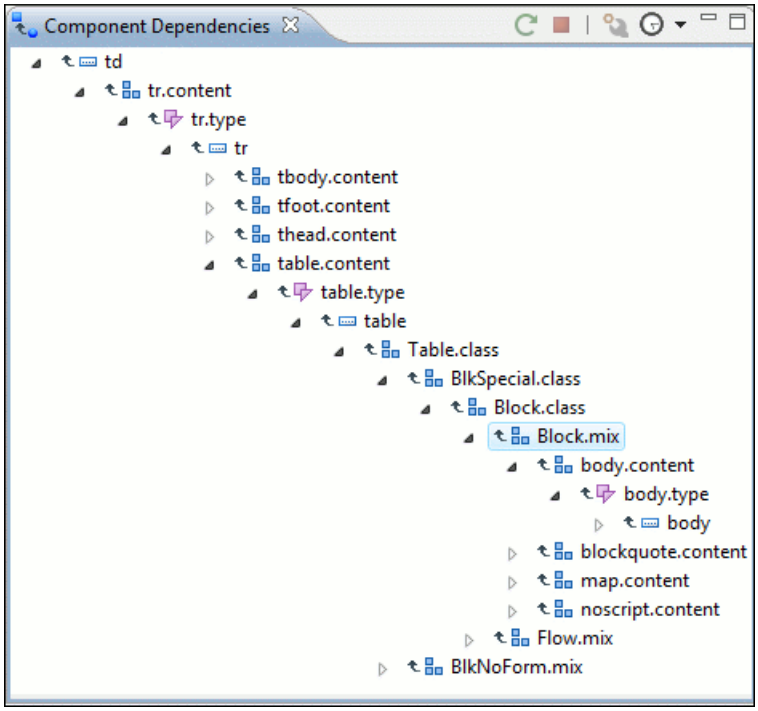
1. Right-click the desired component in the editor or **Outline** view.

2. Select the **Component Dependencies** action from the contextual menu.

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

If a component contains multiple references, a small table is displayed at the bottom of the view that contains all the references. When a recursive reference is encountered, it is marked with a special icon .

Figure 180. Component Dependencies View



The **Component Dependencies** view includes the following toolbar actions:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependency computation.

Configure

Allows you to choose the search scope for computing the dependencies structure. This is helpful for making sure all imported/included resources are computed.

History

Allows you to select from a list of the most recently used dependency computations.

In addition, the following actions are available in the contextual menu:

Go to First Reference

Selects the first reference of the currently selected component in the dependencies tree.

Go to Component

Shows the definition of the currently selected component in the dependencies tree.

For more information, see the **Maintain Complex XML Schemas** section of our **Developing XML Schemas** video demonstration:

https://www.youtube.com/embed/vz1eIZELQgc

Related Information:

Search and Refactor Operations Scope *(on page 318)*

Highlight Component Occurrences

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

This feature is on by default. To configured it, open the **Preferences** dialog box *(on page 36)* 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 *(on page 248)*.

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 currently 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 currently 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:

Go to 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:

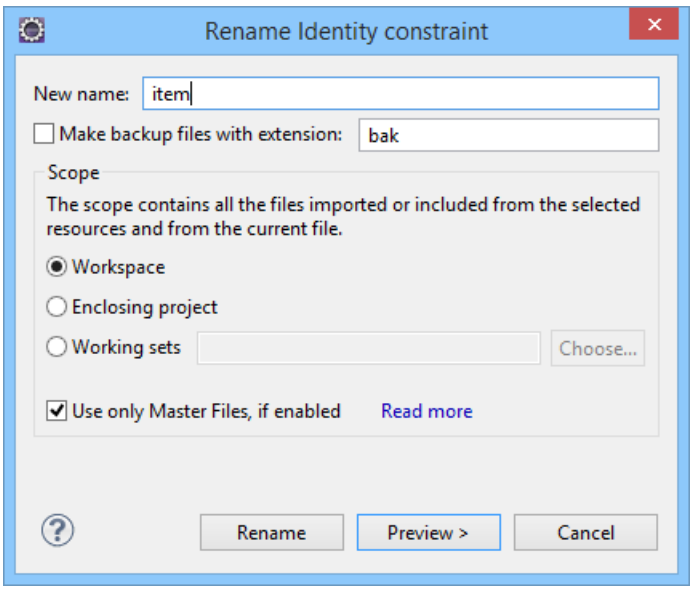
Rename Component

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 181. Rename Identity Constraint Dialog Box



Related Information:

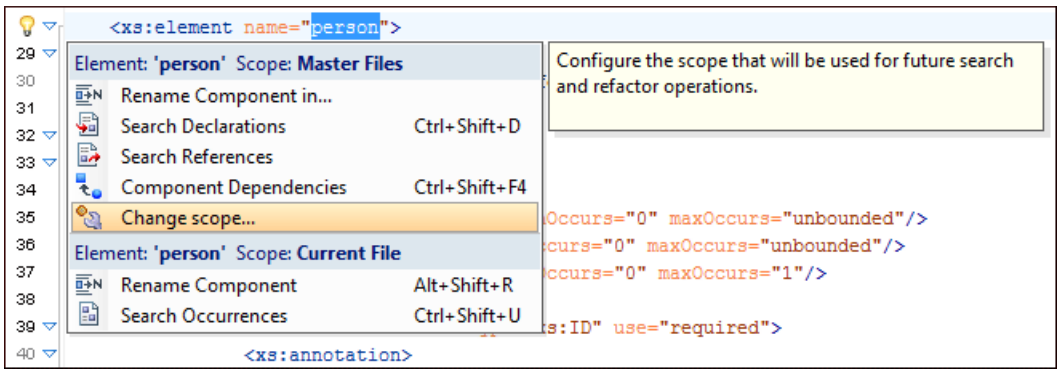
Search and Refactor Operations Scope *(on page 318)*

XML Schema Quick Assist Support

The *Quick Assist* support *(on page 1328)* improves the development work flow, offering fast access to the most commonly used actions when you edit schema documents.

The *Quick Assist* feature *(on page 1328)* 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 182. 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.

For more information about improving schema development using the **Quick Assist** action set, watch our video demonstration:

https://www.youtube.com/embed/X-2-gkrFSGU

Related Information:

Resource Hierarchy / Dependencies View *(on page 454)*

Component Dependencies View *(on page 456)*

Searching and Refactoring Actions *(on page 458)*

Generating Sample XML Files

Oxygen XML Developer Eclipse 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 *(on page 126)*.

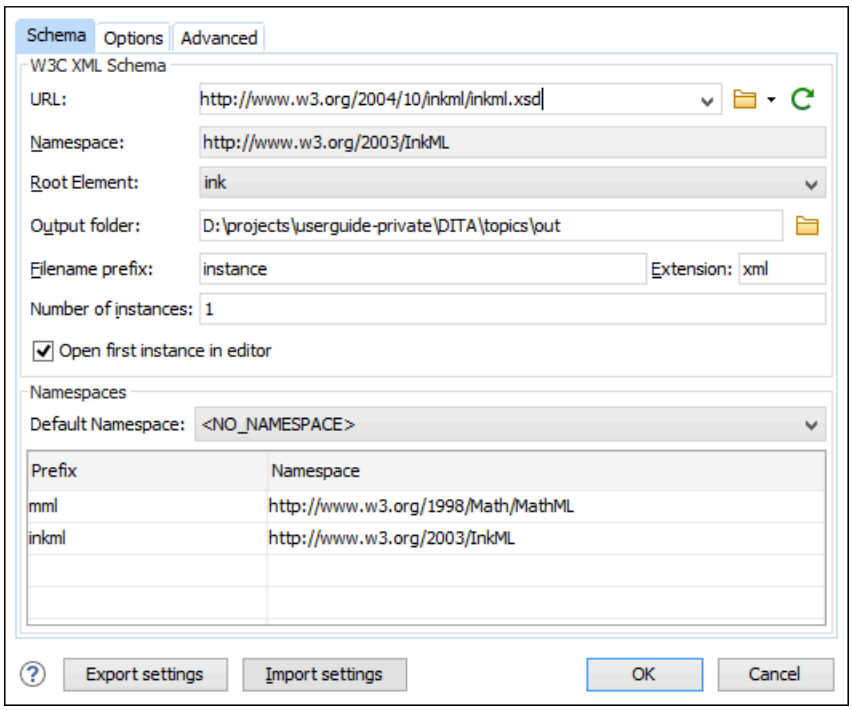
To generate sample XML files from an XML Schema, use the **Generate Sample XML Files** action from the **XML Tools** menu. This action is also available in the contextual menu of the schema Design mode *(on page 410)*. The action opens the **Generate Sample XML Files** dialog box that allows you to configure a variety of options for generating the files.

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 *(on page 119)*.

Schema Tab

The first set of options for the **Generate Sample XML Files** tool are found in the **Schema** tab.

Figure 183. 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 actions 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 selected, 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.

Export settings

Use this button to save the current settings for future use.

Import settings

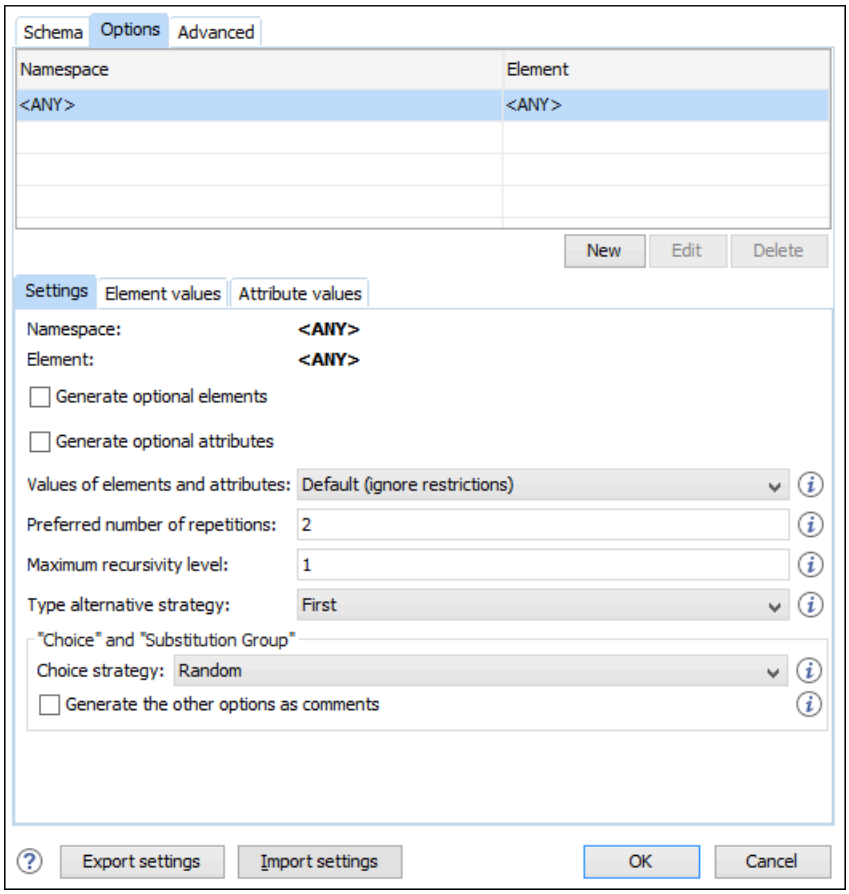
Use this button to load previously exported settings.

You can click **OK** at any point to generate the sample XML files.

Options Tab

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

Figure 184. 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 selected, all elements are generated, including the optional ones (having the `minOccurs` attribute set to 0 in the schema).

Generate optional attributes

When selected, 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 on the 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 **Sample XML Files Generator** preferences page). Note that type restrictions are ignored when this option is selected. 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 on the data type descriptor of the particular element or attribute.

  Important: If all of the following are true, the **Generate Sample XML Files** tool 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 selected, 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.

Export settings

Use this button to save the current settings for future use.

Import settings

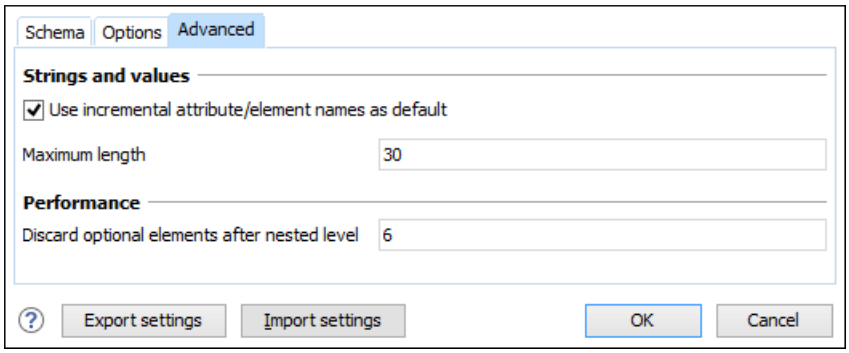
Use this button to load previously exported settings.

You can click **OK** at any point to generate the sample XML files.

Advanced Tab

The **Advanced** tab allows you to set some options regarding output values and performance.

Figure 185. Generate Sample XML Files Dialog Box (Advanced Tab)



This tab includes the following options:

Use incremental attribute / element names as default

If selected, 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 selected, 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.

Export settings

Use this button to save the current settings for future use.

Import settings

Use this button to load previously exported settings.

Tip: This function can be executed from an automated command-line script, but it requires an additional **Oxygen XML Scripting License**. For more details, see https://www.oxygenxml.com/oxygen_scripting.html.

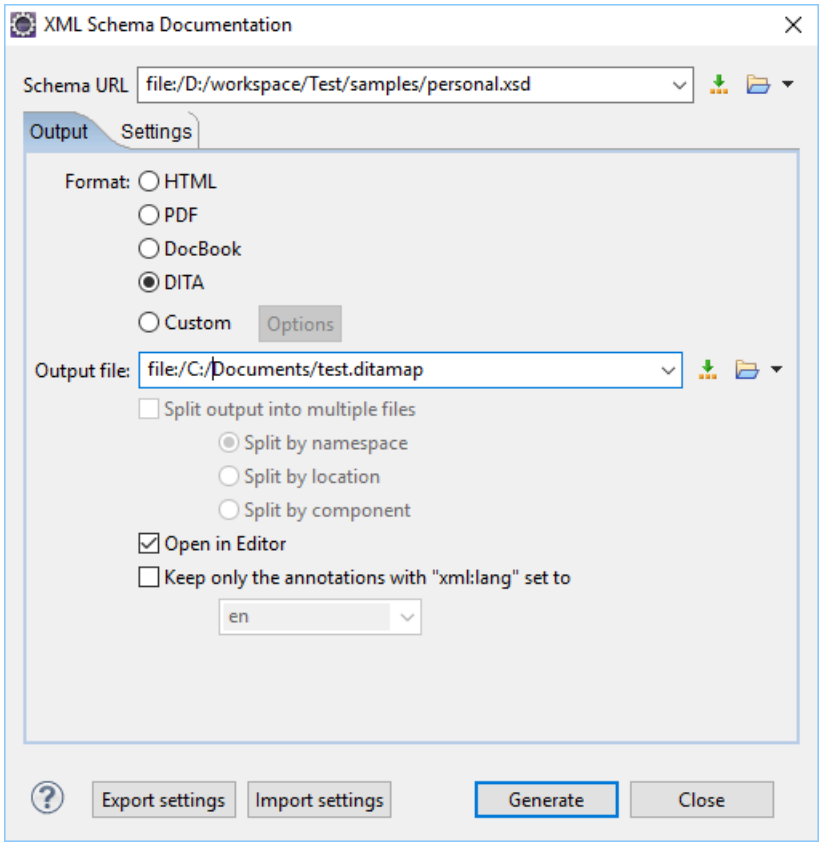
Generating Documentation for an XML Schema

Oxygen XML Developer Eclipse 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 *(on page 192)*.

Figure 186. XML Schema Documentation Dialog Box



The **Schema URL** field of the dialog box must contain the full path to the XML Schema (XSD) file that will have documentation generated. 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 *(on page 470)*.
  - **PDF** - The documentation is generated in PDF output format *(on page 473)*.
  - **DocBook** - The documentation is generated in DocBook output format *(on page 473)*.
  - **DITA** - The documentation is generated in DITA output format *(on page 473)*.
  - **Custom** - The documentation is generated in a custom output format *(on page 473)*, 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. For DITA and DocBook documents, this option appears as **Open in Editor** and the result will be opened in Oxygen XML Developer Eclipse plugin (in the current editor).

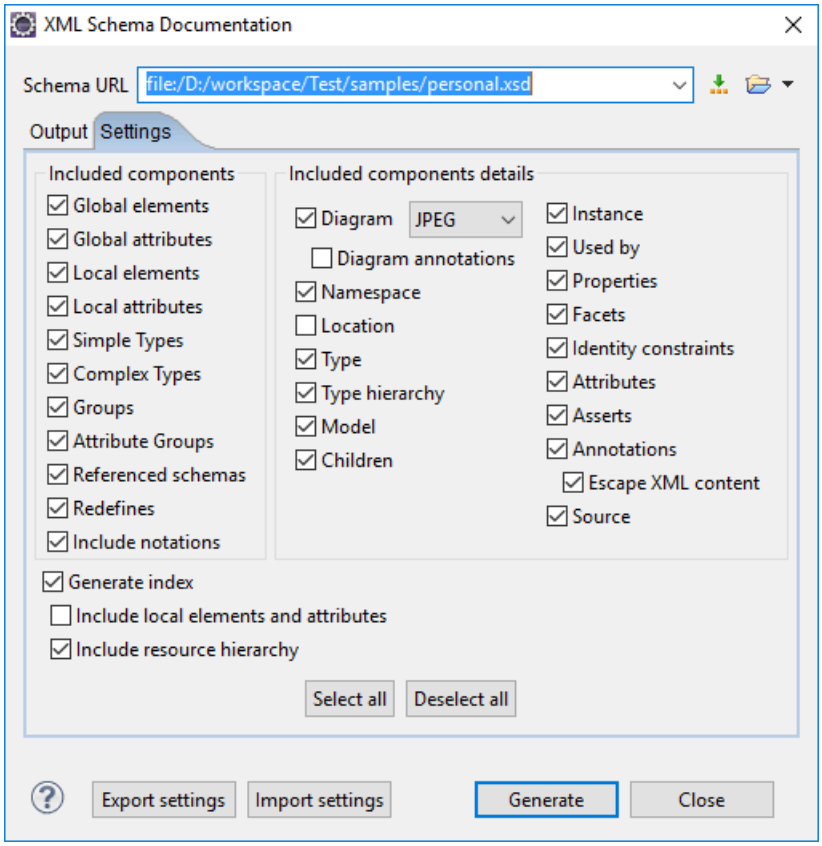
Note: To set the browser or system application that will be used, go to **Window > Preferences > General > Web Browser** and specify it there. This will take precedence over the default system application settings.

- **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 187. 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 *(on page 88)* 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 selected, 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 deselected by default.

Export settings - Save the current settings in a settings file for further use (for example, if you need the exported settings file for generating the documentation from the command-line interface).

Import settings - Reloads the settings from the exported file.

Generate - Use this button to generate the XML Schema documentation.

Tip: This function can be executed from an automated command-line script, but it requires an additional **Oxygen XML Scripting License**. For more details, see https://www.oxygenxml.com/oxygen_scripting.html.

Related Information:

Customizing PDF or DocBook Output of Generated XML Schema Documentation *(on page 474)*

Output Formats for Generating XML Schema Documentation

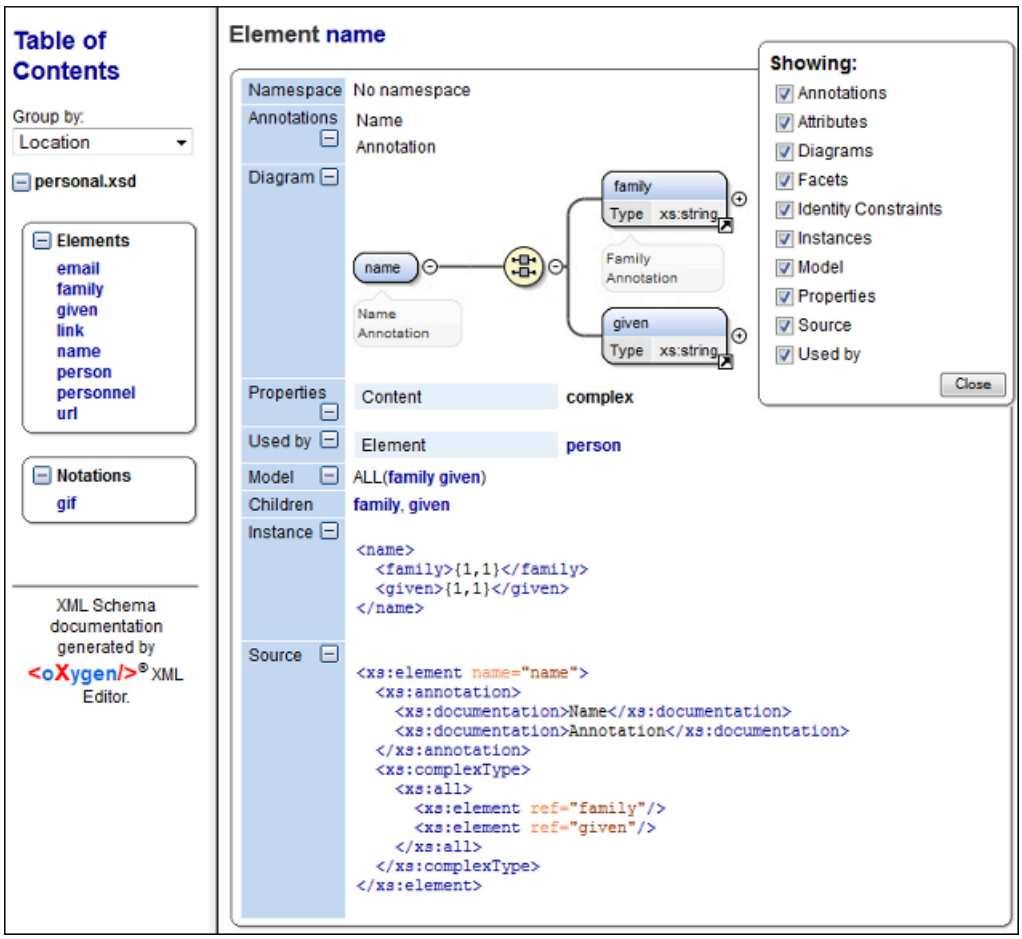
XML Schema documentation can be generated in HTML, PDF, DocBook, or a custom format. You can choose the format from the Schema Documentation *(on page 466)* dialog box. For the PDF and DocBook formats, the option to split the output in multiple files is not available.

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. *(on page 169)* 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 Eclipse plugin XSD editor panel. *(on page 170)*

Figure 188. XML Schema Documentation Example



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

Figure 189. Information About a Schema

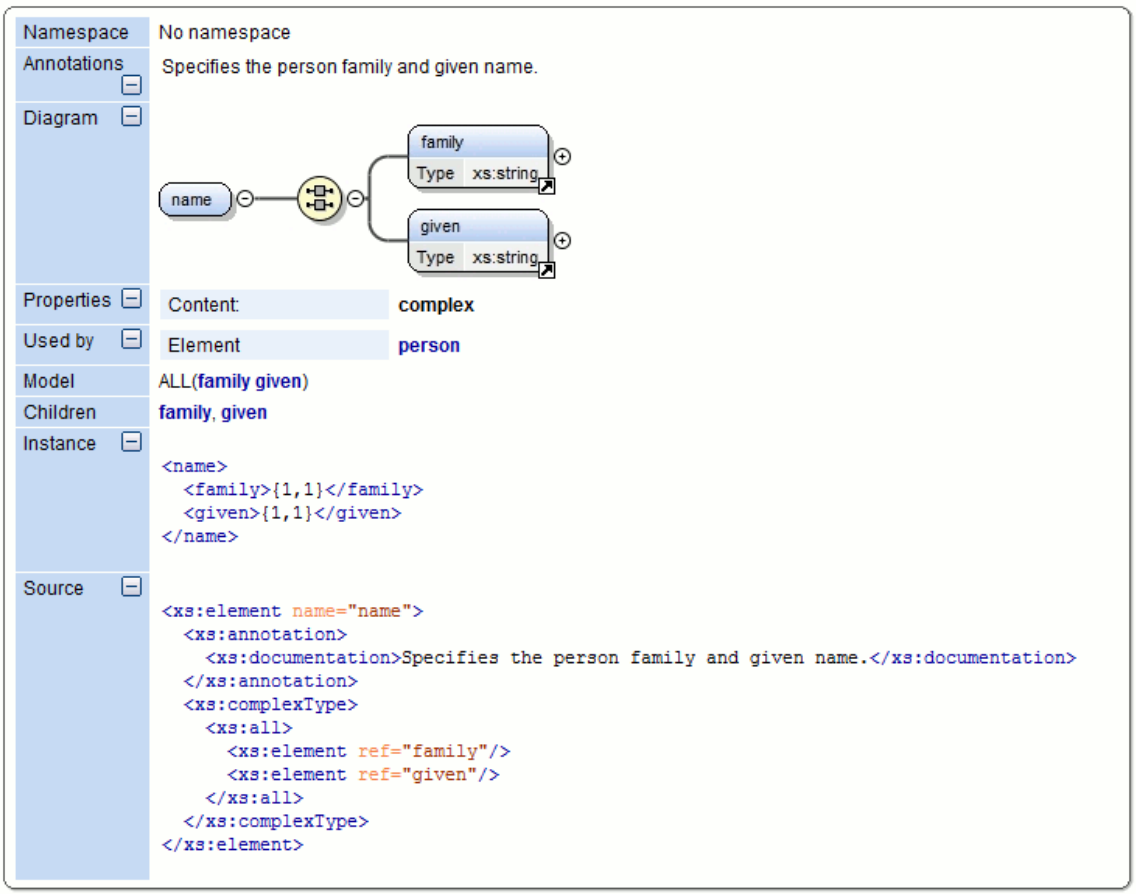
If you choose to split the output into multiple files, the table of contents is displayed in the left frame. The contents are grouped in the same mode. If you split the output by location, each file contains a schema description and the components that you have chosen to include. If you split the output 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 190. Showing Options

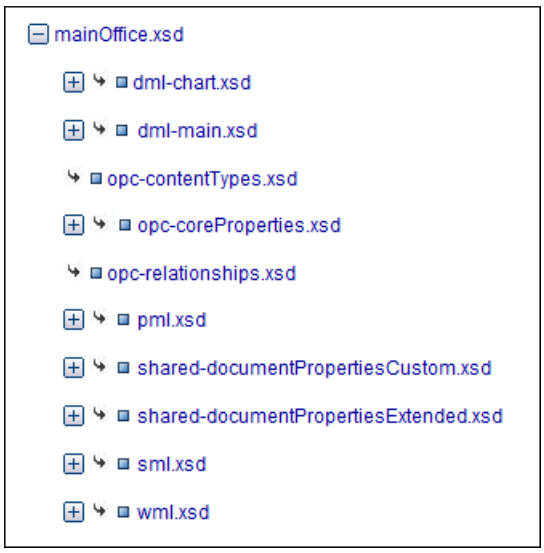
For each component included in the documentation, the section presents the component type follow by the component name. For local elements and attributes, the name of the component is specified as *parent name/component name*. You can easily go to the parent documentation by clicking the parent name.

Figure 191. Documentation for a Schema Component



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

Figure 192. 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 *(on page 115)* preferences page.

For information about customizing the PDF output, see Customizing PDF or DocBook Output of Generated XML Schema Documentation *(on page 474)*.

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 built-in DocBook transformation scenario *(on page 686)* (such as, *DocBook PDF* or *DocBook HTML*) on the output file, or configure your own transformation scenario *(on page 691)* to convert it into whatever format you desire.

For information about customizing the DocBook output, see Customizing PDF or DocBook Output of Generated XML Schema Documentation *(on page 474)*.

DITA Output Format

If you generate the documentation in DITA output format, each element of the schema is converted to a DITA *Topic* and all the generated topics are referenced in a *DITA map (on page 1324)* file. You can then apply a built-in DITA transformation scenario (such as, *DITA Map PDF* or *DITA Map XHTML*), or configure your own DITA-OT transformation scenario *(on page 716)* to convert it into whatever format you desire.

For information about customizing the DITA output, see Customizing DITA Output of Generated XML Schema *(on page 475)*.

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 built-in HTML, PDF, DocBook, and DITA 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.

Important: If you create a custom format for DITA, you must select the **Split output into multiple files** option in the **Output** tab *(on page 468)* and choose **Split by component**.

Customizing PDF or DocBook Output of Generated XML Schema Documentation

To customize the PDF or DocBook 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 or DocBook 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"/>
```

2. Create an intermediary file that holds the content of your XML Schema documentation by following these steps:
    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:
        i. Enter the customized stylesheet in the **Custom XSL** field (*[OXYGEN_INSTALL_DIR]*`/frameworks/schema_documentation/xsl/xsdDocDocbook.xsl`).
        ii. Select the **Copy additional resources to the output folder** option and leave the default selection in the **Resources** field.
        iii. Click **OK**.
    d. When you return to the **XML Schema Documentation** dialog box, just click the **Generate** button to generate a DocBook XML file with an intermediary form of the Schema documentation.
3. If you want the DocBook file to be transformed into a PDF document, follow these steps:
    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 select 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**.

Customizing DITA Output of Generated XML Schema

To customize the DITA output of the generated XML Schema documentation, use the following procedure:

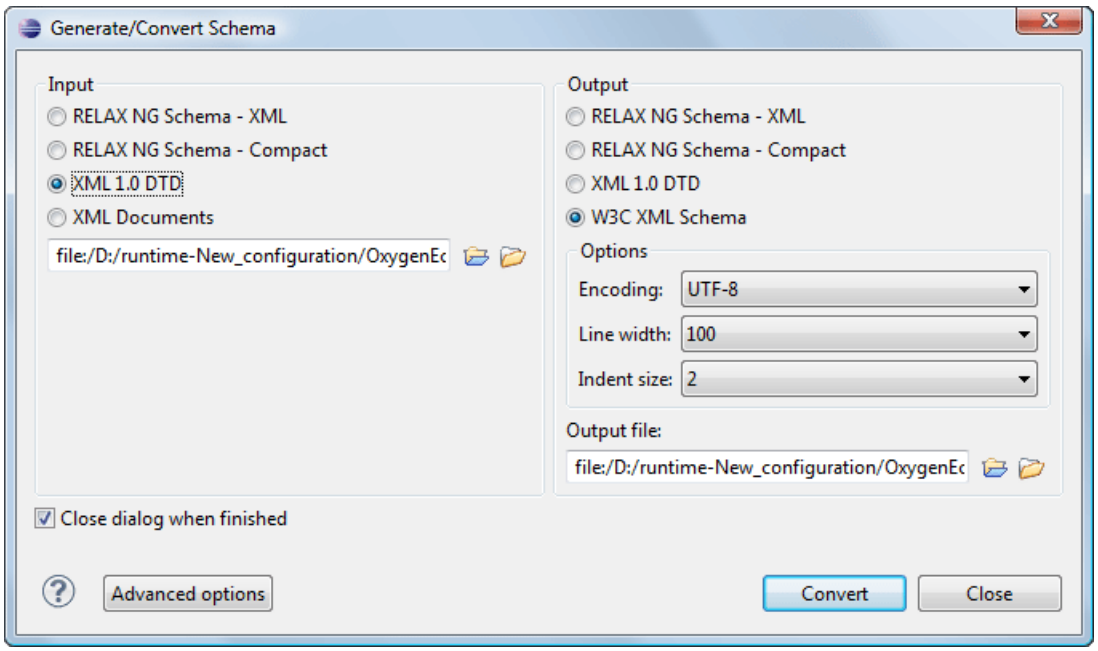
1. Customize the `[OXYGEN_INSTALL_DIR]/frameworks/schema_documentation/xsl/xsdDocDita.xsl` stylesheet to incorporate your desired changes.
2. Create an intermediary file that holds the content of your XML Schema documentation by following these steps:
   a. Go to **Tools > Generate Documentation > XML 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:
      i. Enter the customized stylesheet in the **Custom XSL** field (`[OXYGEN_INSTALL_DIR]/frameworks/schema_documentation/xsl/xsdDocDita.xsl`).
      ii. Select the **Copy additional resources to the output folder** option and leave the default selection in the **Resources** field.
      iii. Click **OK**.
   d. Make sure the **Split output into multiple files** option *(on page 468)* is selected and choose **Split by component**.
   e. When you return to the **XML Schema Documentation** dialog box, just click the **Generate** button to generate a DITA map file that contains the XML Schema documentation.

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 Eclipse plugin generates an approximation of the source schema. Oxygen XML Developer Eclipse plugin uses the *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 193. 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 open 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 a `@prefix:defaultValue` annotation attribute 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 Eclipse 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 *(on page 40)* 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**.

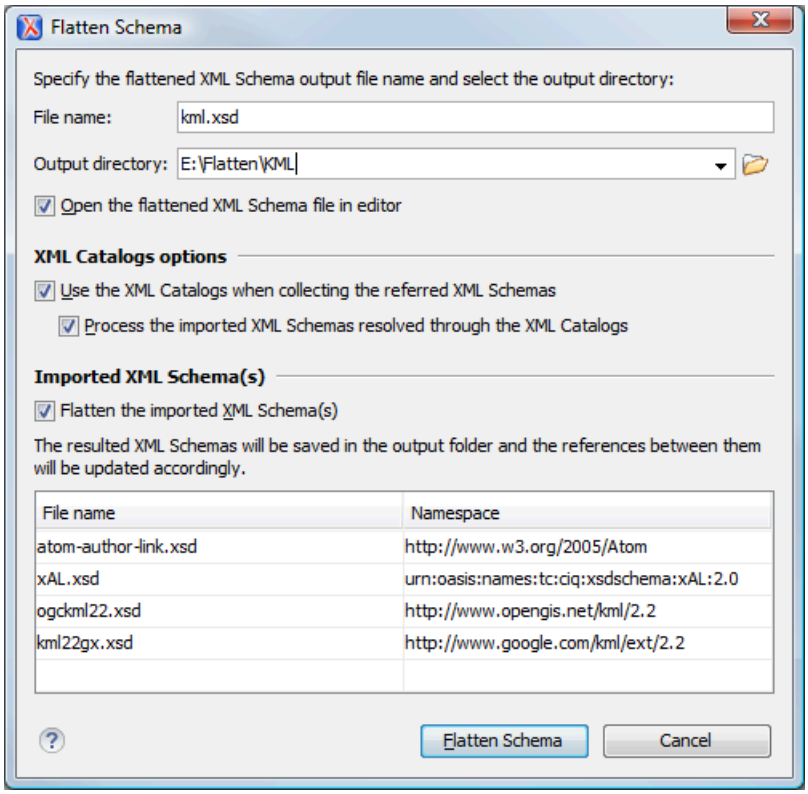
Result: The database structure is converted to an XML Schema and it is opened for viewing and editing.

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 Eclipse 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 194. 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 Eclipse 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 (on page 1329)*.

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 (on page 1329)* are also processed.

Flatten the imported XML Schema(s)

Specifies whether or not the imported schemas are flattened.

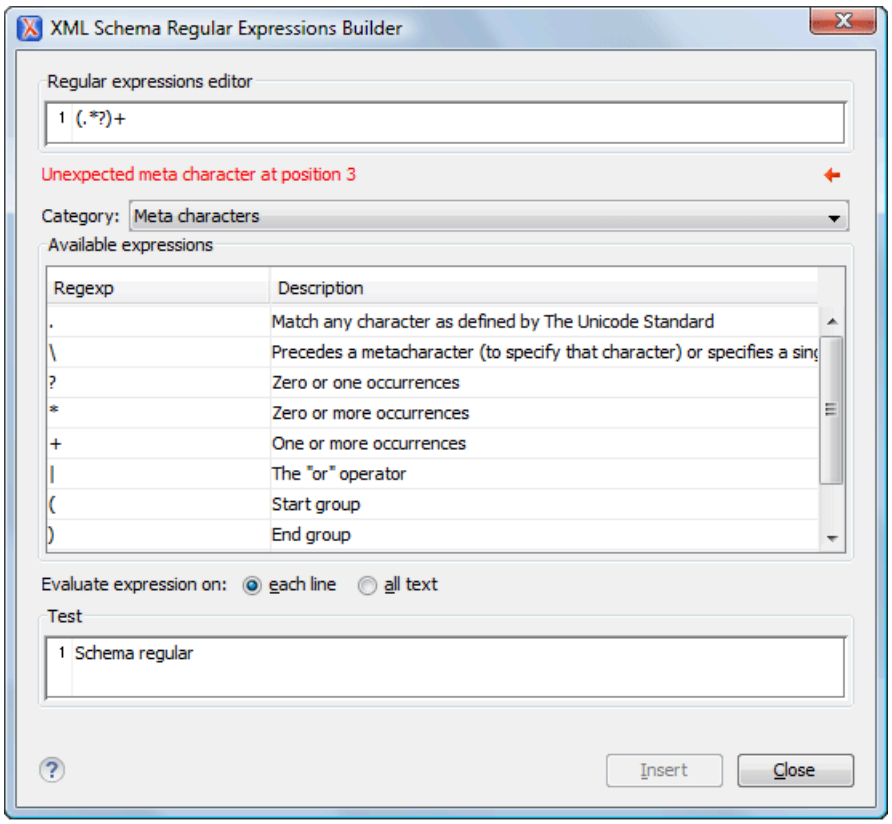
Note: For the schemas skipped by the flatten operation, no files are created in the output folder and the corresponding import statements remain unchanged.

Tip: This function can be executed from an automated command-line script, but it requires an additional **Oxygen XML Scripting License**. For more details, see https://www.oxygenxml.com/oxygen_scripting.html.

XML Schema Regular Expressions Builder Tool

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 195. 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 that will have the regular expression 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 *(on page 168)* or schema **Design** mode *(on page 169)*. Accordingly, the **Insert** button will be not available 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 Eclipse plugin offers full support for XML Schema 1.1, including:

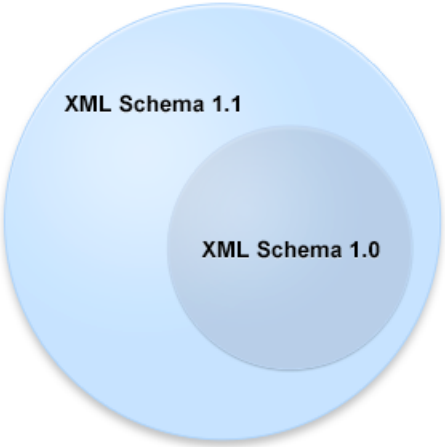
- XML Documents Validation *(on page 277)* and Content Completion *(on page 234)* based on XML Schema 1.1.
- XML Schema 1.1 Validation *(on page 447)* and Content Completion *(on page 448)*.
- Editing XML Schema 1.1 files in the Schema **Design** mode *(on page 169)*.
- The Flatten Schema *(on page 478)* action.
- Resource Hierarchy/Dependencies *(on page 454)* and Refactoring Actions *(on page 458)*.
- *Master files (on page 1326)*.
- Generating Documentation for XML Schema 1.1 *(on page 466)*.
- 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 196. 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 (on page 1324)* presents XML Schema 1.1 elements that you can insert accidentally in an 1.0 XML Schema. So even if you make a document invalid conforming with XML Schema 1.0, the validation process does not signal any issues.

To change the default XML Schema version, open the **Preferences** dialog box *(on page 36)* and go to **XML > XML Parser > XML Schema**.

For more information about the XML Schema 1.1 support, watch our video demonstration:

https://www.youtube.com/embed/DAkrubQNm0w

Related Information:

Setting the XML Schema Version *(on page 484)*

Setting the XML Schema Version

Oxygen XML Developer Eclipse 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
minVersion = "1.0" **maxVersion** = "1.1"	1.0
minVersion = "1.1"	1.1
minVersion = "1.0" **maxVersion** = greater than "1.1"	The XML Schema version defined in the **XML Schema** preferences page *(on page 126)*
Not set in the XML Schema document	The XML Schema version defined in the **XML Schema** preferences page *(on page 126)*

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 Eclipse plugin to its default options.

Oxygen XML Developer Eclipse 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 Eclipse plugin takes into account the *minVersion* and *maxVersion* attributes defined in the XML Schema.

Related Information:

XML Schema 1.1 *(on page 482)*

Editing XQuery Documents

XQuery is the query language for XML and is officially defined by a W3C Recommendation document. Oxygen XML Developer Eclipse plugin provides support for XQuery 3.1, which is also backwards compatible with XQuery 3.0 and 1.0.

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.

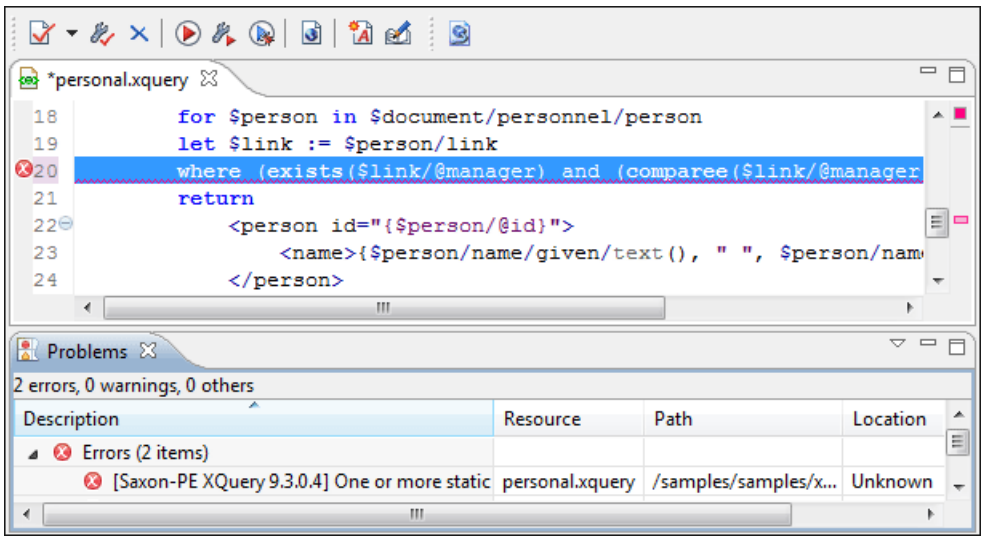
Related Information:

XQuery and Databases *(on page 1186)*

XQuery Validation

With Oxygen XML Developer Eclipse plugin, you can validate your documents before using them in your transformation scenarios. The validation uses the Saxon 9.9.1.5 PE, EE, or HE processor, or you can use some database engines (such as MarkLogic or eXist) if you installed them. Any other XQuery processor that offers an XQJ API implementation *(on page 1180)* 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 197. XQuery Validation



Note: If you choose a processor that does not support XQuery validation, Oxygen XML Developer Eclipse 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 Eclipse plugin uses this connection for validation. If you open an XQuery file using a MarkLogic connection, the validation resolves imports better.

Content Completion in XQuery

Oxygen XML Developer Eclipse plugin provides content completion for keywords and all known XQuery functions and operators. The *Content Completion Assistant (on page 1324)* can be manually activated with the **(Ctrl + Space)** shortcut. The functions and operators are presented together with a description of the parameters and functionality, depending on the validation or transformation engine.

For some supported database engines such as MarkLogic, 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 *(on page 134)* page.
For more information about the support for working with XQuery with regard to databases, see XQuery and Databases *(on page 1186)*.

The extension functions included in the Saxon engine are available on content completion if one of the following conditions are true:

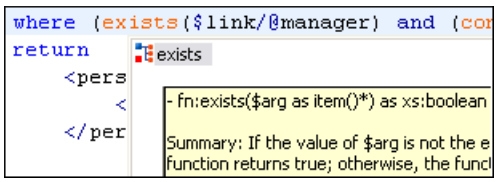
- The edited file has a transformation scenario associated that uses as transformation engine Saxon 9.9.1.5 PE or Saxon 9.9.1.5 EE.
- The edited file has a validation scenario associated that use as validation engine Saxon 9.9.1.5 PE or Saxon 9.9.1.5 EE.
- The validation engine specified in Preferences *(on page 134)* is Saxon 9.9.1.5 PE or Saxon 9.9.1.5 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 198. XQuery Content Completion



Syntax Highlighting in XQuery

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for XQuery files, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
3. Select and expand the **XQuery/XPath** 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:

Customize Syntax Highlight colors *(on page 101)*

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 Eclipse 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.

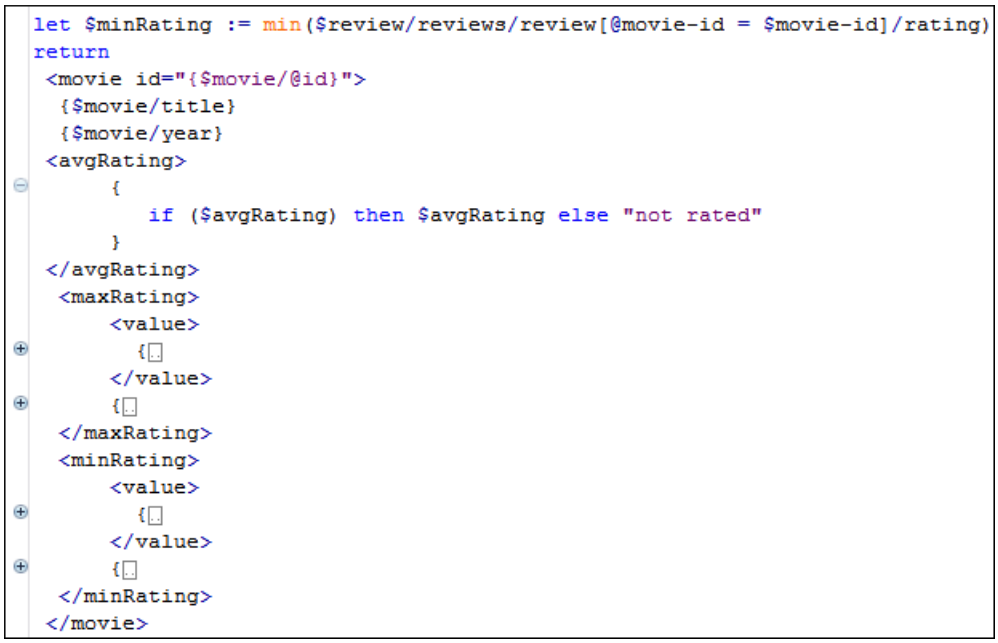
- Formats complex expressions on multiple, more readable lines by properly indenting each of them. The amount of whitespaces that form an indent unit is controlled through one of the **Indent with tabs** and **Indent size** options from the **Format Preferences** page *(on page 89)*.

Note: These operations can be performed only if your XQuery document conforms with XQuery 1.0, 3.0, 3.1, or XQuery Update Facility 1.0 specifications. If the *Format and Indent* operation fails, the document is left unaltered and an error message is presented in the **Results** view *(on page 248)*.

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 *(on page 231)* are also available in XQuery documents.

Figure 199. Folding in XQuery Documents

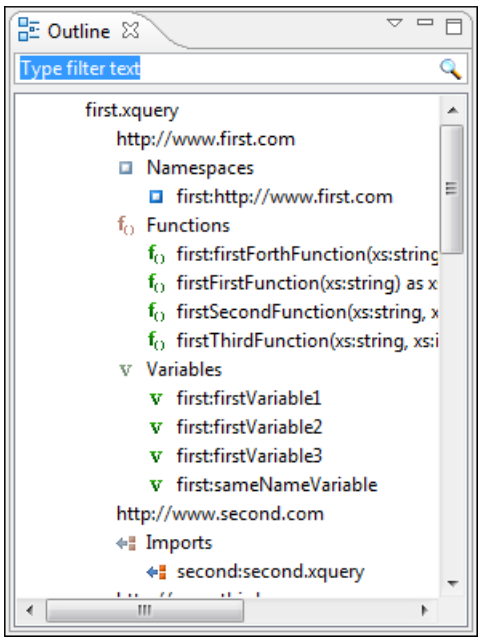


There is available the action **Go to Matching Bracket** **Ctrl + Shift + G (Command + Shift + G on OS X)** on contextual menu of XQuery editor for going to matching character when cursor is located at '{' character or '}' character. It helps for finding quickly matching character of current *folding element (on page 1325)*.

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 by selecting it from the **Window > Show View** menu.

Figure 200. 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 (such as `*` or `?`) and separate multiple patterns with commas.

XQuery Builder View

The **XPath/XQuery Builder** view allows you to compose complex XQuery expressions and execute them over the currently edited XML document. You can use the `doc()` function to specify the source file that will have the expressions 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 Eclipse plugin executes the expressions over the XML document in the associated scenario.

If the view is not displayed, it can be opened by selecting it 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 2.0 SA, XPath 3.1, XPath 3.1 SA, 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 3.1 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 Eclipse plugin uses Saxon to execute XPath 3.1 expressions. Since Saxon implements a part of the 3.1 functions, when using a function that is not implemented, Oxygen XML Developer Eclipse 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 *(on page 248)*.

Favorites button

Allows you to save certain expressions that you can later reuse. To add an expression as a favorite, click this 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 Eclipse 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 selected and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed. For JSON documents, it displays the XPath expression for the current property.

Evaluate as you type

When you select this option, the XPath expression you are composing is evaluated in real time.

Note: This 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 Eclipse plugin allows you to define a scope for the XPath operation to be executed. You can choose where the XPath expression will be executed:

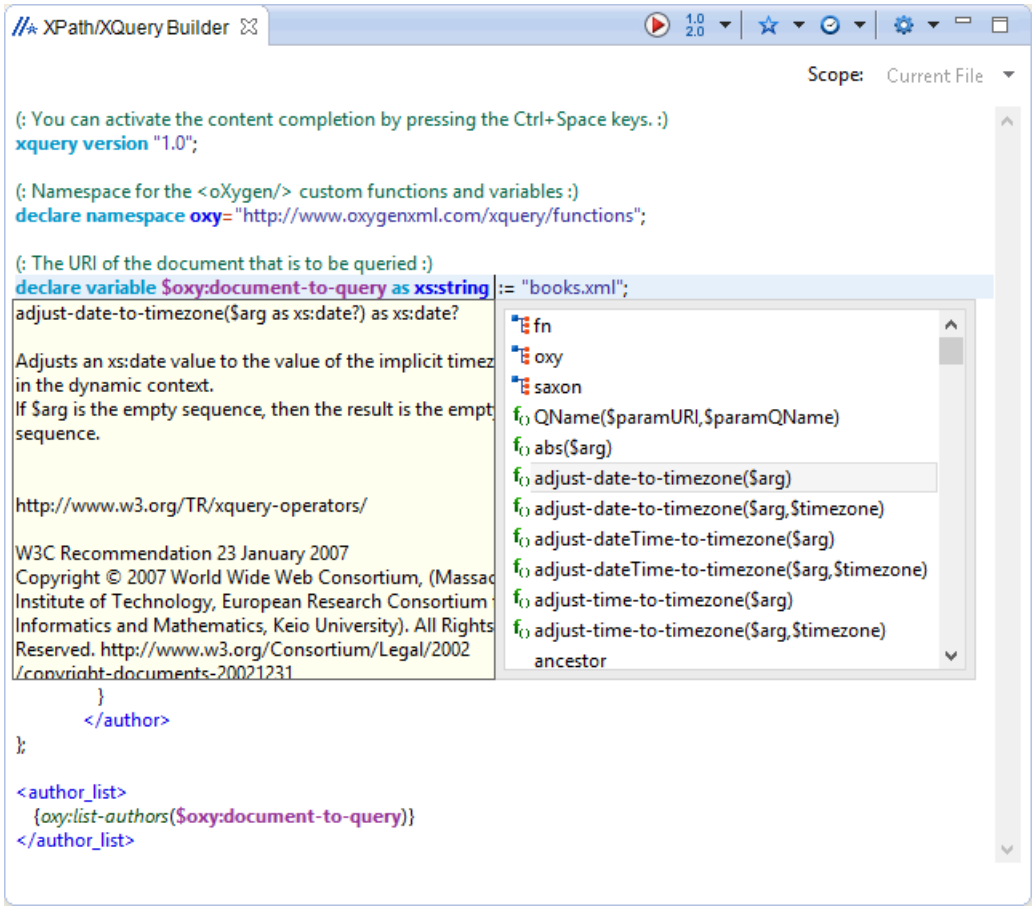
- **Current file** - Currently selected file only.
- **Enclosing project** - All the files of the project that encloses the currently edited file.
- **Workspace selected files** - The files selected in the workspace. The files are collected from the last selected resource provider view (**Navigator** *(on page 192)*, **Project Explorer** or **Package Explorer**).
- **All opened files** - All files that are opened in the application.
- **Opened archive** - Files that are opened in the **Archive Browser** view *(on page 1126)*.
- **Working sets** - The selected *working sets (on page 1329)*.

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 (on page 1329)*.

- **XPath file filter** - You can filter the files from the selected scope that will have the XPath expression executed. By default, the XPath expression will be executed only on XML or JSON files, but you can also define a set of patterns that will filter out files from the current scope.

Figure 201. XPath/XQuery Builder View



When you hover your cursor over the version icon (1.0 2.0), a tooltip is displayed to let you know what engine Oxygen XML Developer Eclipse plugin is currently using.

While you edit an XPath or XQuery expression, Oxygen XML Developer Eclipse plugin assists you with the following features:

- *Content Completion Assistant (on page 1324)* - 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 Highlighting - Allows you to identify the components of an expression. To customize the colors of the components of the expression, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Syntax Highlight** *(on page 101)*.
- Automatic validation of the expression as you type.

Note: When you type invalid syntax, a red serrated line underlines the invalid fragments.

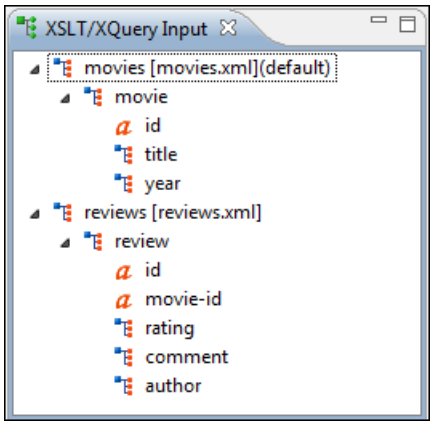
- Function signature and documentation balloon, when the cursor is located inside a function.

XQuery Input View

The structure of the source documents of an edited XQuery is displayed in a tree form in a view called the **XQuery Input** view. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu. The tree nodes represent the elements of the documents.

You can use the **XQuery Input** view to drag and drop a node into the editing area to quickly insert XQuery expressions.

Figure 202. XQuery Input View



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">
```

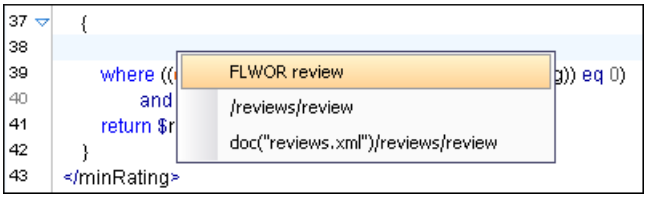
```
<rating>3</rating>
<comment>Tom Hanks does a really nice acting.</comment>
<author>Beatrice</author>
</review>
<review id="104" movie-id="2">
<rating>4</rating>
<comment>Robert De Niro is my favorite actor.</comment>
<author>Maria</author>
</review>
</reviews>
```

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:



Select **FLWOR review**, the resulting document will look like this:

```
{
    for $review in doc("reviews.xml")/reviews/review
    return
    where ((compare($rev/rating/text(), string($minRating)) eq 0)
            and ($rev/@movie-id = $movie/@id))
    return $rev/author
}
```

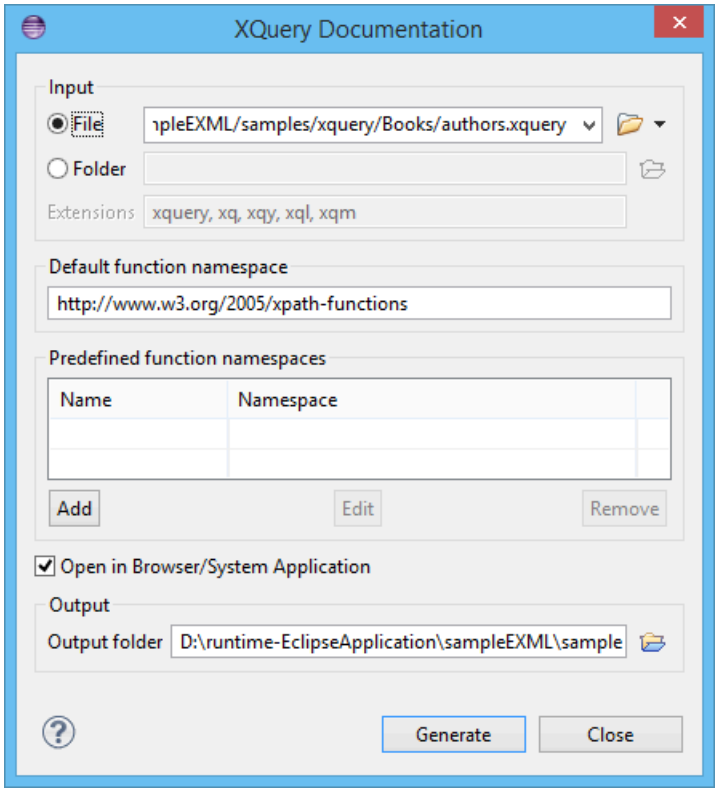
Generating HTML Documentation for an XQuery Document

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 *(on page 192)*.

The dialog box allows you to configure a set of parameters for the process of generating the HTML documentation.

Figure 203. 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 to be used for generating the documentation.
  - **Folder** - The directory that contains the files to be used for generating 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.

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 *(on page 691)*. The result can be saved and opened in the associated application. You can even run a FO processor *(on page 746)* on the output of an XQuery. The transformation scenarios may be shared between many XQuery files, are exported *(on page 147)* together with the XSLT scenarios and can be managed in the **Configure Transformation Scenario** dialog box *(on page 765)*, or in the **Scenarios** view *(on page 770)*. 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 *(on page 691)*. 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 9.9.1.5 HE, Saxon 9.9.1.5 PE, Saxon 9.9.1.5 EE processors, a database connection (details can be found in the Working with Databases *(on page 1131)* chapter - in the XQuery transformation *(on page 1187)* section) or any XQuery processor that provides an XQJ API implementation.

The Saxon 9.9.1.5 EE processor also supports XQuery 3.1 transformations.

Related Information:

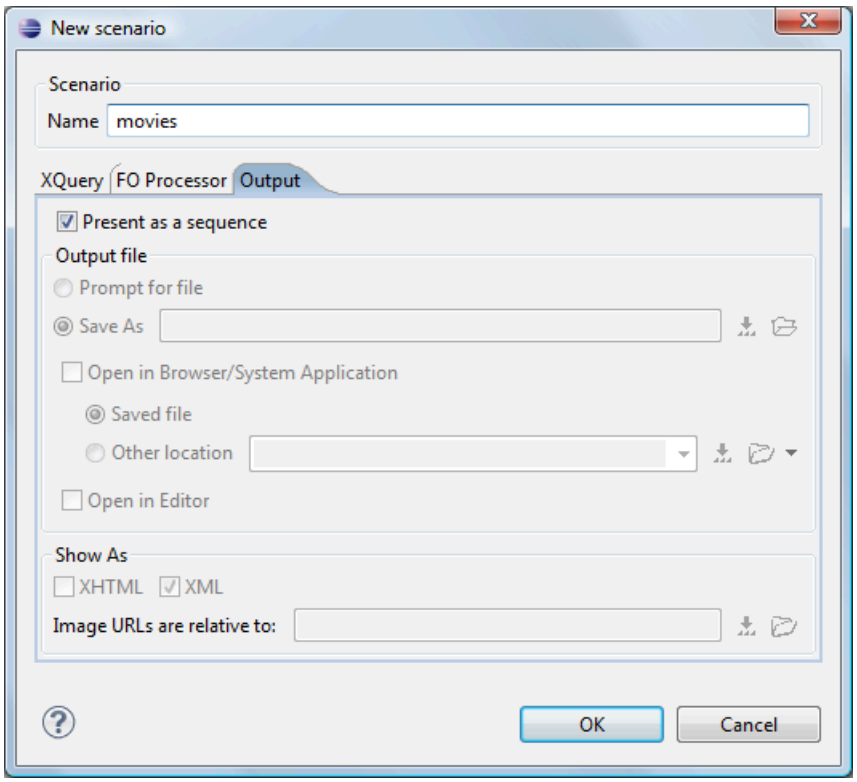
XQuery and Databases *(on page 1186)*

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 *(on page 712)* 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.

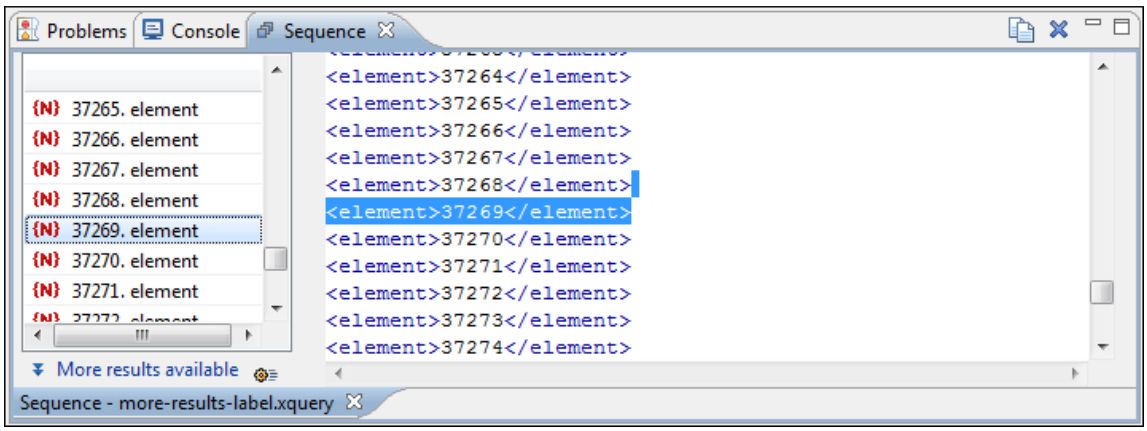
The size of a chunk can be set with the **Size limit of Sequence view** option *(on page 134)*. The **XQuery options** button from the **More results available** label provides a quick access to this option by opening the **XQuery** preferences page *(on page 134)* where the option can be modified.

Figure 204. XQuery transformation result displayed in Sequence view



A chunk of the XQuery transformation result is displayed in the **Sequence** view.

Figure 205. XQuery transformation result displayed in Sequence view



Tip: You can right-click the results in the **Sequence** view and if the item is an XML element, the **Go to definition** action will open the XML file from where the queried node was obtained.

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 *(on page 134)* 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.

Saxon-HE/PE/EE Options

The advanced options for Saxon 9.9.1.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

Use a configuration file ("-config")

Sets a Saxon 9.9.1.5 configuration file that is used for XQuery transformation and validation scenarios.

Enable Optimizations ("-opt")

This option is selected by default, which means that optimization is enabled. If not selected, the optimization is suppressed, which is helpful 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.

Recoverable errors ("-warnings")

Specifies 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")

Specifies 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.

Saxon-PE/EE Options

The following advanced options are specific for Saxon 9.9.1.5 Professional Edition (PE) and Enterprise Edition (EE) only:

Allow calls on extension functions ("-ext")

If selected, calls on external functions are allowed. Selecting 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.

Saxon-EE Options

The advanced options that are specific for Saxon 9.9.1.5 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 allows for parsing the source documents with strict schema-validation enabled.
- **Lax schema validation ("lax")** - If an XML Schema is provided, this mode allows for 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. Selecting 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.

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 selected, backup versions for any XML files updated with an XQuery Update are generated. This option is available when the **Enable XQuery update** option is selected.

Other Options

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.

Updating XML Documents using XQuery Update 1.0

Using the bundled Saxon 9.9.1.5 EE XQuery processor Oxygen XML Developer Eclipse 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 9.9.1.5 EE as a transformer in the scenario associated with the XQuery files containing update statements and Oxygen XML Developer Eclipse plugin will notify you if the update was successful.

Example: Using XQuery Update to modify a tag name in an XML file

```
rename node doc("books.xml")//publisher[1]//book[1] as "firstBook"
```

Editing WSDL Documents

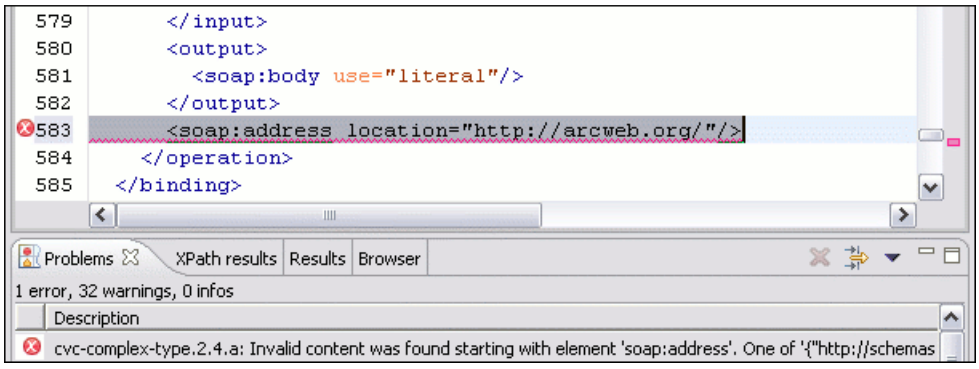
WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

Oxygen XML Developer Eclipse plugin provides a special type of editor dedicated to WSDL documents. The WSDL editor offers support for validation, a specialized *Content Completion Assistant (on page 1324)*, a component oriented **Outline** view *(on page 503)*, 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 Eclipse 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 206. Validating a WSDL file



For more information about the WSDL editing support in Oxygen XML Developer Eclipse plugin, watch our video demonstration:

https://www.youtube.com/embed/OS5Ucm9b8sY

Related Information:

Editing XML Documents in Text Mode *(on page 222)*

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 Eclipse plugin provides the support for defining the main module (or modules), allowing you to edit any of the imported/included files in the context of the larger WSDL structure.

You cat set a main WSDL document either using the *master files* support from the **Navigator** view *(on page 200)*, or using a validation scenario.

To set a *master file* using a validation scenario, add validation units that point to the main modules. Oxygen XML Developer Eclipse 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 (on page 1326)* include:

- Correct validation of a module in the context of a larger WSDL structure.
- *Content Completion Assistant (on page 1324)* displays all components valid in the current context.
- The **Outline** view *(on page 503)* 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.

For more information about editing WSDL documents in the *master files* context, watch our video demonstration:

https://www.youtube.com/embed/gn_YPD5xDCo

Validating WSDL Documents

By default, WSDL files are validated as you type. To change this, open the **Preferences** dialog box *(on page 36)*, go to **Editor > Document Checking**, and deselect the **Enable automatic validation** option *(on page 84)*.

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.

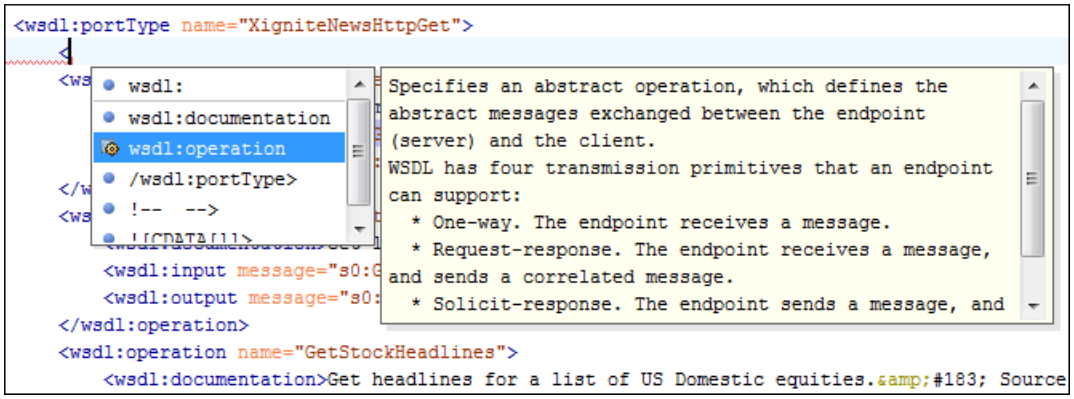
Content Completion Assistance in WSDL Documents

The *Content Completion Assistant (on page 1324)* is a powerful feature that enhances the editing of WSDL documents. It helps you define WSDL components by proposing context-sensitive element names. It can be manually activated with the **Ctrl + Space (Command + Space on OS X)** shortcut.

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.

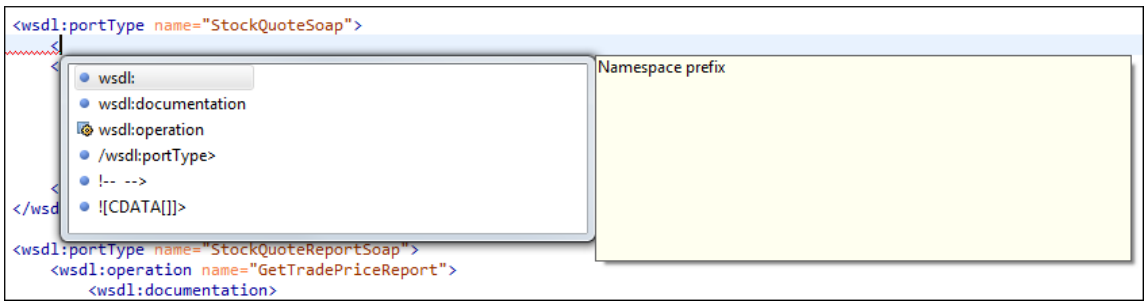
Figure 207. WSDL Content Completion Assistant



Note: If you are using the concept of *master files (on page 1326)* to import/include modules, 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 more information about the *Master Files* support in Oxygen XML Developer Eclipse plugin, see Defining Master Files at Project Level *(on page 200)*.

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 208. 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 Eclipse plugin provides an easy mode to declare them by proposing a prefix for these namespaces.

WSDL Syntax Highlighting

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for WSDL files, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
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 **XML** tab in the **Preview** pane to see the effects of your changes.

Tip: Oxygen XML Developer Eclipse plugin also allows you to specify syntax highlighting colors for specific XML elements and attributes with specific namespace prefixes. This can be done in the **Editor > Syntax Highlight > Elements/Attributes by Prefix** preferences page *(on page 101)*.

Related Information:

Customize Syntax Highlight colors *(on page 101)*

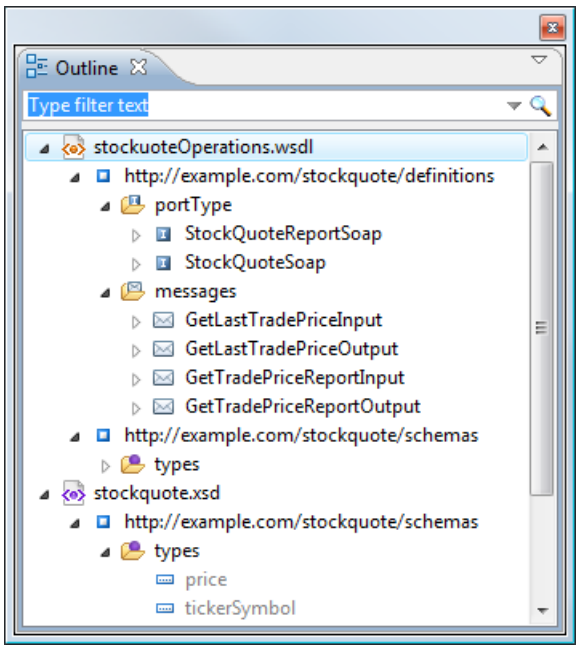
WSDL Outline View

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 *(on page 200)*, 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 by selecting it from the **Window > Show View** menu.

Figure 209. 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 *(on page 144)*.

Configure displayed attributes

Displays the **XML Structured Outline** preferences page *(on page 144)*.

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

Opens the **Component Dependencies** view *(on page 510)* that displays the dependencies of the currently selected component.

Show Resource Hierarchy

Opens the **Resource Hierarchy/Dependencies** view *(on page 507)* that displays the hierarchy for the currently selected resource.

Show Resource Dependencies

Opens the **Resource Hierarchy/Dependencies** view *(on page 507)* that 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

Opens the **Component Dependencies** view *(on page 510)* that 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.

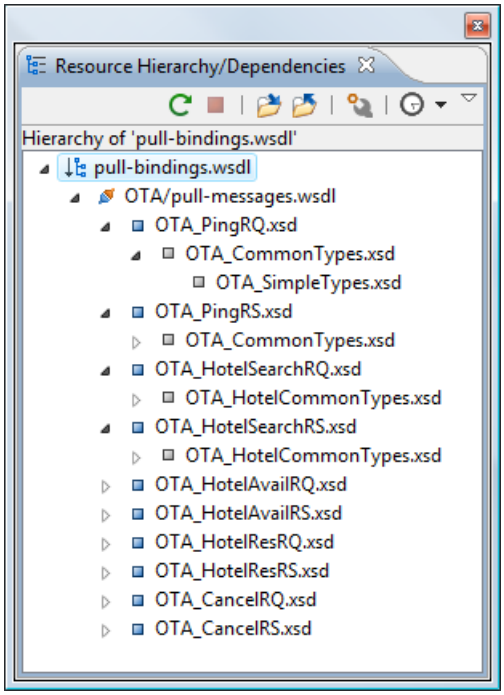
WSDL Resource Hierarchy/Dependencies View in WSDL Documents

The **Resource Hierarchy/Dependencies** view displays the hierarchy or dependencies for a WSDL resource. If the view is not displayed, it can be opened by selecting it 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 or dependencies of a WSDL document, select the document in the **Navigator** view *(on page 192)* and choose **Show Resource Hierarchy** or **Show Resource Dependencies** from the contextual menu.

Figure 210. Resource Hierarchy/Dependencies View



The following actions are available on the toolbar of 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 dependencies search scope

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 for a resource listed in the **Resource Hierarchy/Dependencies** view contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Go to reference

Opens the source document where the resource is referenced.

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 *(on page 200)*.

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 *(on page 509)*.

Related Information:

Search and Refactor Operations Scope *(on page 318)*

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 of the renamed resource(s)** - Select this option to update the references to the resource you are renaming. A **Preview** option is available that allows you to see what will be updated before selecting **Rename** to process the operation.

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)** - Select this option to update the references to the resource you are moving, in accordance with the new location and name. A **Preview** option is available that allows you to see what will be updated before selecting **Move** to process the operation.

WSDL Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected component. This is helpful if you want to see where components are used in the entire hierarchy. For example, if you want to find all the references where a given component is used.

If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

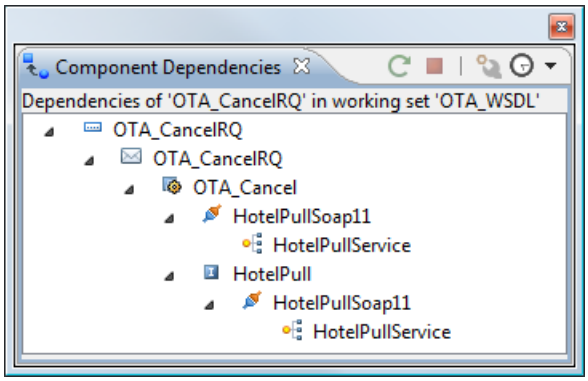
To see the dependencies of a WSDL component:

1. Right-click the desired component in the editor or **Outline** view.
2. Select the **Component Dependencies** action from the contextual menu.

This action is available for all WSDL components (`messages`, `port types`, `operations`, `bindings`, and so on).

If a component contains multiple references, a small table is displayed at the bottom of the view that contains all the references. When a recursive reference is encountered, it is marked with a special icon .

Figure 211. Component Dependencies View



The **Component Dependencies** view includes the following toolbar actions:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependency computation.

Configure

Allows you to choose the search scope for computing the dependencies structure. This is helpful for making sure all imported/included resources are computed.

History

Allows you to select from a list of the most recently used dependency computations.

In addition, the following actions are available in the contextual menu:

Go to First Reference

Selects the first reference of the currently selected component in the dependencies tree.

Go to Component

Shows the definition of the currently selected component in the dependencies tree.

Related Information:

Searching and Refactoring Operations Scope in WSDL Documents *(on page 513)*

Highlight Component Occurrences in WSDL Documents

When you position your mouse cursor over a component in a WSDL document, Oxygen XML Developer Eclipse 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 *(on page 36)* 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 *(on page 248)*.

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 currently 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 currently 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:

Go to 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:

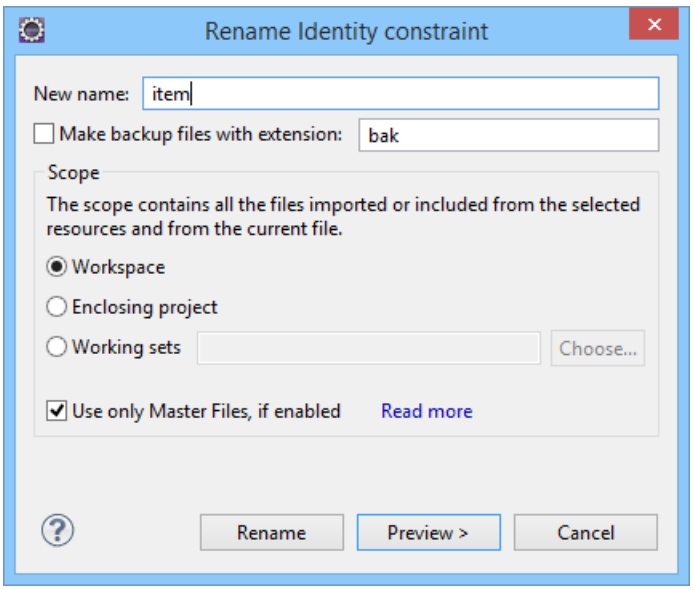
Rename Component

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.

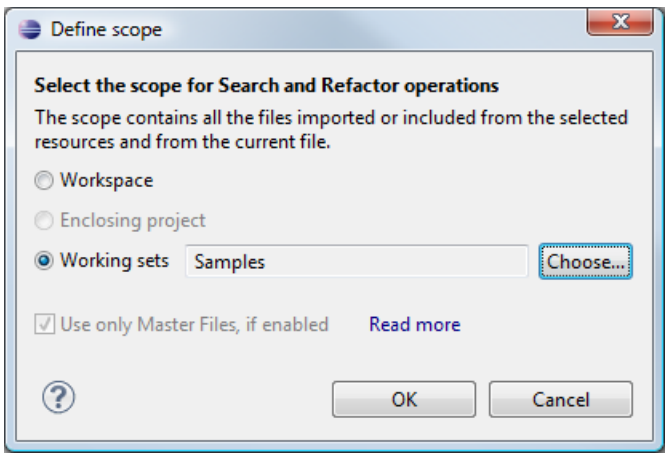
Figure 212. Rename Identity Constraint Dialog Box



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 (on page 1329)*. 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 *(on page 200)*.

Figure 213. Change Scope Dialog Box

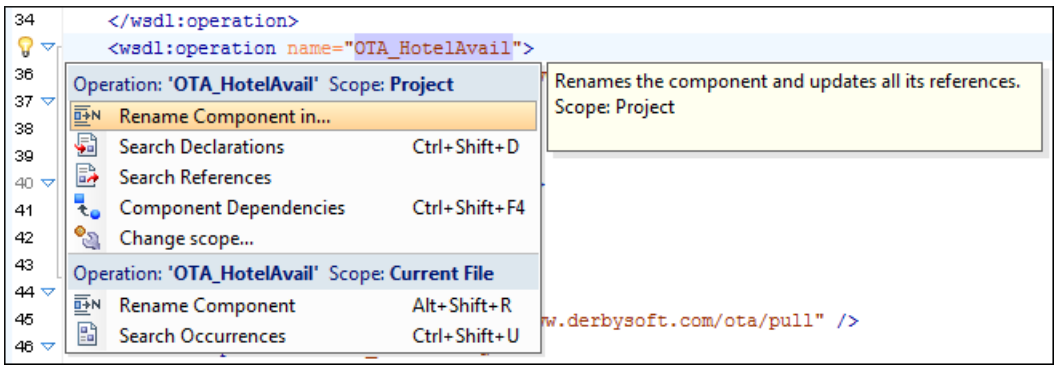


The scope you define is applied to all future search and refactor operations until you modify it. Contextual menu actions allow you to add or delete files, folders, and other resources to the *working set (on page 1329)* structure.

Quick Assist Support in WSDL Documents

The *Quick Assist* feature *(on page 1328)* 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 214. 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

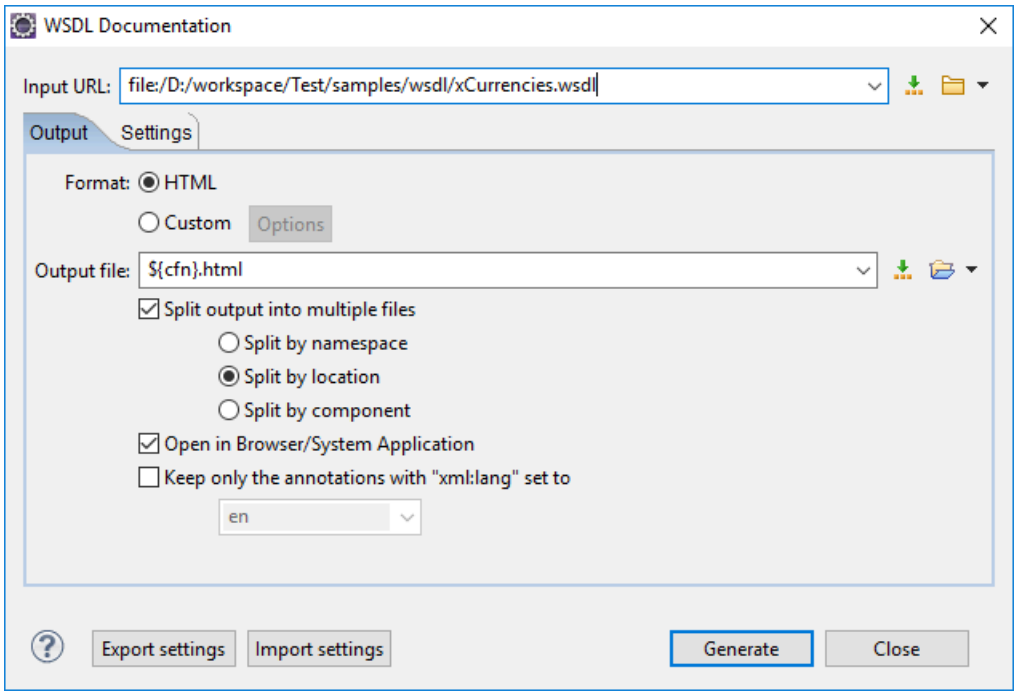
You can use Oxygen XML Developer Eclipse plugin to generate detailed documentation for the components of a WSDL document in HTML format. You can select the WSDL components to include in your output and the

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 *(on page 518)* 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 *(on page 192)*.

Figure 215. 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 *(on page 517)*.
  - **Custom** - The documentation is generated in a custom output format *(on page 518)*, 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.

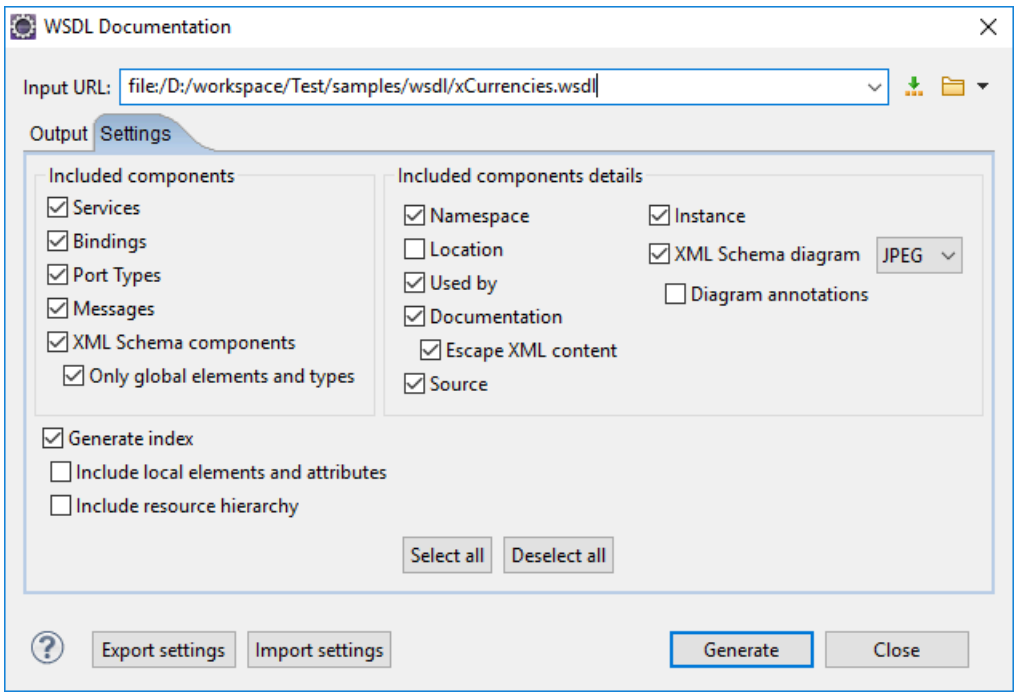
Note: To set the browser or system application that will be used, go to **Window > Preferences > General > Web Browser** and specify it there. This will take precedence over the default system application settings.

- **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 216. 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 selected, 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 deselected by default.

Export settings - Save the current settings in a settings file for further use (for example, if you need the exported settings file for generating the documentation from the command-line interface).

Import settings - Reloads the settings from the exported file.

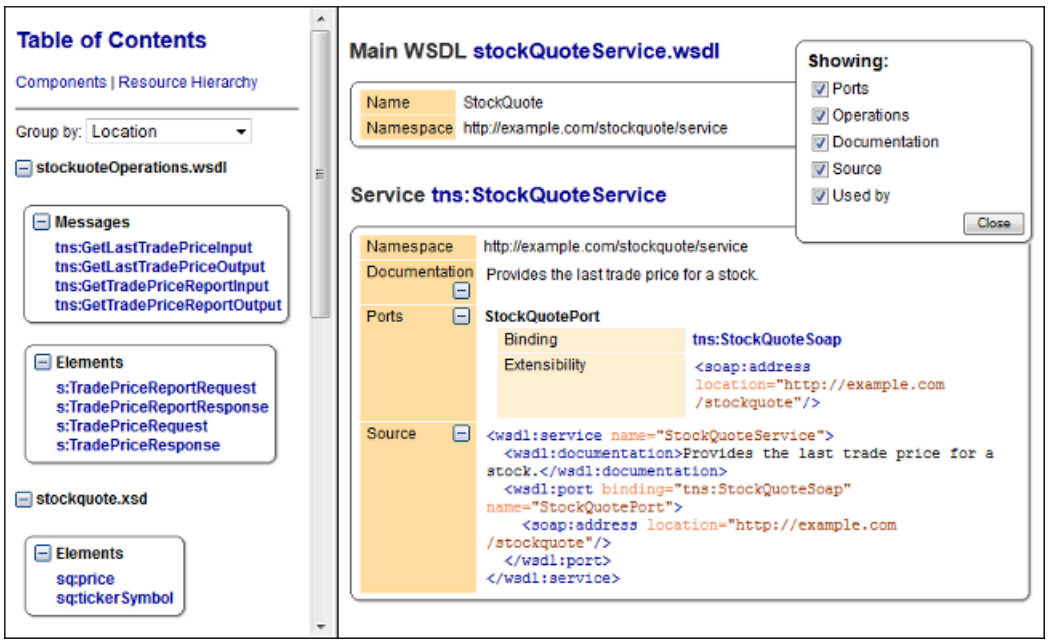
Generate - Use this button to generate the WSDL documentation.

Tip: This function can be executed from an automated command-line script, but it requires an additional **Oxygen XML Scripting License**. For more details, see https://www.oxygenxml.com/oxygen_scripting.html.

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 217. WSDL Documentation in HTML Format



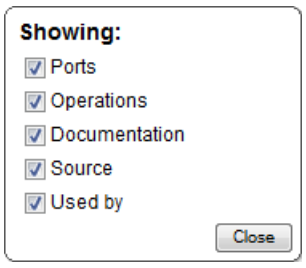
The documentation of each component is presented in a separate section. The title of the section is composed of the component type and the component name. The component information (namespace, documentation, etc.) is presented in a tabular form.

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 218. Showing Options



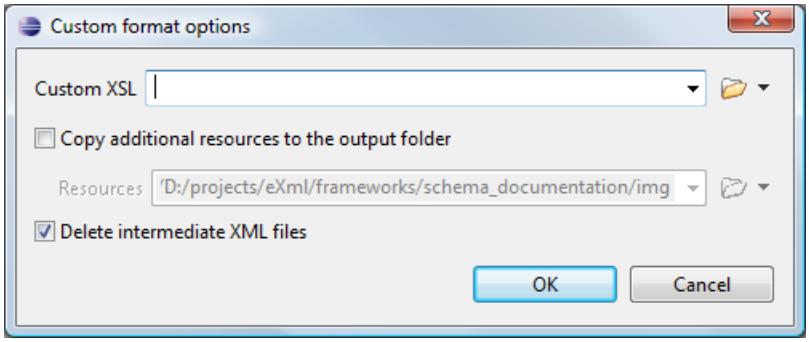
Generating WSDL Documentation in a Custom Format

To obtain the default HTML documentation output from a WSDL document, Oxygen XML Developer Eclipse 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 219. 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.

WSDL SOAP Analyzer Tool

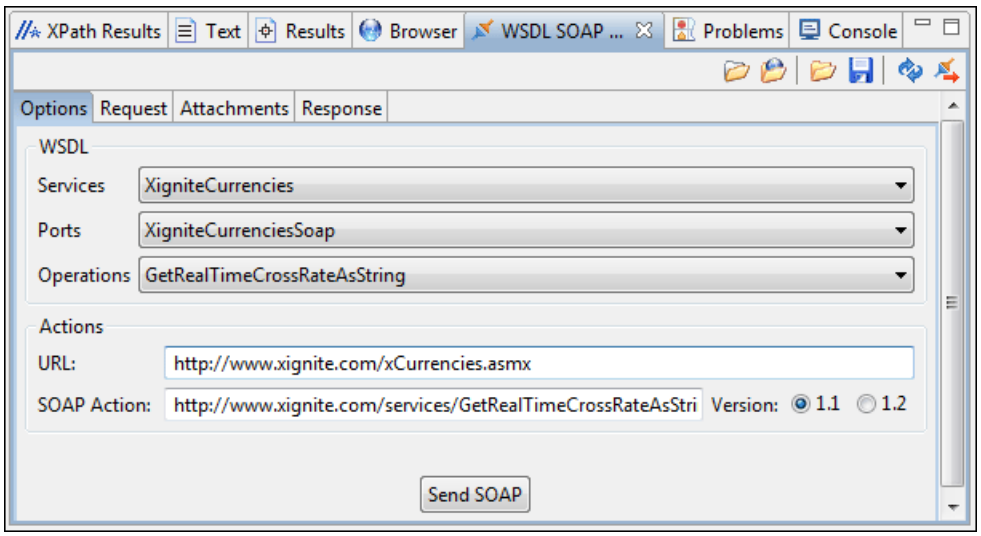
WSDL SOAP Analyzer is a tool that helps you test if the messages defined in a Web Service Descriptor (WSDL) are accepted by a Web Services server.

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).

Oxygen XML Developer Eclipse 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** *(on page 192)* view.

Figure 220. 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 Eclipse 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 (on page 1324)* 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 Eclipse plugin remembers the modified request for each one. You can click 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 Eclipse 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 Eclipse 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 *(on page 521)* 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 Eclipse plugin > WSDL SOAP Analyzer**.
2. Click the **Choose WSDL** button and enter the URL of the remote WSDL file.
3. Click the **OK** button.
   This will open the **WSDL SOAP Analyzer** tool *(on page 519)*. In the **Saved SOAP Request** tab you can open directly a previously saved Web Service SOAP Call (WSSC) file, thus skipping the analysis phase.

Editing CSS Stylesheets

Oxygen XML Developer Eclipse 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 *(on page 171)*.
- **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 Eclipse 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 Eclipse plugin includes a built-in *CSS Validator*, integrated with general validation support. This makes the usual validation features *(on page 279)* 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 *(on page 38)*. 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 *(on page 522)* 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>
```

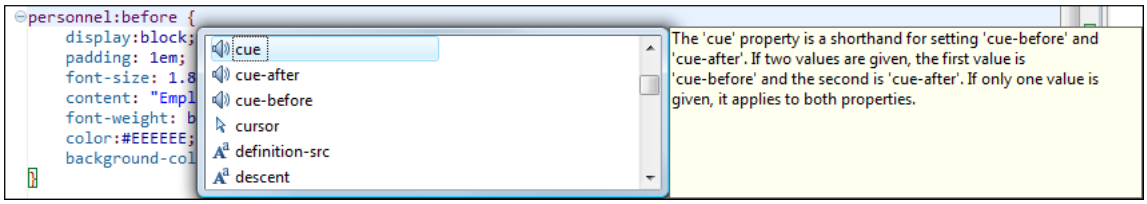
2. Go to your desktop and create the `builtin/css-validator/` folder structure.
3. Press and hold **Shift** and right-click anywhere on your desktop. From the contextual menu, select **Open Command Window Here**.
4. In the command line, run the `jar cvf custom_props.jar builtin/` command.
   The `custom_props.jar` 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 (on page 1324)*, similar to the one available for XML documents *(on page 234)* offers the CSS properties and the values available for each property. It can be manually 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 *(on page 238)* into CSS stylesheets. The code templates that are proposed include form controls, actions, and **Author** mode operations.

Figure 221. Content Completion in CSS Stylesheets



The properties and values available are dependent on the CSS Profile selected in the **CSS** preferences *(on page 38)*. 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 Information:

Specify Custom CSS Properties *(on page 522)*

Syntax Highlighting in CSS Files

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for CSS files, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
3. Select and expand the **CSS** 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 101)*

CSS Outline View

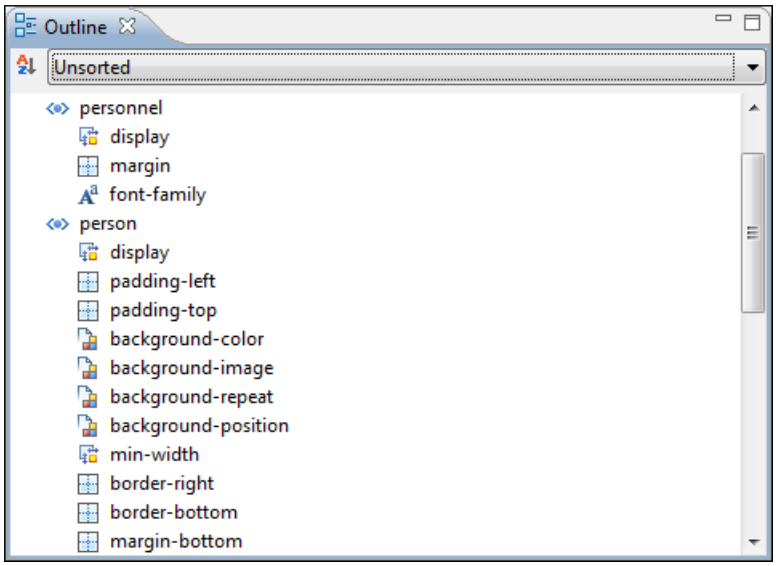
The **Outline** view for CSS stylesheets presents the import declarations for other CSS stylesheet files and all the selectors defined in the current CSS document. The selector entries can be presented as follows:

- 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 Eclipse 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 by selecting it from the **Window > Show View** menu.

Figure 222. 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 *(on page 231)* 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 *(on page 251)* 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 Eclipse 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 Eclipse 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 Eclipse plugin offers support for editing Relax NG schema files in the following editing modes:

- **Text editing mode** ***(on page 446)*** - 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** *(on page 526)* and **Logical Model View** *(on page 527)*.
- **Grid editing mode** ***(on page 168)*** - 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 303)*.

Related Information:

Associating a Schema to XML Documents *(on page 303)*

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 Eclipse plugin provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

You cat set a main Relax NG document either using the *master files* support from the **Navigator** view *(on page 200)*, or using a validation scenario.

To set a *master file* using a validation scenario, add validation units that point to the main schemas. Oxygen XML Developer Eclipse 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 main advantage of editing in the context of a *master file (on page 1326)* is that it provides correct validation of a module in the context of a larger schema structure.

Related Information:

Creating a New Validation Scenario *(on page 288)*

XML Schema Outline View *(on page 450)*

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 Eclipse 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.

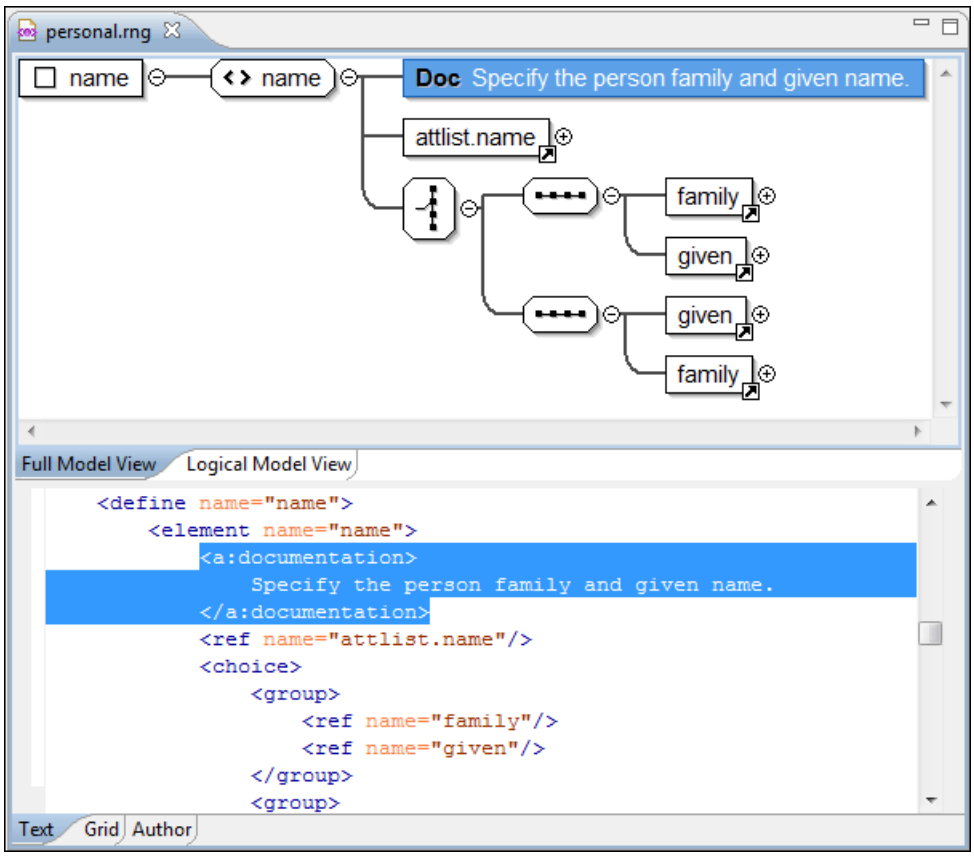
Oxygen XML Developer Eclipse plugin is the only XML editor to provide a side by side source and diagram presentation and have them real-time synchronized:

- 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** *(on page 527)*.

Figure 223. Relax NG Schema Editor - Full Model View



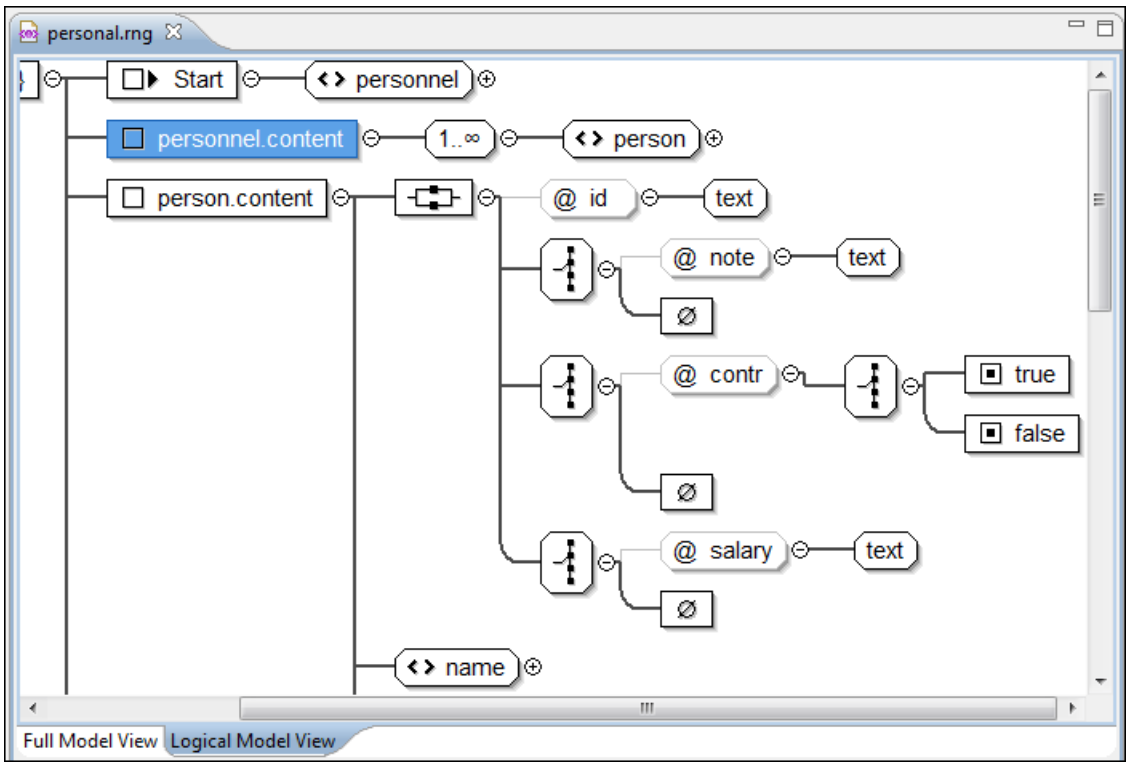
The following references can be expanded in place: patterns, includes, and external references. This expansion mechanism, coupled with the synchronization support, makes the schema navigation easy.

All the element and attribute names are editable by double-clicking the names.

Logical Model View

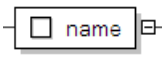
The **Logical Model View** presents the compiled schema in the form of a single pattern. The patterns that form the element content are defined as top-level patterns with generated names. These names are generated depending of the elements name class.

Figure 224. Logical Model View for a Relax NG Schema



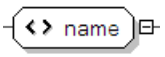
Symbols Used in the Schema Diagram

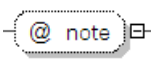
The views in the schema diagram editor renders all the Relax NG schema patterns with the following intuitive symbols:

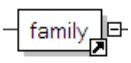
- 

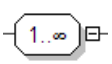
- `define` pattern with the `@name` attribute set to the value shown inside the rectangle (in this example `name`).
-

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

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

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

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

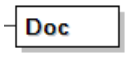
- `ref` pattern with the `@name` attribute set to the value shown inside the rectangle (in this case `family`).
- 

- `oneOrMore` pattern.

-

- `zeroOrMore` pattern.
-

- `optional` pattern.
-
- `choice` pattern.
-

- `value` pattern (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** *(on page 526)*, the contextual menu offers the following actions:

Go to definition (Available for imported components)

This action is available for imported components from other RNG files, and it shows where that component is defined.

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** *(on page 527)* 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 *(on page 36)*, go to **Editor > Document Checking**, and deselect the **Enable automatic validation** option *(on page 84)*.

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 Eclipse 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 Information:

- Validating XML Documents Against a Schema *(on page 279)*
- Embedding Schematron Rules in XML Schema or RELAX NG *(on page 595)*
- Validation Scenario *(on page 287)*
- Associating a Schema to XML Documents *(on page 303)*

Presenting Validation Errors in Text Mode *(on page 281)*

Content Completion in Relax NG Schemas

The intelligent *Content Completion Assistant (on page 1324)* allows you to quickly identify and insert elements, attributes, and attribute values that are valid in the current 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 *(on page 36)*, go to **Editor > Content Completion**, and deselect the **Enable content completion** option *(on page 78)*.

When active, the *Content Completion Assistant* displays a list of context-sensitive proposals valid at the current cursor position. It can be manually activated with the **Ctrl + Space (Command + Space on OS X)** shortcut. 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.

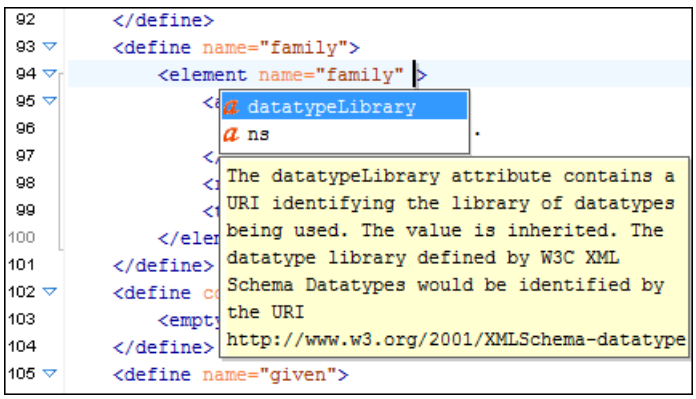
To insert the selected proposal in **Text** mode, do one of the following:

- 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.

If you are using the concept of *master files (on page 1326)* to import/include modules, 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 more information about the *Master Files* support in Oxygen XML Developer Eclipse plugin, see Defining Master Files at Project Level *(on page 200)*.

The *Content Completion Assistant* also offers additional information for the element and attribute proposals in the form of schema annotations that is displayed in a tooltip.

Figure 225. Relax NG Content Completion Assistant



Syntax Highlighting in Relax NG Schemas

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for Relax NG schemas, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
3. Select and expand the **XML** section in the top pane (for RELAX NG Compact Syntax schemas, select and expand the **RNC** section).
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 **XML** tab in the **Preview** pane to see the effects of your changes (for RELAX NG Compact Syntax schemas, the tab is **RNC**).

Tip: Oxygen XML Developer Eclipse plugin also allows you to specify syntax highlighting colors for specific XML elements and attributes with specific namespace prefixes. This can be done in the **Editor > Syntax Highlight > Elements/Attributes by Prefix** preferences page *(on page 101)*.

Related Information:

Syntax Highlight Preferences *(on page 101)*

Quick Fixes for DTD, XSD, and Relax NG Errors

Oxygen XML Developer Eclipse plugin offers *Quick Fixes (on page 1328)* 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 Eclipse 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

Problem Type	Available Quick Fixes
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

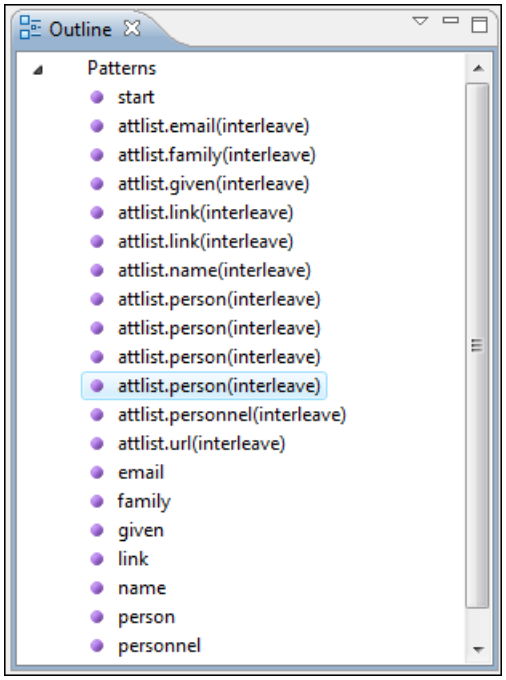
Related Information:

Schematron Quick Fixes (SQF) *(on page 303)*

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** *(on page 526)* and **Logical Model View** *(on page 527)* 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 by selecting it from the **Window > Show View** menu.

Figure 226. 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 *(on page 144)*.

Configure displayed attributes

Displays the **XML Structured Outline** preferences page *(on page 144)*.

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

Opens the **Component Dependencies** view *(on page 538)* that 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.

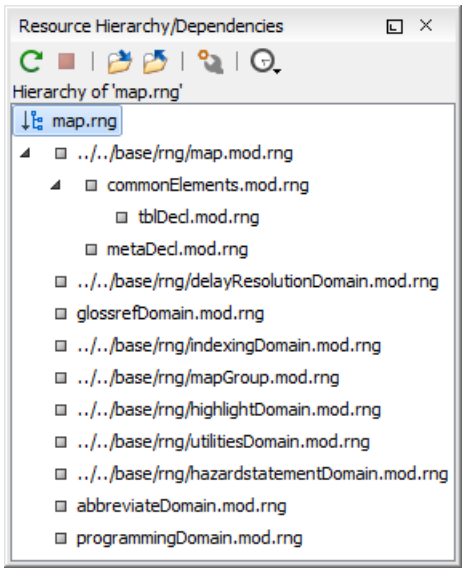
The upper part of the **Outline** view contains a filter box that allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (such as `*` or `?`) and separate multiple patterns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

RNG Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view displays the hierarchy or dependencies for resources included in an RNG schema. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

If you want to see the hierarchy or dependencies of an RNG schema, select the desired schema in the **Navigator** view *(on page 192)* and choose **Show Resource Hierarchy** or **Show Resource Dependencies** from the contextual menu.

Figure 227. Resource Hierarchy/Dependencies View



The following actions are available on the toolbar of 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 dependencies search scope

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 for a resource listed in the **Resource Hierarchy/Dependencies** view contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Go to reference

Opens the source document where the resource is referenced.

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 *(on page 200)*.

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 *(on page 509)*.

Related Information:

Search and Refactor Operations Scope *(on page 318)*

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 of the renamed resource(s)** - Select this option to update the references to the resource you are renaming. A **Preview** option is available that allows you to see what will be updated before selecting **Rename** to process the operation.

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)** - Select this option to update the references to the resource you are moving, in accordance with the new location and name. A **Preview** option is available that allows you to see what will be updated before selecting **Move** to process the operation.

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.

Relax NG Schema Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected component. This is helpful if you want to see where components are used in the entire hierarchy. For example, if you want to find all the references where a given component is used.

If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

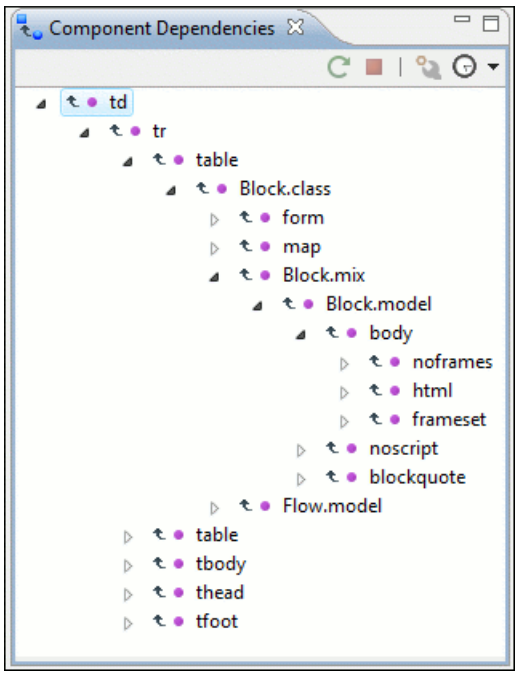
To see the dependencies of a Relax NG component:

1. Right-click the desired component in the editor or **Outline** view.
2. Select the **Component Dependencies** action from the contextual menu.

The action is available for all named defines.

If a component contains multiple references, a small table is displayed at the bottom of the view that contains all the references. When a recursive reference is encountered, it is marked with a special icon .

Figure 228. Component Dependencies View



The **Component Dependencies** view includes the following toolbar actions:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependency computation.

Configure

Allows you to choose the search scope for computing the dependencies structure. This is helpful for making sure all imported/included resources are computed.

History

Allows you to select from a list of the most recently used dependency computations.

In addition, the following actions are available in the contextual menu:

Go to First Reference

Selects the first reference of the currently selected component in the dependencies tree.

Go to Component

Shows the definition of the currently selected component in the dependencies tree.

Related Information:

Search and Refactor Operations Scope *(on page 318)*

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 currently 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 currently 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:

Go to 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:

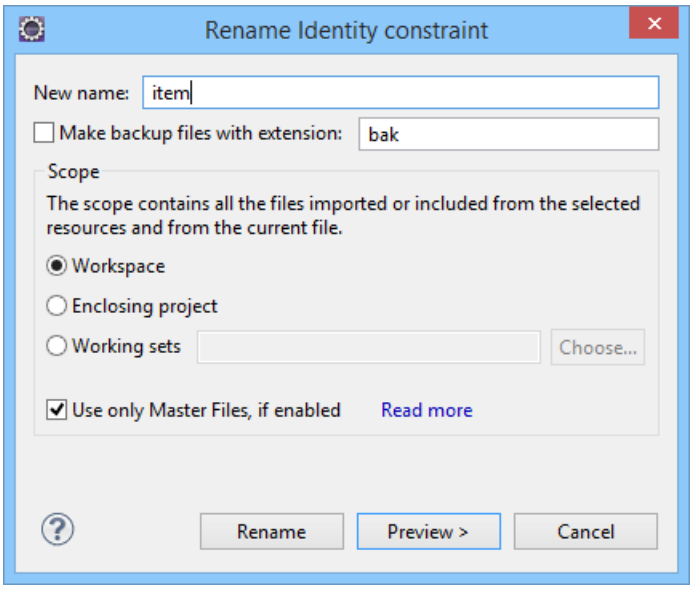
Rename Component

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 229. Rename Identity Constraint Dialog Box

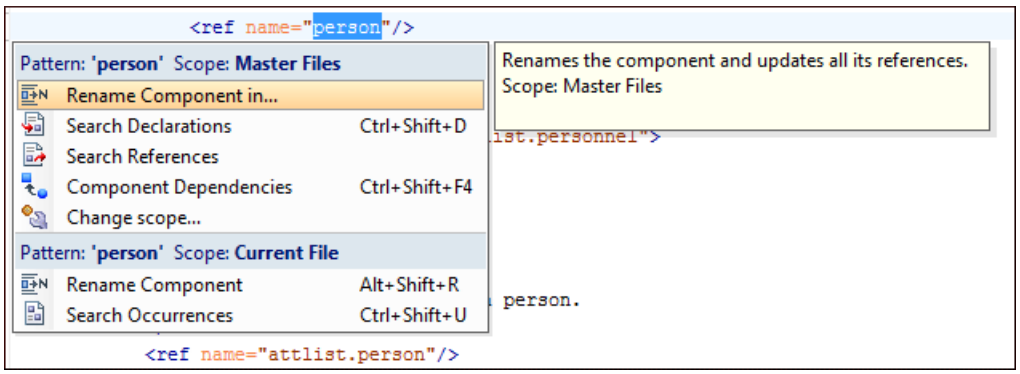


RNG Quick Assist Support

The *Quick Assist* support *(on page 1328)* improves the development work flow, offering fast access to the most commonly used actions when you edit schema documents.

The *Quick Assist* feature *(on page 1328)* 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 230. 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 538)*

Resource Hierarchy/Dependencies View *(on page 535)*

Searching and Refactoring Actions *(on page 539)*

Search and Refactor Operations Scope *(on page 318)*

Configuring a Custom Datatype Library for a RELAX NG Schema

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

The *JAR (on page 1326)* 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 Eclipse plugin offers support for editing NVDL schema files in the following editing modes:

- **Text editing mode** ***(on page 446)*** - 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** *(on page 543)* and **Logical Model View** *(on page 544)*.
- **Grid editing mode** ***(on page 168)*** - 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 303)*.

Related Information:

Associating a Schema to XML Documents *(on page 303)*

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 Eclipse 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.

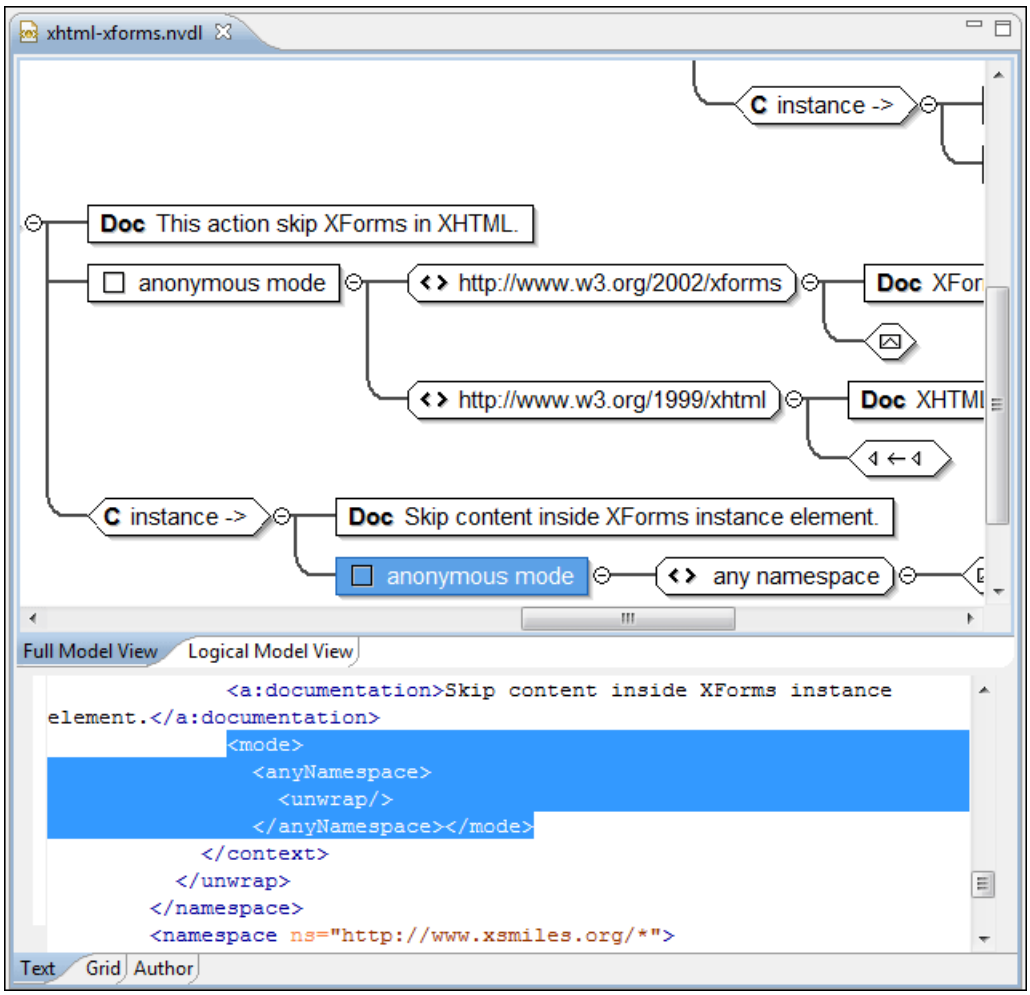
Oxygen XML Developer Eclipse plugin is the only XML Editor to provide a side by side source and diagram presentation and have them real-time synchronized:

- 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** *(on page 544)*.

Figure 231. NVDL Schema Editor - Full Model View



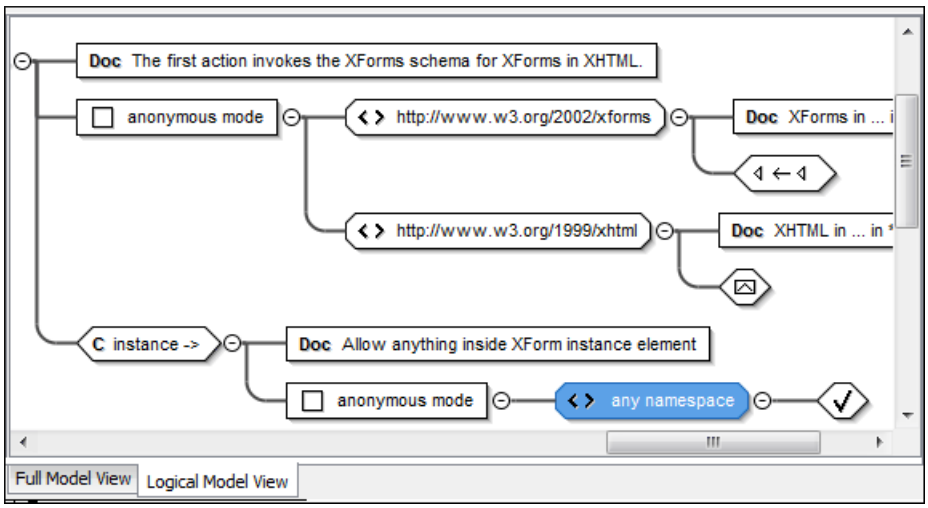
The **Full Model View** renders all the NVDL elements with intuitive icons. This representation coupled with the synchronization support makes the schema navigation easy.

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 232. 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** *(on page 544)* instead of the diagram.

Validating NVDL Schema Documents

By default, NVDL schema files are validated as you type. To change this, open the **Preferences** dialog box *(on page 36)*, go to **Editor > Document Checking**, and deselect the **Enable automatic validation** option *(on page 84)*.

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 Eclipse 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 279)*

Content Completion in NVDL Schemas

The intelligent *Content Completion Assistant (on page 1324)* allows you to quickly identify and insert elements, attributes, and attribute values that are valid in the current 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 *(on page 36)*, go to **Editor > Content Completion**, and deselect the **Enable content completion** option *(on page 78)*.

When active, the *Content Completion Assistant* displays a list of context-sensitive proposals valid at the current cursor position. It can be manually activated with the **Ctrl + Space (Command + Space on OS X)** shortcut. 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.

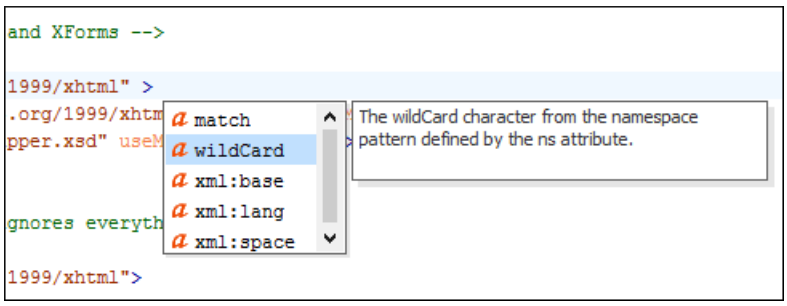
To insert the selected proposal in **Text** mode, do one of the following:

- 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.

If you are using the concept of *master files (on page 1326)* to import/include modules, 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 more information about the *Master Files* support in Oxygen XML Developer Eclipse plugin, see Defining Master Files at Project Level *(on page 200)*.

The *Content Completion Assistant* also offers additional information for the element and attribute proposals in the form of schema annotations that is displayed in a tooltip.

Figure 233. NVDL Content Completion Assistant



Syntax Highlighting in NVDL Schemas

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for NVDL schemas, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
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 **XML** tab in the **Preview** pane to see the effects of your changes.

Tip: Oxygen XML Developer Eclipse plugin also allows you to specify syntax highlighting colors for specific XML elements and attributes with specific namespace prefixes. This can be done in the **Editor > Syntax Highlight > Elements/Attributes by Prefix** preferences page *(on page 101)*.

Related Information:

Syntax Highlight Preferences *(on page 101)*

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 by selecting it from the **Window > Show View** menu.

NVDL Schema Component Dependencies View

The **Component Dependencies** view allows you to see the dependencies for a selected component. This is helpful if you want to see where components are used in the entire hierarchy. For example, if you want to find all the references where a given component is used.

If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

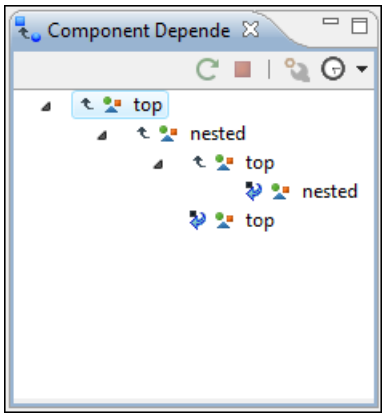
To see the dependencies of an NVDL component:

1. Right-click the desired component in the editor or **Outline** view.
2. Select the **Component Dependencies** action from the contextual menu.

The action is available for all named modes.

If a component contains multiple references, a small table is displayed at the bottom of the view that contains all the references. When a recursive reference is encountered, it is marked with a special icon .

Figure 234. Component Dependencies View



The **Component Dependencies** view includes the following toolbar actions:

Refresh

Refreshes the dependencies structure.

Stop

Stops the dependency computation.

Configure

Allows you to choose the search scope for computing the dependencies structure. This is helpful for making sure all imported/included resources are computed.

History

Allows you to select from a list of the most recently used dependency computations.

In addition, the following actions are available in the contextual menu:

Go to First Reference

Selects the first reference of the currently selected component in the dependencies tree.

Go to Component

Shows the definition of the currently selected component in the dependencies tree.

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 currently 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 currently 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:

Go to 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:

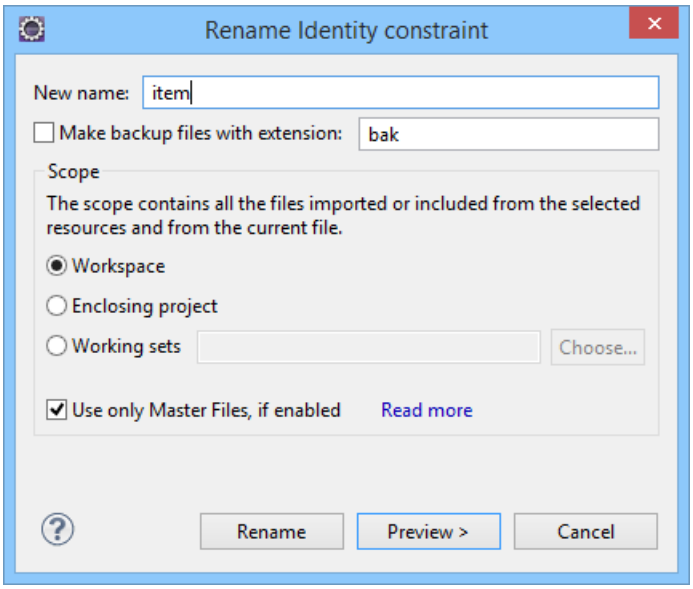
Rename Component

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 235. Rename Identity Constraint Dialog Box



Editing JSON Documents

Oxygen XML Developer Eclipse plugin includes a document template to help you get started with JSON documents. The template is called **JSON** and it can be found in the **New Document** folder in the **New from templates** wizard *(on page 177)*.

This section explains the features of the Oxygen XML Developer Eclipse plugin JSON Editor and how to use them.

For more information about JSON editing support, watch our video demonstration:

https://www.youtube.com/embed/wgIkagwSDrU

For more information about various JSON tools available in Oxygen XML Developer Eclipse plugin, watch our video demonstration:

https://www.youtube.com/embed/k3LHBU01GFI

Editing JSON Documents in Text Mode

When editing JSON documents in the **Text** editing mode, the usual text editing actions *(on page 225)* are available, along with other editor-specific actions, including:

- Search and Find/Replace *(on page 202)*
- Drag and Drop *(on page 232)*
- Validation *(on page 552)*
- Format and Indent (Pretty Print) *(on page 251)*

Note: You can run XPath expressions on open 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.

Navigating References in JSON Documents

When editing JSON documents (or JSON Schema), you can easily navigate *JSON Pointer* references and hyperlinks by using the **CTRL + Click** shortcut. Holding the **CTRL** key while hovering over a *JSON Pointer* references or hyperlink will change the reference to a clickable link.

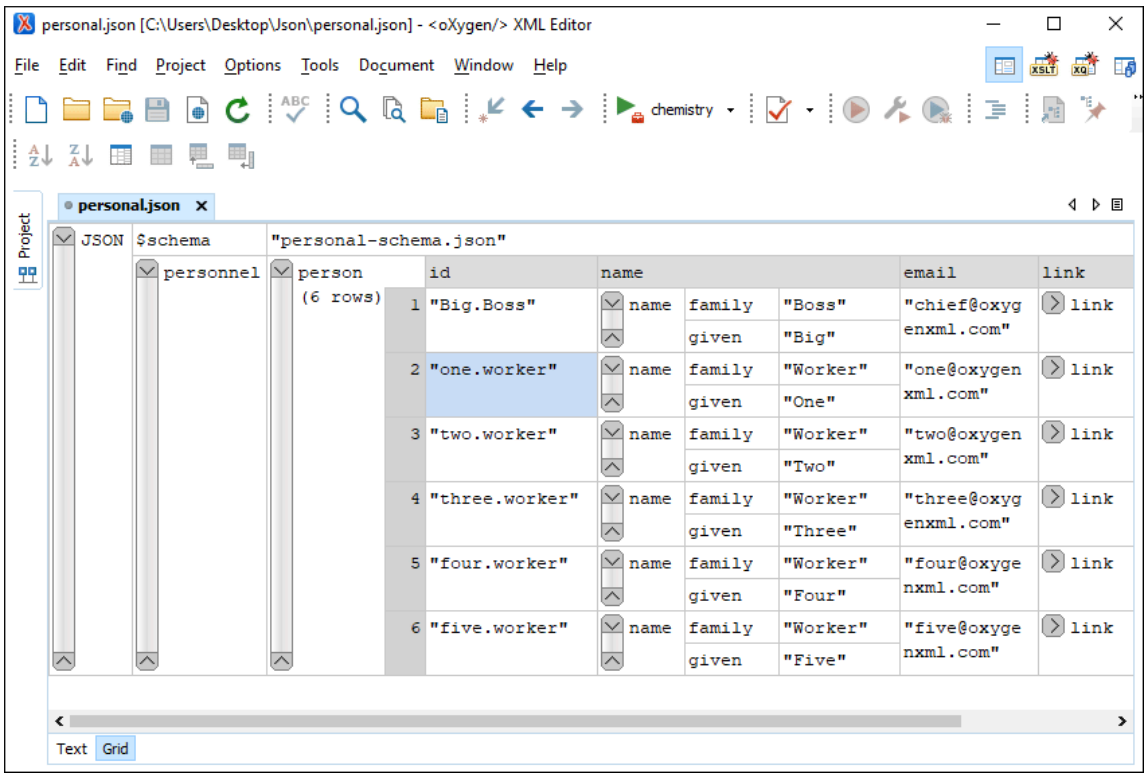
Related Information:

Editing JSON Documents in Grid Mode *(on page 551)*

Editing JSON Documents in Grid Mode

Oxygen XML Developer Eclipse plugin allows you to view and edit the JSON documents in the **Grid** mode *(on page 168)*. The JSON is represented in **Grid** mode as a compound layout of nested tables and the JSON data and structure can be easily manipulated with table-specific operations or drag and drop operations on the grid components.

Figure 236. JSON Editor Grid Mode



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 *(on page 85)* 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 168)*

Grid Preferences *(on page 85)*

Validating JSON Documents

Oxygen XML Developer Eclipse plugin includes a built-in JSON validator that is used to validate JSON documents against JSON Schemas, as well as a built-in JSON *Well-Formedness* validator (based on the free JAVA source code available at *www.json.org*). A built-in JSON Schematron Validator engine is also provided to validate JSON documents against a specified Schematron schema.

For more information, see the following video demonstration:

https://www.youtube.com/embed/3JEL6nFUozQ

Checking Well-Formedness in JSON Documents

A *Well-formed* JSON document is a sequence of Unicode code points that strictly conforms to the JSON grammar defined by the JSON Data Interchange Syntax specification. By default, Oxygen XML Developer Eclipse plugin automatically checks the document for *Well-formedness* as you type.

Check for Well-Formedness Manually

To manually check documents for *Well-Formedness*:

- Select the **Check Well-Formedness (Alt + Shift + V, W (Command + Alt + V, W on OS X))** action from the **Validation** drop-down menu on the toolbar or from the **XML** menu.
- A selection of files can be checked for well-formedness by selecting the **Check Well-Formedness** action from the **Validate** submenu when invoking the contextual menu in the **Navigator** view *(on page 192)*.

Result: 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.

Example: A non *Well-formed* JSON Document

```
{"person": { "name": "John Doe" }
```

This would result in the following error:

```
Expected a ',' or '}'
```

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, you would need to identify where the missing end bracket needs to be placed.

Validating JSON Documents Against JSON Schema or Schematron

A *valid* JSON document is a *well-formed* document that also conforms to the rules of a JSON Schema that defines the legal syntax of a JSON document. The purpose of the JSON schema is to define the legal properties and values of a JSON document.

This section contains topics that explain the automatic and manual validation possibilities in Oxygen XML Developer Eclipse plugin, how validation errors are presented, and information about built-in validation scenarios.

Oxygen XML Developer Eclipse plugin also includes a built-in JSON Schematron Validator engine to validate JSON documents against a Schematron schema specified in a custom validation scenario or using the **Validate with** action *(on page 554)*.

Tip: Inside the samples folder, there are a few files you can use to see how Schematron validation can be done with JSON files. The path of the folder containing these sample files is: `[OXYGEN_INSTALL_DIR]/samples/json/schematron/`.

For information about how to associate a schema for the purposes of validation, see Associating a Schema Through a Validation Scenario *(on page 560)*.

Automatic Validation

By default, Oxygen XML Developer Eclipse plugin is configured to automatically mark validation errors in the JSON document as you are editing. The **Enable automatic validation** option *(on page 84)* in the **Document Checking** preferences page *(on page 84)* 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 *(on page 84)* 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 554)*
- Presenting Validation Errors in JSON Documents *(on page 555)*

Manual Validation Actions

You can choose to validate JSON documents at any time by using the manual validation actions that are available in Oxygen XML Developer Eclipse plugin.

Manual Validation Actions

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 **JSON** menu, or from the **Validate** submenu when invoking the contextual menu in the **Navigator** view *(on page 192)*.

Validate with

Available from the **Validation** drop-down menu on the toolbar or the **JSON** menu.

This action opens a dialog box that allows you to specify a schema for validating the current document *(on page 561)*.

Validate with Schema

Available from the **Validate** submenu when invoking the contextual menu in the **Navigator** view *(on page 192)*.

This action opens a dialog box that allows you to specify a JSON or Schematron schema for validating the current document *(on page 561)*.

Other Validation Options

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 *(on page 84)*.

Related Information:

Automatic Validation *(on page 554)*

Presenting Validation Errors in JSON Documents *(on page 555)*

Presenting Validation Errors in JSON Documents

Validation errors and warnings in JSON documents are presented in various locations within the interface.

Validation Marker Locations

Validation issues are marked in the following locations:

- In the main editing pane, with the issue underlined in a color according to the type of issue.
- In the right-side vertical stripe, with a marker that is colored according to the type of issue.
- In the **Outline** view, with an icon that is colored according to the type of issue.

Validation Marker Colors

The colors for each type of issue are as follows:

- **Validation Errors** [Red] - By default, the underline in the editing pane, the marker in the right vertical stripe, and the foreground color of the attribute in the **Attributes** view are colored in red.
- **Validation Warnings** [Yellow] - By default, the underline in the editing pane, the marker in the right vertical stripe, and the foreground color of the attribute in the **Attributes** view are colored in yellow.
- **Validation Info** [Blue] - By default, the underline in the editing pane, the marker in the right vertical stripe, and the foreground color of the attribute in the **Attributes** view are colored in blue.

You can configure the colors and how the various types of validation problems are rendered from the Eclipse **Annotations** preferences page (**Window ('Eclipse' on Mac OSX) > Preferences > General > Editors > Text Editors > Annotations**).

Validation Markers in the Right-Side Stripe

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 tooltip when you hover over indicator square. If there are numerous problems, only the first three are presented in the tooltip.

Middle Part of the Stripe

Errors are presented 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 *(on page 36)*, go to **Editor > Document checking**, and specify the desired limit in the **Maximum number of validation highlights** option *(on page 84)*.

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

Bottom Part of the Stripe

Two navigation arrows ( ) can be used to jump to the next or previous issue. The same actions can be triggered from **Document > Automatic validation > Next Error/Highlight** (**Ctrl + Period (Command + Period on OS X)**) and **Document > Automatic validation > Previous Error/Highlight** (**Ctrl + Comma (Command + Comma on OS X)**).

Hovering Over Validation Issues

Hovering over a validation issue presents a tooltip message with more details about the problem. Also, when hovering over an issue, pressing **F2** will change the focus to the tooltip.

Details About Validation Issues

- Information about the issue is also displayed in the message area on the bottom of the editor panel (clicking the **Document checking options** button opens the **Document Checking** preferences page *(on page 84)* where you can configure some validation options.
- Status messages from every validation action are logged in the **Console** view *(on page 220)* (the **Enable Oxygen consoles** option *(on page 112)* must be selected in the **View** preferences page).
- If you want to see all the validation messages grouped in the **Results** view *(on page 248)*, you should use the **Validate** action from the toolbar or **JSON** 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.

Creating a JSON Validation Scenario

Validation scenarios can be used to associate one or more JSON Schemas with a JSON document *(on page 560)*. Oxygen XML Developer Eclipse plugin also includes a built-in JSON Schematron Validator engine that can be specified in the validation scenario to validate JSON documents against a specified Schematron schema.

Creating a JSON Validation Scenario

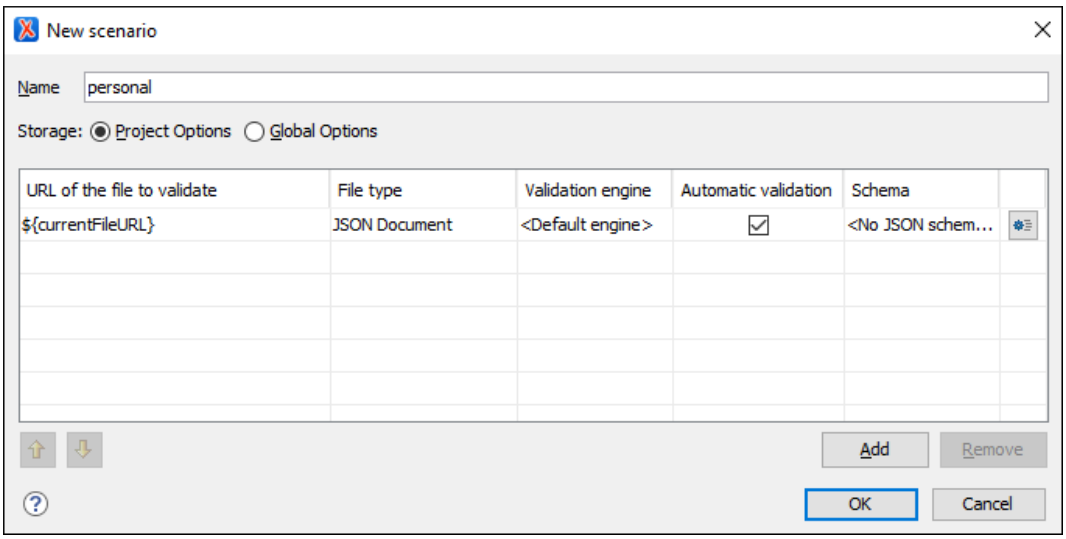
To create a validation scenario, follow these steps:

1. Select the **Configure Validation Scenario(s)** action in one of the following ways:
   - From the toolbar.
   - From the **JSON** menu.
   - From the **Validate** submenu, when invoking the contextual menu on a file in the **Navigator** view *(on page 192)*.

   Step Result: The **Configure Validation Scenario(s)** dialog box is displayed.
2. Click the **New** button.

   Step Result: A validation scenario configuration dialog box is displayed.

Figure 237. Validation Scenario Configuration Dialog Box



This scenario configuration dialog box allows you to configure the following information and options:

Name

The name of the validation scenario.

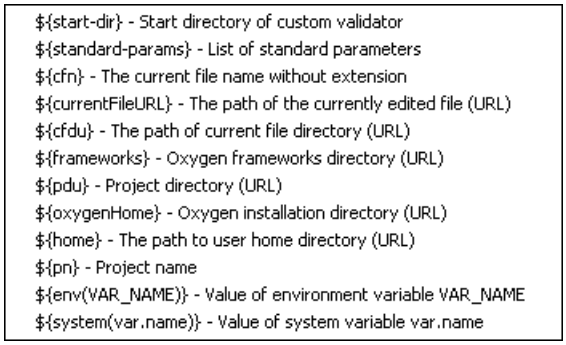
URL of the file to validate

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 *(on page 148)* or a custom editor variable *(on page 155)*.

Figure 238. Insert an Editor Variable



File type

The type of the document that is validated in the current validation unit. Oxygen XML Developer Eclipse plugin automatically selects the file type depending on the value of the **URL of the file to validate** field.

Validation engine

You can choose between the following types of validation engines for validating JSON documents:

- **Default engine** - The built-in JSON Validator will be used. For JSON Schema documents, this type should not be chosen unless the document has a schema version specified.
- **JSON Schema Validator** - This type is for JSON Schema documents only. It will use the version specified in the JSON Schema, or if a version is not specified, the JSON Schema draft-04 will be used.
- **JSON Schematron Validator** - The built-in JSON Schematron Validator will be used to validation JSON documents against a specified Schematron schema.

Note: For proper error localization, the root element of the Schematron schema should include the `@queryBinding` attribute with the value of **xslt2** after the Schematron namespace declaration:

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding=
"xslt2">
```

Automatic validation

If this option is selected, the validation operation defined by this row is also applied by the automatic validation feature *(on page 279)*. If the **Automatic validation** feature is disabled in the Document Checking preferences page *(on page 84)*, then this option is ignored, as the preference setting has a higher priority.

Schema

Displays the specified schema.

Specify Schema

Opens the **Specify Schema** dialog box that allows you to set a schema to be used for validating JSON documents.

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. You can use the buttons at the bottom of the table to add, remove, or move validation units.
4. Click **OK**.

Result: 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 click the **Apply associated** button to run the validation scenario.

Sharing JSON Validation Scenarios

The validation scenarios and their settings can be shared with other users by exporting them to a specialized scenarios file *(on page 147)* that can then be imported.

Resolving References with an XML Catalog

If a reference to a remote JSON schema must be used but a local copy of the schema should actually be preferred for performance reasons, the reference can be resolved to the local copy with an *XML Catalog (on page 1329)*.

For example, if the JSON schema contains a reference to a remote schema such as:

```
{"$ref": "http://json-schema.org/example/geo.json"}
```

the reference can be resolved to a local copy of the schema by inserting the following catalog entry:

```
<uri name="http://json-schema.org/example/geo.json" uri="schemas/geo.json"/>
```

Related Information:

Working with XML Catalogs *(on page 312)*

Associating a Schema to JSON Documents

To provide as-you-type validation and to compute valid proposals for the *Content Completion Assistant (on page 1324)*, Oxygen XML Developer Eclipse plugin requires a schema to be associated with the JSON document. The schema specifies how the internal structure is defined.

Detecting the Schema(s) for Validation and Content Completion

For validation, Oxygen XML Developer Eclipse plugin tries to detect the JSON Schema by searching in the following order:

1. The schema referenced in validation stages from the validation scenario(s) *(on page 560)* associated with the current JSON document.
2. If a schema is not detected, then it falls back to the schema associated directly in the JSON document *(on page 562)*.

Tip: To quickly open the schema used for validating the current document, select the **Open Associated Schema** action from the toolbar (or **JSON** menu).

Associating a Schema Through a Validation Scenario

Oxygen XML Developer Eclipse 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 JSON documents. Oxygen XML Developer Eclipse plugin includes built-in validation engines for validating JSON documents against a JSON Schema or Schematron schema. There are several methods that can be used to validate JSON document with a schema.

Configure a Validation Scenario and Specify the Schema

You can specify the schema to be used for validation directly in the JSON validation scenario *(on page 557)*. 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, from the **JSON** menu, or from the **Validate** submenu when invoking the contextual menu on a JSON file in the **Navigator** view *(on page 192)*).
2. Click the **New** button to create a new validation scenario *(on page 557)* or the **Edit** button to modify an existing one.
3. Add or configure validation units according to your needs. For details about all of the configuration options, see Creating a JSON Validation Scenario *(on page 557)*.
4. Click the **Specify Schema** button to select the schema to be associated with the validation unit.
5. Click **OK** on both dialog boxes.

Result: The schema is now associated with that validation scenario whenever it is invoked.

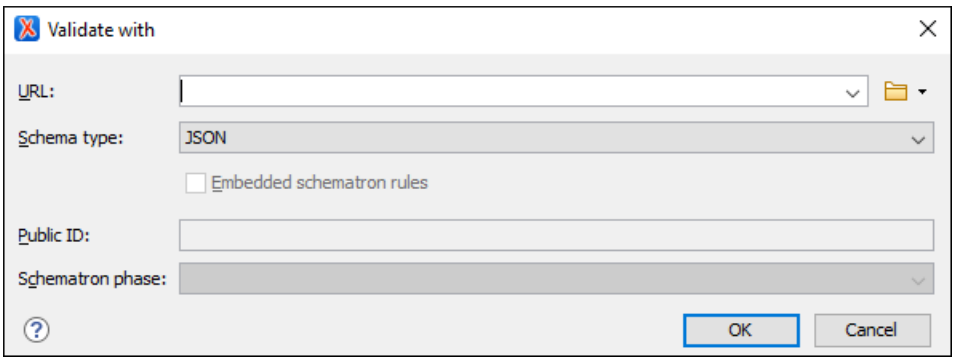
Use the Validate with Action to Specify a Schema for Validating the Current Document

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 **JSON** menu).

   Step Result: The **Validate with** dialog box is displayed:

   Figure 239. 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** *(on page 148)* button, or the browsing actions in the **Browse** drop-down list.
   - **Schema type** - You can select one of the following two types (other types of schema will not work with JSON documents):
     - **JSON** - Used for validating JSON documents against a specified JSON Schema.
     - **Schematron** - Used for validating JSON documents against a specified Schematron schema. You can also select a **Schematron phase** that you want to use for the validation.

       Note: For proper error localization, the root element of the Schematron schema should include the `@queryBinding` attribute with the value of **xslt2** after the Schematron namespace declaration:

       ```
       <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2
       ">
       ```

2. Select the schema to be associated with the manual validation.
3. Click **OK**.

Result: The current document is validated using the schema you specified.

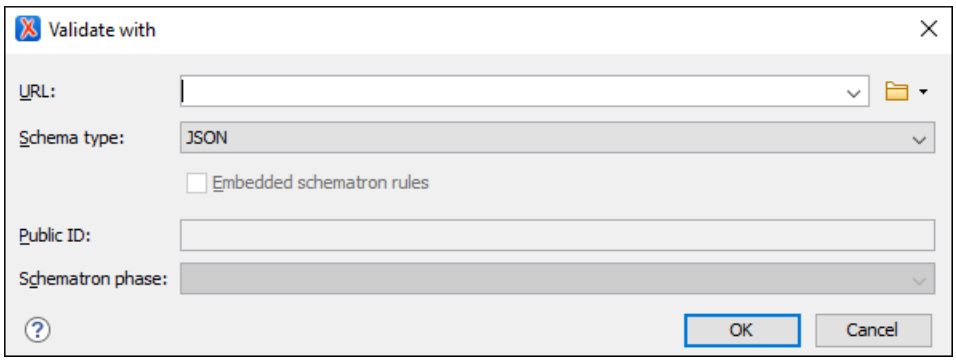
Use the Validate with Schema Action to Specify a Schema for Validating all Selected JSON Documents

To validate multiple JSON documents using a specified schema, follow these steps:

1. Select all the JSON 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 240. 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** *(on page 148)* button, or the browsing actions in the **Browse** drop-down list.
   - **Schema type** - You can select one of the following two types (other types of schema will not work with JSON documents):
     - **JSON** - Used for validating JSON documents against a specified JSON Schema.
     - **Schematron** - Used for validating JSON documents against a specified Schematron schema. You can also select a **Schematron phase** that you want to use for the validation.

       Note: For proper error localization, the root element of the Schematron schema should include the `@queryBinding` attribute with the value of **xslt2** after the Schematron namespace declaration:

       ```
       <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2
       ">
       ```
3. Select the JSON schema that you want to use to validate all selected JSON documents.
4. Click **OK**.

Result: The selected JSON documents are validated using the JSON schema you specified.

Associating a Schema Directly in JSON Documents

Associate Schema Action

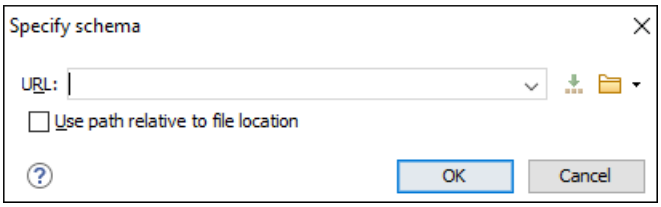
The schema used by the *Content Completion Assistant (on page 1324)* and document validation engine can be associated with the current document by using the **Associate Schema** action. The association can specify a relative file path or a URL of the schema.

To associate a JSON Schema to the current JSON document, follow these steps:

1. Select the **Associate Schema** action from the toolbar (or **JSON** menu).

   Step Result: The **Associate Schema** dialog box is displayed:

   Figure 241. Associate Schema Dialog Box

   This dialog box contains the following options for JSON documents:
   - **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** - Select this option if the JSON instance document and the associated schema contain relative paths. The location of the schema file is inserted in the JSON 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.
2. Select the JSON Schema that will be associated with the JSON document.
3. Click **OK**.

Result: A `$schema` property is added at the beginning of the document with its value set to the specified URL. If the document already contained a schema association, the old association will be replaced with the new one.

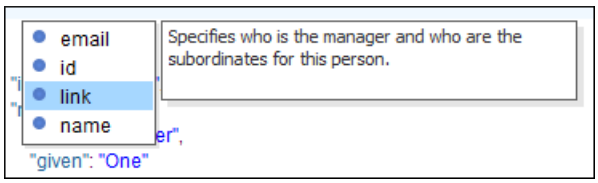
Tip: To quickly open the schema used for validating the current document, select the **Open Associated Schema** action from the toolbar (or **JSON** menu).

Content Completion Assistant in JSON

Oxygen XML Developer Eclipse plugin includes an intelligent *Content Completion Assistant (on page 1324)* that offers proposals for inserting JSON structures that are valid at the current editing location.

The *Content Completion Assistant* is enabled by default. To disable it, open the **Preferences** dialog box *(on page 36)*, go to **Editor > Content Completion**, and deselect the **Enable content completion** option *(on page 78)*.

Figure 242. Content Completion Assistant in JSON



Content Completion and the Associated Schema

The *Content Completion Assistant* feature is schema-driven and the list of proposals in the *Content Completion Assistant (on page 1324)* depend on the associated JSON Schema. For information about ways to associate a schema to a JSON document, see the Associating a Schema to JSON Documents *(on page 560)* section.

Using the Content Completion Assistant in JSON

The feature is activated in **Text** mode for JSON documents by:

- Typing a quote symbol (") to insert a property or value.
- Pressing **Ctrl + Space (Command + Space on OS X)** or **Alt + ForwardSlash (Command + Alt + ForwardSlash on OS X)**.

You can navigate through the list of proposals by using the **Up** and **Down** keys on your keyboard. In some cases, the *Content Completion Assistant* displays a documentation window with information about the particular proposal *(on page 565)*. You can also change the size of the documentation window by dragging its top, right, and bottom borders.

To insert the selected proposal, press **Enter** or **Tab**.

Types of Proposals Listed in the Content Completion Assistant for JSON

The proposals that populate the *Content Completion Assistant* for JSON documents depend on the structure defined in the associated JSON Schema. The types of structure proposed in the content completion window include:

- JSON properties
- JSON values
- JSON arrays
- JSON objects

The number and type of proposals displayed by the *Content Completion Assistant* is dependent on the cursor's current position in the JSON document and the child items displayed within a given context are defined by the structure of the specified JSON Schema.

Code Templates in the Content Completion

Oxygen XML Developer Eclipse plugin includes a set of built-in code templates for JSON documents that can be selected from the *Content Completion Assistant*. The code templates are displayed with a symbol in the

content completion list. You can also define your own code templates and share them with others. For more information, see Code Templates *(on page 238)*.

Schema Annotations in JSON Content Completion

A schema annotation is a documentation snippet that appears in the *Content Completion Assistant (on page 1324)* offering more information about the current proposal.

This feature is enabled by default, but you can disable it by deselecting the **Show annotations in Content Completion Assistant** *(on page 79)* option in the **Annotations** preferences page.

Collecting Annotations from the JSON Schema

In a JSON Schema, the annotations are specified in the value of the `title` and `description` properties like this:

```
"idType": {
  "title": "The 'id' property",
  "description": "Specifies a required ID for this person.",
  "type": "string",
  "maxLength": 20
}
```

Syntax Highlighting in JSON Documents

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for JSON files, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
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 101)*

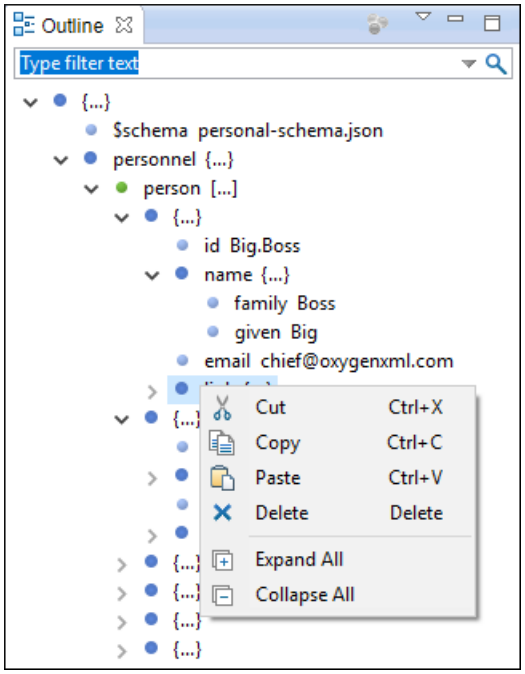
Folding in JSON

In a large JSON document, the data enclosed in the curly bracket characters `{}` can be collapsed so that only the needed data remains in focus. The folding features available for XML documents *(on page 231)* are also available in JSON documents.

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 by selecting it from the **Window > Show View** menu.

Figure 243. JSON Outline View



Outline View Features

The **Outline** view allows you to:

- Quickly navigate through the document by selecting nodes in the **Outline** tree.
- 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 by selecting it 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 (such as `*` or `?`) and separate multiple patterns with commas.

It also includes a **View menu** in the top-right corner that presents the following options to help you filter the view even further.

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.

Drag and Drop Actions in the Outline View

Entire JSON properties, objects, and arrays 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 a JSON node in the **Outline** view and drop it on another node, then the dragged node will be moved after the drop target.
- If you hold the mouse pointer over the drop target for a short time before the drop then the drop target node will be expanded first and the dragged node will be moved inside the drop target.
- You can also drop a node before or after another node if you hold the mouse pointer towards the upper or lower part of the target. A marker will indicate whether the drop will be performed before or after the target node.
- If you hold down the **Ctrl (Command on OS X)** key after dragging, a copy operation will be performed instead of a move.

Contextual Menu Actions

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

Cut

Cuts the currently selected component.

Copy

Copies the currently selected component.

Paste

Pastes the copied 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.

XSD to JSON Schema Converter

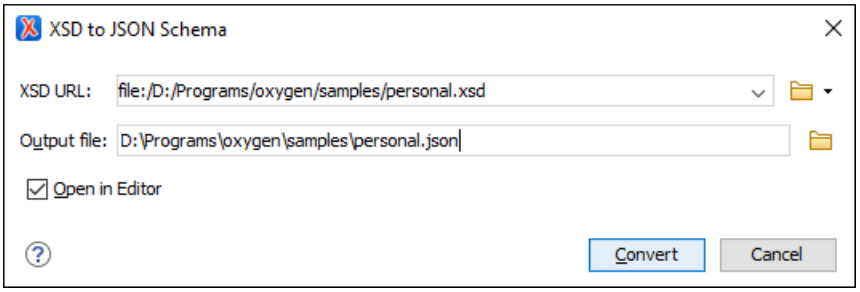
Oxygen XML Developer Eclipse plugin includes a tool for converting an XML Schema file (XSD) to a JSON Schema file. The **XSD to JSON Schema** action for invoking the tool can be found in the **Tools > JSON Tools** menu. It requires an additional add-on to be installed, so the first time you invoke the action, Oxygen XML Developer Eclipse plugin will present a dialog box asking if you want to install it. Once installed, you need to restart Oxygen XML Developer Eclipse plugin and the **XSD to JSON Schema** action will invoke the tool.

To convert an XML Schema (XSD) to a JSON Schema, follow these steps:

1. Select the **XSD to JSON Schema** action from the **Tools > JSON Tools** menu.

   Step Result: The **XSD to JSON Schema** dialog box is displayed:

   Figure 244. XSD to JSON Schema Dialog Box

2. Choose or enter the **XSD URL** of the XML Schema document. The conversion supports XSD versions 1.0 and 1.1.
3. Choose the path for the **Output file**.
4. Select the **Open in Editor** option to open the resulting JSON Schema document in the main editing pane.
5. Click the **Convert** button.

Result: The original XSD document is now converted to a JSON Schema document. The resulting JSON Schema will be *draft-07* and will contain:

- The `$id` of the schema, generated from XSD `targetNamespace`.
- The `$definitions` section, which declares complex and *enum* types.
- The `anyOf` section, which lists possible top-level elements as an array of objects.

Other Possible Results:

- If an XSD type extends another type, then its schema is combined with the schema of the base type using the `allOf` keyword.
- If an extension in XSD defines an element with the same name as an attribute in the base, a property named `rest` is generated to avoid name conflicts in JSON.
- If a property of a complex type is a collection property, the schema of the collection items will be wrapped in the JSON array schema.

Conversion Mappings

The following table lists the specific conversion mapping details.

XML Schema Type	JSON Schema Representation
anySimpleType	string, number, integer, boolean, null
anyType	string, number, integer, boolean, null, object, array
string	string
normalizedString	string
token	string
language	string
Name	string
NCName	string
ID	string
IDREF	string
IDREFS	array of strings
ENTITY	string
ENTITIES	array of strings
NMTOKEN	string
NMTOKENS	array of strings
boolean	boolean
base64Binary	array of integers
hexBinary	array of integers
float	number
decimal	number
integer	integer
nonPositiveInteger	integer
negativeInteger	integer
long	integer
int	integer
short	integer
byte	integer

XML Schema Type	JSON Schema Representation
nonNegativeInteger	integer
unsignedLong	integer
unsignedInt	integer
unsignedShort	integer
unsignedByte	integer
positiveInteger	integer
double	number
anyURI	string with "format":"uri"
QName	object with "namespaceURI", "localPart", "prefix"
duration	string
dateTime	string with "format":"date-time"
date	string with "format":"date"
time	string with "format":"time"

Conversion Limitations

In most cases, the conversion creates an equivalent schema, but there are some limitations:

- Restrictions/facets are not taken into consideration when converting (*fractionDigits*, *pattern*, *totalDigits*, *whiteSpace*, *minInclusive*, *maxInclusive*, and the restrictions for length, except *enumeration)*. However, extensions and indicators are properly converted (*minOccurs*, *maxOccurs*, *group*, *sequence*, *choice*).
- The `<documentation>` element is not converted into `<description>`.
- The `@substitutionGroup` attribute for an element that has no declared type becomes a reference to the element that can substitute it.
- The `@block` attribute is not taken into consideration during the conversion.

JSON to XML Converter

Online JSON to XML Converter

⚠ **Attention:**

For a simple ONLINE tool for converting a single JSON file to XML, or vice versa, go to: https://www.oxygenxml.com/xml_json_converter.html.

Converting JSON to XML in Oxygen

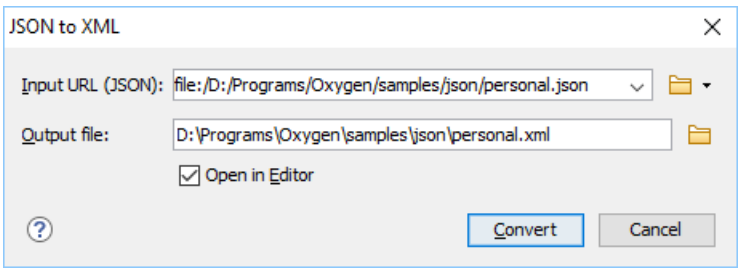
Oxygen XML Developer Eclipse plugin includes a useful and simple tool for converting JSON files to XML. The **JSON to XML** action for invoking the tool can be found in the **XML Tools > JSON Tools** menu.

To convert a JSON document to XML, follow these steps:

1. Select the **JSON to XML** action from the **XML Tools > JSON Tools** menu.

   The **JSON to XML** dialog box is displayed:

   Figure 245. JSON to XML Dialog Box

2. Choose or enter the **Input URL** of the JSON document.
3. Choose the path of the **Output file** that will contain the resulting XML document.
4. Select the **Open in Editor** option to open the resulting XML document in the main editing pane.
5. Click the **Convert** button.

Result: The original JSON document is now converted to an XML document.

Figure 246. Example: XML to JSON Operation Result

Conversion Details

- If the JSON document has more than one property on the first level, the converted XML document will have an additional root element called `<JSON>`.

For example, the following JSON document:

```
{
   "personnel": {
    "person": [
      {"name": "Boss"},
      {"name": "Worker"}
    ]
  },
  "id":"personnel-id"
}
```

it is converted to:

```
<?xml version="1.0" encoding="UTF-8"?>
<JSON>
  <personnel>
    <person>
      <name>Boss</name>
    </person>
    <person>
      <name>Worker</name>
    </person>
  </personnel>
  <id>personnel-id</id>
</JSON>
```

- If the JSON document is an array, the converted XML document will have a root element called `<array>` and for each item within the array, another `<array>` is created.

```
[
  {"name": "Boss"},
  {"name": "Worker"}
]
```

it is converted to:

```
<?xml version="1.0" encoding="UTF-8"?>
<array>
  <array>
    <name>Boss</name>
  </array>
  <array>
    <name>Worker</name>
  </array>
</array>
```

- If the name of a JSON property contains characters that are not valid in XML element names (for example, **$**), then the invalid characters will be escaped as its hexadecimal equivalent in the converted XML.

  ```
  {"$id": "personnel-id"}
  ```

  is converted to:

  ```
  <_X24_id>personnel-id</_X24_id>
  ```

Related Information:

XML to JSON Converter *(on page 573)*

XML to JSON Converter

Online XML to JSON Converter

⚠ **Attention:**

For a simple ONLINE tool for converting a single XML file to JSON, or vice versa, go to: https://www.oxygenxml.com/xml_json_converter.html.

Converting XML to JSON in Oxygen

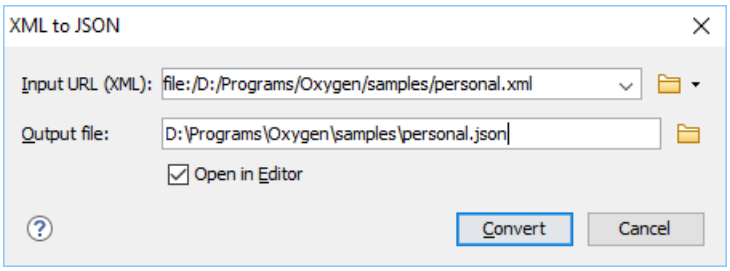
Oxygen XML Developer Eclipse plugin includes a useful and simple tool for converting XML files to JSON. The **XML to JSON** action for invoking the tool can be found in the **XML Tools > JSON Tools** menu.

To convert an XML document to JSON, follow these steps:

1. Select the **XML to JSON** action from the **XML Tools > JSON Tools** menu.

   Step Result: The **XML to JSON** dialog box is displayed:

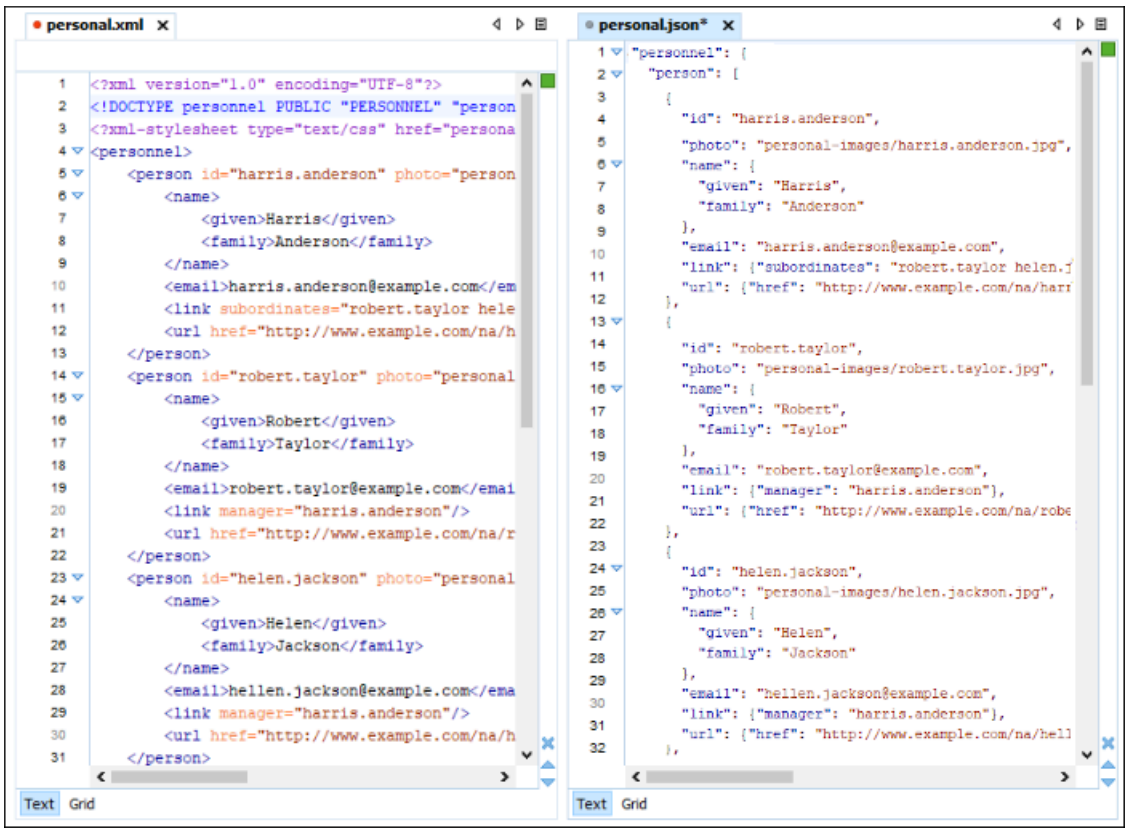
   Figure 247. XML to JSON Dialog Box

2. Choose or enter the **Input URL** of the XML document.
3. Choose the path of the **Output file** that will contain the resulting JSON document.
4. Select the **Open in Editor** option to open the resulting JSON document in the main editing pane.
5. Click the **Convert** button.

Result: The original XML document is now converted to a JSON document.

Figure 248. Example: XML to JSON Operation Result



Conversion Details

- If any elements contain attributes in the XML document, the attributes are converted to properties in the converted JSON document. If the XML document contains more than one element with the same name, they will be converted into an array of object in the converted JSON document.

  For example, the following XML document:

```xml
<personnel>
  <person id="person.one">
    <name>Boss</name>
  </person>
  <person id="person.two">
    <name>Worker</name>
  </person>
</personnel>
```

  it is converted to:

```
{
  "personnel": {
    "person": [
      {
        "id": "person.one",
        "name": "Boss"
```

```
      },
      {
        "id": "person.two",
        "name": "Worker"
      }
    ]
  }
}
```

- If the XML document contains elements with mixed content (text plus elements), the converted JSON document will contain a `#text` property with its value set as the text content. If there are multiple text nodes, the subsequent `#text` properties will contain a number (i.e. `#text1`, `#text2`).

  ```
  <person>
    <name>Boss <i>One</i>!</name>
  </person>
  ```

  is converted to:

  ```
  {
    "person": {
      "name": {
        "#text": "Boss",
        "i": "One"
        "#text1": "!"
      }
    }
  }
  ```

- If the XML document contains element names that contains hexadecimal codes (for example, if they were escaped during a JSON to XML conversion *(on page 570)*), it will be converted to the normal character value in the converted JSON document.

  ```
  <_X24_id>personnel-id</_X24_id>
  ```

  is converted to:

  ```
  {"$id": "personnel-id"}
  ```

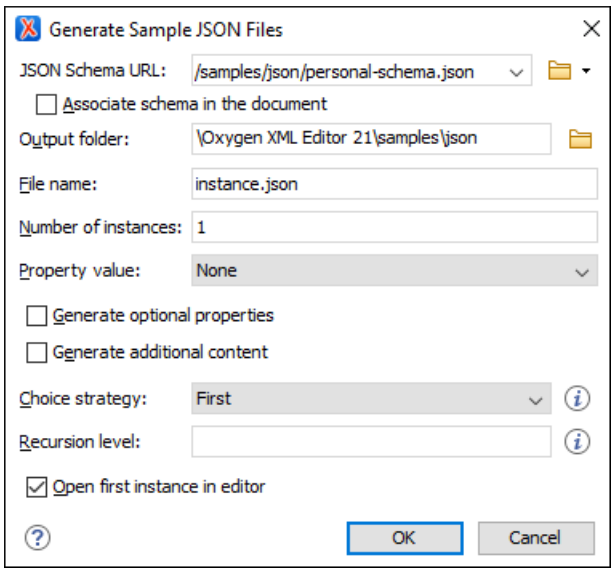
Related Information:

JSON to XML Converter *(on page 570)*

Generating Sample JSON Files

Oxygen XML Developer Eclipse plugin includes a tool for generating sample JSON files. To generate sample JSON files from a JSON Schema, select **Generate Sample JSON Files** from the **XML Tools > JSON Tools** menu. The action opens a dialog box where you can configure a variety of options for generating the files.

Figure 249. Generate Sample JSON Files Dialog Box



The **Generate Sample JSON Files** dialog box includes the following fields and options:

Schema URL

The URL of the Schema location. You can specify the path by using the text field, the history drop-down menu, or the browsing actions in the **Browse** drop-down list.

Associate schema in the document

If enabled, the specified schema will be associated with the generated files.

Output folder

Path to the folder where the generated JSON instances will be saved.

File name

The name of the instance(s) that will be generated. By default, `instance.json` is used.

Number of instances

The desired number of JSON instances to be generated. When more than one instance is generated, the index of the instance will be added to its file name.

Property value

You can specify the way the values of the properties are generated. The following options are available:

- *None* - Assigns empty values for properties (a template file will be generated). This is the default value.
- *Default* - Assigns the name of the property as the value (for strings) or assigns the specified minimum value (for numbers).
- *Random* - Assigns random values according to schema restrictions.

Generate optional properties

If selected, the JSON instance will be generated with optional properties that are defined in the JSON schema. Otherwise, only the required properties will be generated.

Generate additional content

If selected, the JSON instance will be generated with additional properties that are defined in the JSON schema as `additionalProperties` and additional items that are defined as `additionalItems` (in the case of an Array).

Choice strategy

You can specify the way an instance will be generated from a schema that contains a `CombinedSchema` (with either *oneOf* or *anyOf*). The following options are available:

- *First* - The first defined schema in *oneOf* or *anyOf* will be used.
- *Random* - A random schema defined in *oneOf* or *anyOf* will be used.

Recursion level

This option controls the maximum allowed depth (must be a number), in case the selected schema contains recursive calls of `$ref` schemas referencing one another. By default, it is set to 1, meaning that the generation for the recursive calls will stop after the first iteration.

Open first instance in editor

If selected, the first generated instance is opened in the editor.

You can click **OK** at any point to generate the sample JSON files.

Contextual Menu Actions in JSON Documents

When editing JSON documents, Oxygen XML Developer Eclipse plugin provides the following actions in the contextual menu:

Cut, Copy, Paste

Executes the typical editing actions on the currently selected content.

Copy JSON Pointer

Creates a *JSON Pointer* at the current cursor location and copies the expression that denotes the JSON pointer to the system clipboard.

Copy XPath

Copies the XPath expression of the current property from the current editor to the clipboard.

Go to Matching Bracket

Moves the cursor to the end bracket that matches the start bracket, or vice versa.

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.

Transforming and Querying JSON Documents

Oxygen XML Developer Eclipse plugin provides the ability to transform JSON documents to XML or HTML through XSLT or XQuery processing. You also have access to some powerful tools for querying JSON through XPath expressions or XQuery.

For more information about transforming and querying in JSON, watch our video demonstration:

https://www.youtube.com/embed/1LHoMhEFagA

Transforming JSON Documents with XSLT

It is possible to transform JSON documents through XSLT processing. To do so, follow these steps:

1. Create an XSLT 3.0 stylesheet that has the `xsl:initial-template`. You can use one of the following two templates available in the New Document Wizard.
   - **XSLT Stylesheet for JSON** - Processes a JSON document by using a `json-doc()` function and matches the JSON properties from the JSON map.
   - **XSLT Stylesheet for JSON to XML** - Processes a JSON document by using a `json-to-xml()` function and matches the converted XML content.
2. Create a new **XSLT transformation** scenario for your stylesheet.
3. Reference the JSON document that you want to transform using one of these two methods:
   - In the transformation scenario, click the **Parameters** button in the **XSLT** tab and add a parameter that specifies the URL to your JSON document in its value. For example, if you are transforming one of the built-in templates mentioned above, the `input` parameter is added by default and you could specify the URL in its value.
   - Specify the URL to your JSON document in the stylesheet you created. For example, if you use one of the built-in templates mentioned above, you would specify the URL in the value of the `input` parameter (in the `xsl:param` element).
4. Run the transformation.

Tip: There are some sample files in the `[OXYGEN_INSTALL_DIR]/samples/json/transform` folder that can be used to transform a JSON document to XML or HTML.

Related Information:

Blog: Transforming JSON

XSLT Functions on JSON Data

Transforming JSON Documents with XQuery

It is possible to transform JSON documents through XQuery processing. To do so, follow these steps:

1. Create an XQuery file.
2. Create a new **XQuery transformation** scenario for your XQuery file.
3. Reference the JSON document that you want to transform using one of these two methods:
    - In the transformation scenario, click the **Parameters** button in the **XQuery** tab and add a parameter that specifies the URL to your JSON document in its value.
    - Specify the URL to your JSON document in the XQuery file you created.
4. Run the transformation.

Tip: There is a sample XQuery file in the `[OXYGEN_INSTALL_DIR]/samples/json/transform` folder that can be used to transform a JSON document.

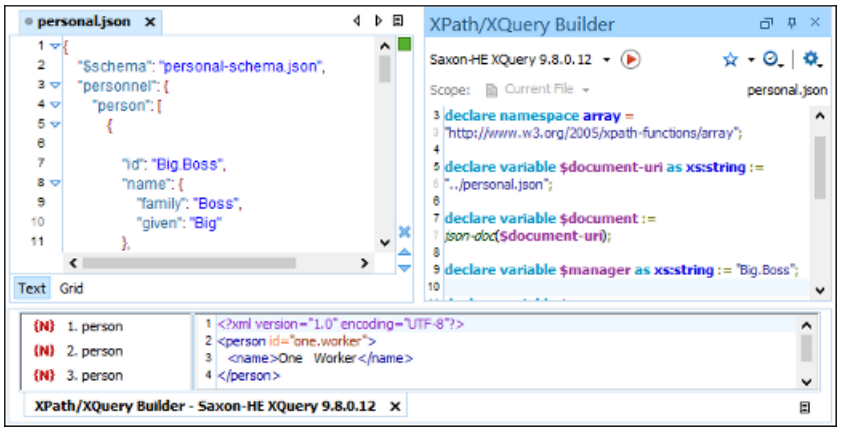
Querying JSON Documents with XPath or XQuery

Oxygen XML Developer Eclipse plugin provides a dedicated **XPath/XQuery Builder** view that allows you to compose complex XPath or XQuery expressions and execute them over JSON documents.

XPath/XQuery Builder View

You can also use the **XPath/XQuery** view to run XPath and XQuery expressions over a JSON document. For XQuery, you need to reference the JSON document in your XQuery content. For more information about this view, see XPath Builder View *(on page 1119)*.

Figure 250. XPath/XQuery Builder View for JSON



Details About Querying JSON Documents Using XPath Expressions

To execute XPath expressions over a JSON document, the document is converted to XML and the XPath is executed over the converted XML document. For this conversion, Oxygen XML Developer Eclipse plugin uses the built-in JSON to XML Converter tool *(on page 570)*. The results are mapped back to the original JSON document.

For example, if you have the following JSON document:

```
{
   "personnel": {
    "person": [
      {"name": "Boss"},
```

```
      {"name": "Worker"}
    ]
  },
  "id":"personnel-id"
}
```

and you want to match the name of the second person, the XPath expression would look like this:

```
/JSON/personnel/person[2]/name
```

The reason why the first element is `JSON` is because if the JSON document contains more than one property on the first level, the converted XML document will have an additional root element called `<JSON>`. For more information, see JSON to XML Conversion Details *(on page 571)*.

The `[2]` in the expression represents the index of the `person` in the array and in this case, it matches the second `person` because the index counting starts with 1.

Editing JSON Schema Documents

The purpose of the JSON schema is to define the legal properties and values of a JSON document to keep it valid and well formed. When editing JSON Schema, all of the same features in the JSON editor *(on page 550)* are available.

Oxygen XML Developer Eclipse plugin includes a new document template to help you get started creating a JSON Schema document. The template is called **JSON Schema** and it can be found in the **New Document** folder in the **New from templates** wizard *(on page 177)*. You can also customize your own JSON Schema templates *(on page 179)* and specify other versions (draft 04, 06, or 07).

For information about using a JSON Schema to validate documents, see Validating JSON Documents Against JSON Schema or Schematron *(on page 553)*

For information about how to associate a JSON schema for the purposes of validation, see Associating a Schema Through a Validation Scenario *(on page 560)*.

Validating JSON Schema Documents

A *valid* JSON Schema document is a *well-formed* document that also conforms to the JSON meta-schema rules that defines the legal syntax of a JSON Schema document.

If a JSON document includes a meta-schema URL in the document root with the **"$schema"** key, the file will be validated as a JSON Schema against the specified meta-schema.

Quick Reference

- If there is a **"$schema": "http://json-schema.org/draft-04/schema"** property in the schema root, then Draft 4 will be used.
- If there is a **"$schema": "http://json-schema.org/draft-06/schema"** property in the schema root, then Draft 6 will be used.

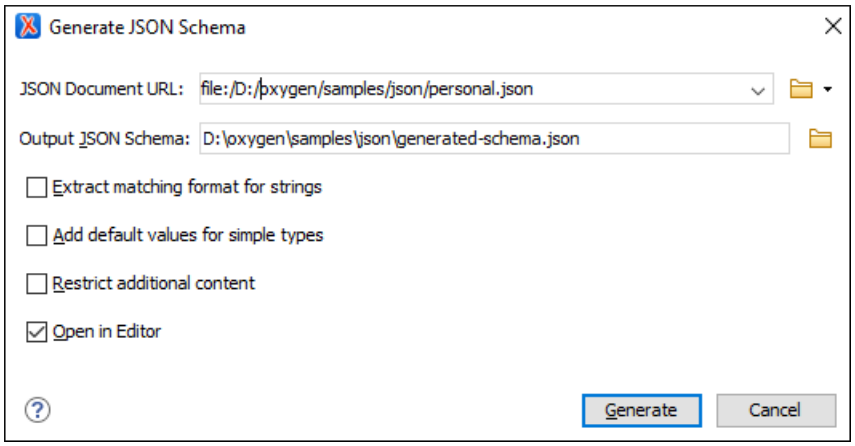
- If there is a **"$schema": "http://json-schema.org/draft-07/schema"** property in the schema root, then Draft 7 will be used.
- If there is a **"$schema": "http://json-schema.org/draft/2019-09/schema"** property in the schema root, then the Draft 7 compatibility mode will be used.
- If there is a **"$schema"** property in the schema root, but with a different draft value, then an error will be displayed (*"no meta-schema is known with URI"*).
- If none of these are found, then it is validated as a simple JSON instance.
- You could also select the **JSON Schema Validator** in a JSON validation scenario *(on page 557)* and it will use the version specified in the JSON Schema, or if a version is not specified, the JSON Schema draft-04 will be used.

For information about how to associate a JSON schema for the purposes of validation, see Associating a Schema Through a Validation Scenario *(on page 560)*.

Generating JSON Schema from a JSON File

Oxygen XML Developer Eclipse plugin includes a tool for generating a sample JSON Schema from a JSON file. To generate a sample JSON Schema, select **Generate JSON Schema** from the **XML Tools > JSON Tools** menu. The action opens a dialog box where you can configure a variety of options for generating the JSON Schema file.

Figure 251. Generate JSON Schema Dialog Box



The **Generate JSON Schema** dialog box includes the following fields and options:

JSON Document URL

The URL of the JSON file. You can specify the path by using the text field, the history drop-down menu, or the browsing actions in the **Browse** drop-down list.

Output JSON Schema

Path to the folder where the generated JSON Schema will be saved.

Extract matching format for strings

If selected, the generator will try to find a format that matches the string values from the JSON Document.

Add default values for simple types

If selected, the *default* values (*0* for number, *""* for string, *false* for boolean) and *examples* for string will be added.

Restrict additional content

If selected, *additionalProperties* (for objects) and *additionalItems* (for arrays) will be set to *false* in the resulting schema. By default, these keys are not in the schema, meaning that providing additional content (according to the schema) is allowed.

Open in Editor

If selected, the generated JSON Schema is opened in the editor.

You can click **Generate** at any point to generate the JSON Schema.

Editing StratML Documents

Strategy Markup Language (StratML) is an XML vocabulary and schema for strategic plans. Oxygen XML Developer Eclipse 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 *(on page 240)*. Oxygen XML Developer Eclipse 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 Eclipse plugin provides the following support for editing XLIFF documents:

XLIFF Version 1.2, 2.0, and 2.1 Support:

- New document 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/schemas/1.2`, for version 2.0 in *`[OXYGEN_INSTALL_DIR]`*`/frameworks/`

`xliff/schemas/2.0`, and for version 2.1 in *`[OXYGEN_INSTALL_DIR]`*`/frameworks/xliff/schemas/2.1`.

XLIFF Version 2.0 and 2.1 Enhanced Support:

Support for validating XLIFF 2.0 and 2.1 documents using modules. For version 2.0, the default modules are stored in *`[OXYGEN_INSTALL_DIR]`*`/frameworks/xliff/schemas/2.0/modules` and for version 2.1, they are stored in *`[OXYGEN_INSTALL_DIR]`*`/frameworks/xliff/schemas/2.1`.

Editing XLIFF Documents in Author Mode

By default, when you create a new XLIFF document from a template *(on page 177)*, Oxygen XML Developer Eclipse plugin opens it in **Text** mode. Aside from the normal editing features found in **Text** mode, you can also switch to **Author** mode where Oxygen XML Developer Eclipse plugin offers some special form controls specifically for XLIFF documents. These form controls simply allow you to add or edit XLIFF attribute values and content in a visual mode.

For XLIFF version 2.0 and 2.1 documents, you can also change the style of the visual editing mode. The **Styles** drop-down menu on the toolbar offers the following styles that are specifically designed to render XLIFF 2.0 and 2.1 documents in **Author** mode:

- **Default**
- **Classic**
- **Translate**

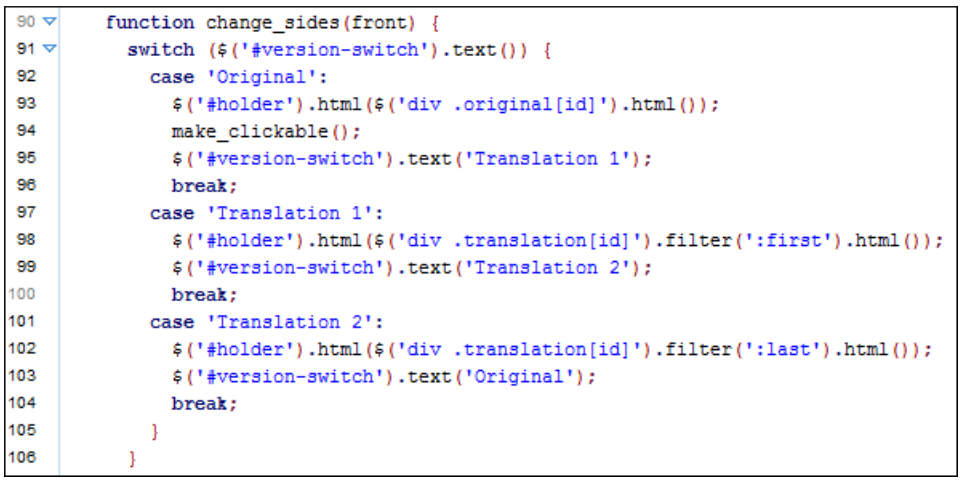
Editing JavaScript Documents

This section explains the features of the Oxygen XML Developer Eclipse plugin JavaScript Editor and how you can use them.

JavaScript Editing Actions

Oxygen XML Developer Eclipse 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.

Figure 252. 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 Eclipse 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 Eclipse 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 Eclipse 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 (on page 1324)* presents you a list of the elements you can insert at the cursor position. It can be manually activated with the **Ctrl + Space (Command + Space on OS X)** shortcut.

For an enhanced assistance, JQuery methods are also presented. The following icons decorate the elements in the content completion list of proposals depending on their type:

- f() - function
- v - variable
- ▪ - object
- ● - property
- f() - method

Note: These icons decorate both the elements from the content completion list of proposals and from the **Outline** view *(on page 587)*.

Figure 253. 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.

Syntax Highlighting in JavaScript Documents

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for JavaScript files, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
3. Select and expand the **JavaScript** 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.

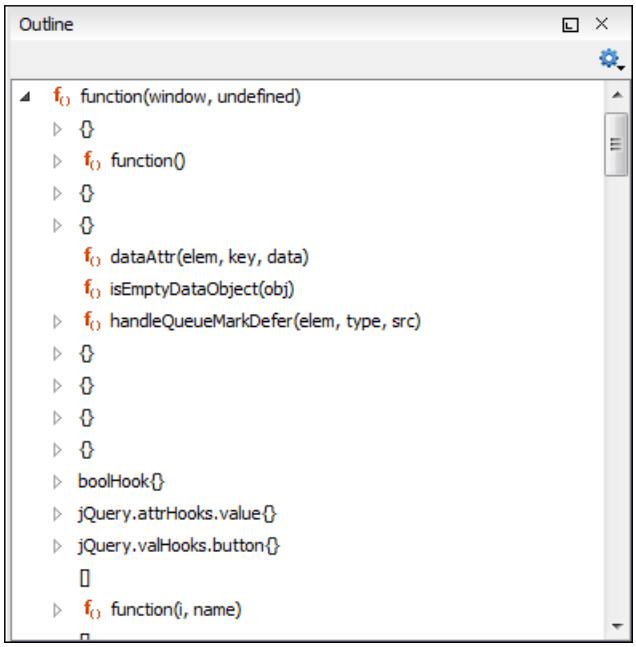
Related Information:

Syntax Highlight Preferences *(on page 101)*

JavaScript Outline View

Oxygen XML Developer Eclipse 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 by selecting it from the **Window > Show View** menu.

Figure 254. JavaScript Outline View



The following icons decorate the elements in the **Outline** view depending on their type:

- f() - function
- v - variable
- ▫ - object
- ● - property
- f() - method

The contextual menu of the JavaScript **Outline** view contains the usual **Cut**, **Copy**, **Paste**, and **Delete** actions. From the **Settings** menu, you can select the **Update selection on cursor move** option to synchronize the **Outline** view with the editing area.

Editing XProc Scripts

XProc is an XML pipeline language that can be used to script transformations. An XProc script is edited as an XML document that is validated against a RELAX NG schema, or if the script has an associated transformation scenario, then the XProc engine selected in the scenario is used as the validating engine (if the XProc engine supports validation). The default engine for XProc scenarios is a version of the *Calabash* engine that comes bundled with Oxygen XML Developer Eclipse plugin version 22.1.

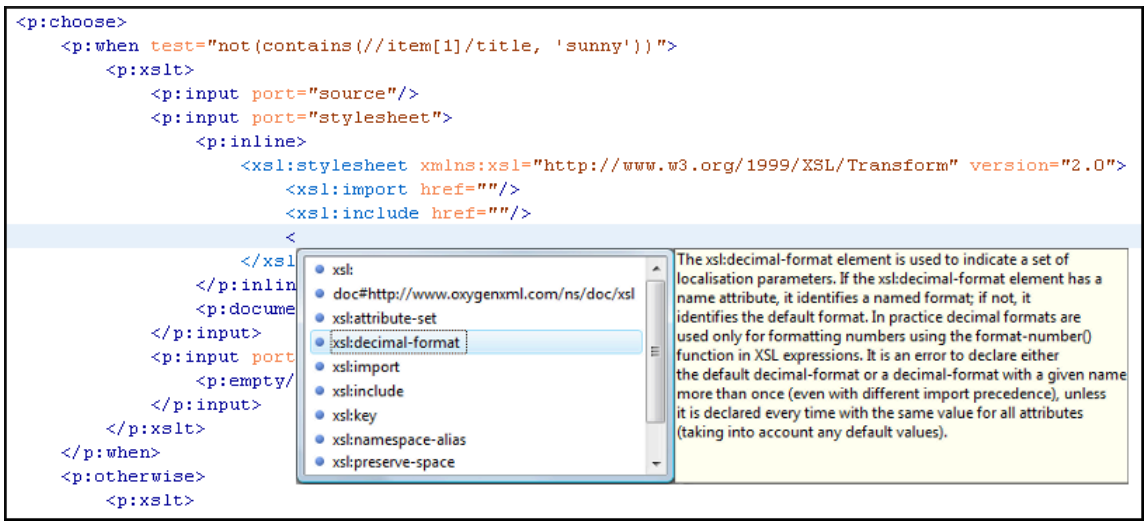
XProc Content Completion

Oxygen XML Developer Eclipse plugin helps you edit a XProc scripts through the *Content Completion Assistant (on page 1324)*, offering proposals that are valid at the cursor position. It can be manually activated with the **Ctrl + Space (Command + Space on OS X)** shortcut.

The content completion inside the `<input/inline>` element 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 `<inline>` XProc element, 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.

Figure 255. XProc Content Completion



XProc Syntax Highlighting

The XProc editor assists you in writing XPath expressions by offering dedicated coloring schemes for syntax highlighting.

To customize the colors or styles used for the syntax highlighting colors for XProc, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
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 **XML** tab in the **Preview** pane to see the effects of your changes.

Enabling Extensions in Calabash

If you are using the default Calabash engine, it is possible to configure extensions (for a list of the valid extensions, see http://xmlcalabash.com/docs/reference/cfg.extension.html).

To configure an extension:

1. Edit the following file: `OXYGEN_INSTALL_DIR/lib/xproc/calabash/engine.xml`.
2. Add the extension and its value as a `system-property`, as in the following example:

```
<system-property name="com.xmlcalabash.allow-text-results" value="true"/>
```

Related Information:

Creating an XProc Transformation Scenario *(on page 749)*

Integrating an External XProc Engine *(on page 753)*

XProc Preferences *(on page 128)*

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.

There are numerous online resources out there to help you get started with writing Schematron rules. Here are just a few that might help you:

- Guide to Schema Writing with Schematron
- Presentation: Schematron Development with Oxygen

Oxygen XML Developer Eclipse 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 *(on page 597)*. You can create a new Schematron schema using one of the Schematron templates available in the **New from Templates** wizard *(on page 177)*.

For information about applying and detecting Schematron schemas, see Associating a Schema to XML Documents *(on page 303)*.

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 *(on page 591)* 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 *(on page 36)*, go to **XML > XML Parser > Schematron**, and select it in the **Embedded rules query language binding** option *(on page 125)*.
- 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 *(on page 36)*, go to **XML > XML Parser > Schematron**, and select the version in the **XPath Version** option *(on page 126)*.

Multi-Lingual Support in Schematron Messages

You can specify the desired language for the validation messages in the **Schematron Preferences** page *(on page 124)*. The Schematron validation messages can be presented in multiple languages by defining the language for each message using the Schematron `<diagnostics>` element. 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 color schemes. To customize the coloring schemes, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Syntax Highlight**.

Schematron Transformation Scenario

When you create a Schematron document, Oxygen XML Developer Eclipse 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.

For more information about the Schematron support in Oxygen XML Developer Eclipse plugin, watch our video demonstrations:

https://www.youtube.com/embed/HdcZA3DJi7E

https://www.youtube.com/embed/y3u3wlO92e4

https://www.youtube.com/embed/FQNSsg57S4E

Related Information:

Editing XML Documents in Text Mode *(on page 222)*

Associating a Schema to XML Documents *(on page 303)*

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 Eclipse plugin provides the support for defining the main module (or modules), thus allowing you to edit any of the imported/included schema files in the context of the larger schema structure.

You cat set a main Schematron document either using the *master files* support from the **Navigator** view *(on page 200)*, 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 Eclipse 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 (on page 1324)* displays all the referable components valid in the current context. This include components defined in modules other than the currently edited one.

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 Eclipse 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 Eclipse 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 *information*. 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 Eclipse 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 Eclipse 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 Information:

Validating XML Documents Against a Schema *(on page 279)*

Validation Scenario *(on page 287)*

Associating a Schema to XML Documents *(on page 303)*

Presenting Validation Errors in Text Mode *(on page 281)*

Integrating Schematron Rules in a Framework and Sharing Them

Custom **Schematron** rules are a great way to ensure consistency for XML authoring, especially when there is a large team working on the same set of documents. You can use **Schematron** for numerous use cases. For example, to restrict certain elements from being used, to impose restrictions on the amount of text for an element, or to impose restrictions on certain elements based on various attribute values or text content set in other elements. Furthermore, you can define quick fixes for each **Schematron** rule *(on page 607)* to offer technical writers proposed solutions for reported problems.

Once you define the **Schematron** rules, they can be shared with the other members of your team by integrating them in a *framework (on page 1325)* (document type) configuration.

How to Integrate Schematron Rules in a Framework

To integrate a Schematron rule in an existing framework bundled with the application, follow these steps:

1. Create a folder structure for an extended framework and save it somewhere on disk where you have full write access (for example, `custom_frameworks/dita-extension`).
2. In that new folder structure, create another folder that will contain all of your custom Schematron files (for example, `custom_frameworks/dita-extension/rules`).
3. Define the Schematron rules in an existing or new Schematron file and save it in the folder you created in step 2. There are numerous online resources out there to help you get started with writing Schematron rules. Here are just a few that might help you:
   - Guide to Schema Writing with Schematron
   - Presentation: Schematron Development with Oxygen
4. Open the **Preferences** dialog box *(on page 36)* and go to **Document Type Association > Locations** *(on page 51)*. In this preferences page, add the path to your `custom_frameworks` folder in the **Additional frameworks directories** list, then click **OK** or **Apply** to save your changes.
5. Go to the **Document Type Association** preferences page *(on page 50)* and select a *framework* configuration (for example, **DITA**) and use the **Extend** button to create an extension for it.
6. Give the extension an appropriate name (for example, *DITA - Custom*), select **External** for the **Storage** option, and specify an appropriate path to your framework configuration file (for example, `path/to/.../custom_frameworks/dita-extension/dita-extension.framework`).
7. Make whatever changes you desire to the extension, then go to the **Validation** tab, edit the default validation scenario (select the scenario and click the **Edit** button), and add an extra validation unit to it (one that uses your custom Schematron file).
8. Click **OK** to close the dialog box and then **OK** or **Apply** to save the changes to the **Document Type Association** preferences page *(on page 50)*.
9. Open an XML document that matches your framework configuration and test the new rule.
10. You can continue to refine the Schematron and develop additional rules as needed.

Sharing Schematron Rules

To share Schematron rules with other members of your team, you simply need to share the framework where you integrated the Schematron rules.

Related Information:

Defining Schematron Quick Fixes *(on page 607)*

Associating a Schema in Validation Scenarios Defined in the Document Type *(on page 308)*

Validating Schematron Documents

By default, a Schematron schema is validated as you type. To change this, open the **Preferences** dialog box *(on page 36)*, go to **Editor > Document Checking**, and deselect the **Enable automatic validation** option *(on page 84)*.

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 Eclipse 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 Eclipse 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 *(on page 591)*

Content Completion in Schematron Documents

Oxygen XML Developer Eclipse plugin helps you edit a Schematron schema through the *Content Completion Assistant (on page 1324)*, offering proposals that are valid at the cursor position. It can be manually activated with the **Ctrl + Space (Command + Space on OS X)** shortcut.

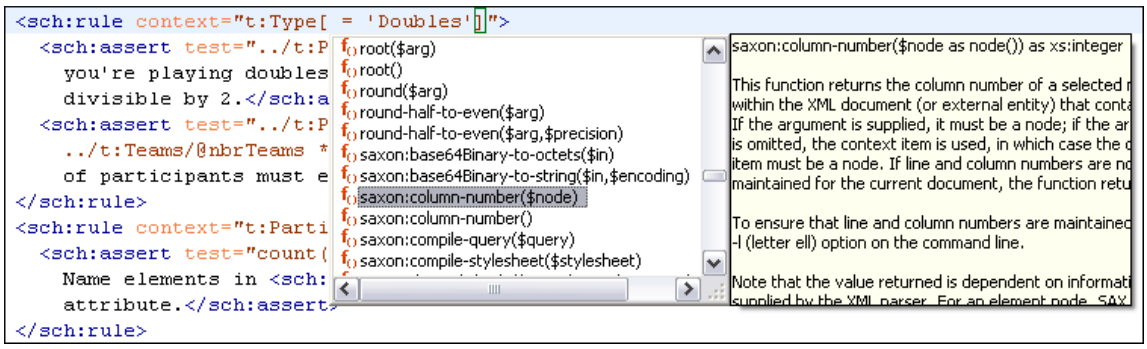
When you edit the value of an attribute that refers a component, the proposed components are collected from the entire schema hierarchy. For example, if the editing context is `phase/active/@pattern`, the *Content Completion Assistant* proposes all the defined patterns.

Note: For Schematron resources, the *Content Completion Assistant* collects its components starting from the *master files (on page 1326)*. 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 *(on page 200)*.

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 *(on page 124)* that are set in Preferences pages. If the Saxon 6.5.5 namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 9.9.1.5 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 256. 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 *(on page 238)* into Schematron documents.

Syntax Highlighting in Schematron

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for Schematron schemas, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
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 **XML** tab in the **Preview** pane to see the effects of your changes.

Tip: Oxygen XML Developer Eclipse plugin also allows you to specify syntax highlighting colors for specific XML elements and attributes with specific namespace prefixes. This can be done in the **Editor > Syntax Highlight > Elements/Attributes by Prefix** preferences page *(on page 101)*.

Related Information:

Syntax Highlight Preferences *(on page 101)*

Embedding Schematron Rules in XML Schema or RELAX NG

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 Eclipse plugin supports Schematron validation schemas and it is able to extract and use the embedded rules.

Validating XML Documents with XML Schema and Embedded Schematron

To validate an XML document with XML Schema and its embedded Schematron, you can associate the document like this:

```
<?xml-model href="percent.xsd" type="application/xml"
            schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

Validating XML Documents with Relax NG and Embedded Schematron

To validate an XML document with RELAX NG schema and its embedded Schematron rules, you need to associate the document with both schemas like this:

```
<?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.

Note: When you work with XML Schema or Relax NG documents that have embedded Schematron rules Oxygen XML Developer Eclipse 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 XML Schema

```
<xsd:appinfo>
  <sch:pattern>
    <sch:rule context="...">
 <sch:assert test="...">Message.</sch:assert>
    </sch:rule>
  </sch:pattern>
</xsd:appinfo>
```

Example: Embedded Schematron in Relax NG Schema

```
<grammar
    xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:sch="http://purl.oclc.org/dsdl/schematron" >
    <sch:pattern>
      <sch:rule context="...">
        <sch:assert test="...">Message.</sch:assert>
      </sch:rule>
    </sch:pattern>
    <start>
     ..............
    </start>
```

```
</grammar>
```

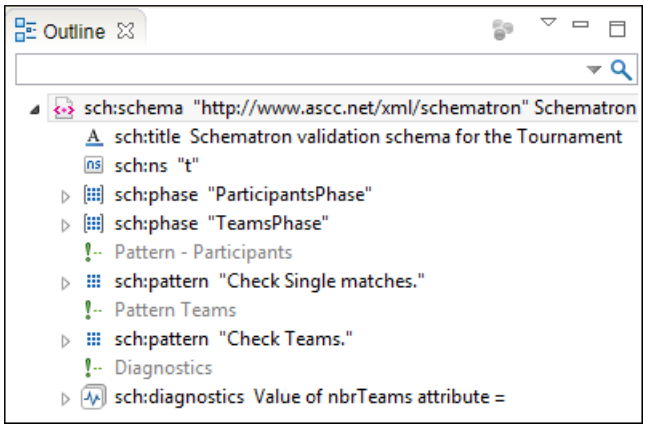
Related Information:

Embedding Schematron Quick Fixes in Relax NG or XML Schema *(on page 620)*

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 by selecting it from the **Window > Show View** menu.

Figure 257. 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 *(on page 144)*.

Configure displayed attributes

Displays the XML Structured Outline preferences page *(on page 144)*.

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.

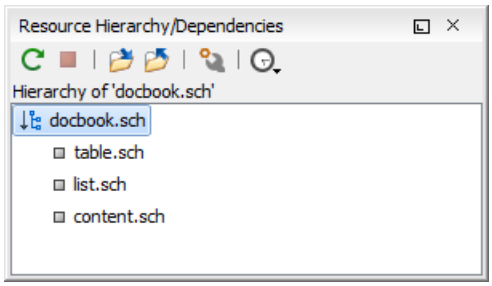
The upper part of the **Outline** view contains a filter box that allows you to focus on the relevant components. Type a text fragment in the filter box and only the components that match it are presented. For advanced usage you can use wildcard characters (such as `*` or `?`) and separate multiple patterns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

Schematron Resource Hierarchy/Dependencies View

The **Resource Hierarchy/Dependencies** view displays the hierarchy or dependencies for resources included in a Schematron schema. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

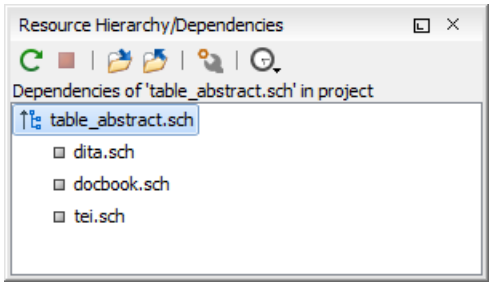
If you want to see the hierarchy of a schema, select the desired schema in the **Navigator** view *(on page 192)* and choose **Show Resource Hierarchy** from the contextual menu.

Figure 258. Resource Hierarchy



If you want to see the dependencies of a schema, select the desired schema in the **Navigator** view *(on page 192)* and choose **Show Resource Dependencies** from the contextual menu.

Figure 259. Resource Dependencies



The following actions are available on the toolbar of 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 dependencies search scope

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 for a resource listed in the **Resource Hierarchy/Dependencies** view contains the following actions:

Open

Opens the resource. You can also double-click a resource in the Hierarchy/Dependencies structure to open it.

Go to reference

Opens the source document where the resource is referenced.

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 *(on page 200)*.

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 *(on page 600)*.

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 of the renamed resource(s)** - Select this option to update the references to the resource you are renaming. A **Preview** option is available that allows you to see what will be updated before selecting **Rename** to process the operation.

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)** - Select this option to update the references to the resource you are moving, in accordance with the new location and name. A **Preview** option is available that allows you to see what will be updated before selecting **Move** to process the operation.

Highlight Component Occurrences in Schematron Documents

When you position your mouse cursor over a component in a Schematron document, Oxygen XML Developer Eclipse 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 *(on page 36)* 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 *(on page 248)*.

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 currently 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 currently 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:

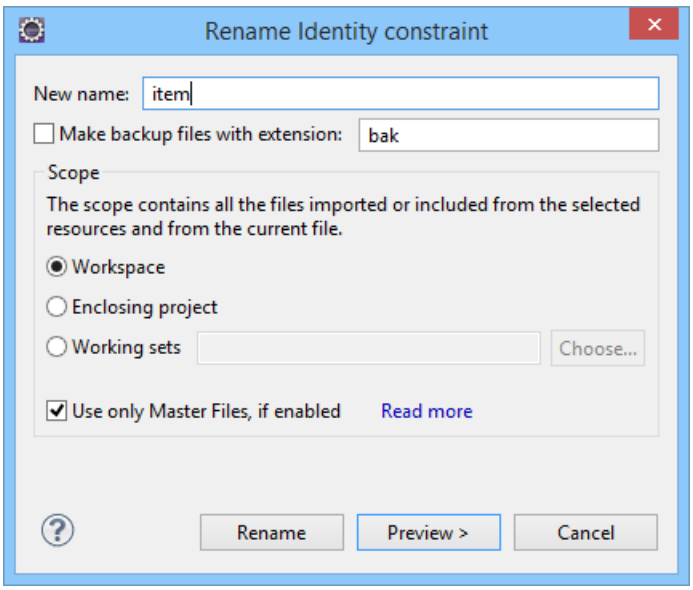
Rename Component

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.

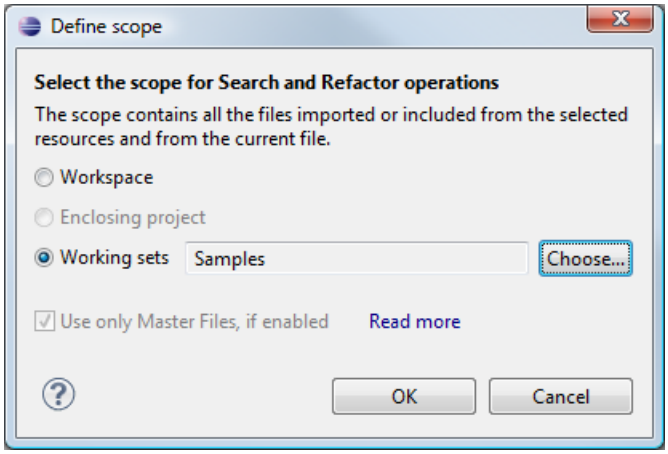
Figure 260. Rename Identity Constraint Dialog Box



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 (on page 1329)*. 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 *(on page 200)*.

Figure 261. Define Scope Dialog Box



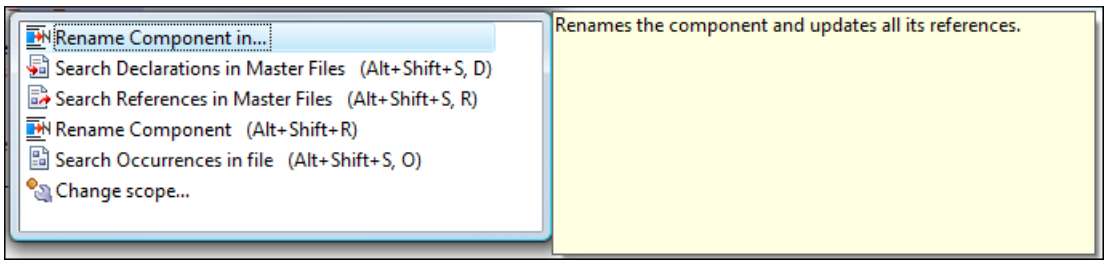
The scope you define is applied to all future search and refactor operations until you modify it. Contextual menu actions allow you to add or delete files, folders, and other resources to the *working set (on page 1329)* structure.

Quick Assist Support in Schematron Documents

The *Quick Assist* support *(on page 1328)* improves the development work flow, offering fast access to the most commonly used actions when you edit schema documents.

The *Quick Assist* feature *(on page 1328)* 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 262. 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.

Schematron Unit Test (XSpec)

XSpec is a behavior driven development (BDD) *framework* for XSLT, XQuery, and Schematron. XSpec consists of syntax for describing the behavior of your XSLT, XQuery, or Schematron code, and some code that enables you to test your code against those descriptions.

Creating a Schematron Unit Test

To create a Schematron Unit Test, go to **File > New > Schematron Unit Test**. This is simple document template to help you get started.

Running a Schematron Unit Test

To run a Unit Test, open the XSpec file in an editor and click **Apply Transformation Scenario(s)** on the main toolbar. This will run the built-in **Run XSpec Test** transformation scenario that is defined in the XSpec *framework (on page 1325)*.

Testing a Stylesheet

An XSpec file contains one or more test scenarios.

Example

Suppose you have this Schematron rule that says sections should have a title:

```
<sch:pattern>
  <sch:rule context="section">
    <sch:assert test="title" id="a002">
      section should have a title
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

The XSpec test could look like this:

```
<x:description xmlns:x="http://www.jenitennison.com/xslt/xspec" schematron="demo-01.sch">
   <x:scenario label="section should have a title">
     <x:context>
       <article>
         <section>
           <title>Introduction</title>
           <p>This is an example.</p>
         </section>
         <section>
           <p>This is an example.</p>
         </section>
       </article>
     </x:context>
```

```
      <x:expect-not-assert id="a002" location="/article[1]/section[1]"/>
      <x:expect-assert id="a002" location="/article[1]/section[2]"/>
    </x:scenario>
</x:description>
```

The `<sch:assert>` with the `id="a002"` is not expected to be triggered on the first section since it includes a title. This requirement is expressed with the `<x:expect-not-assert>` element.

Since the second section does not have a title, you would expect the Schematron rule to be triggered and this requirement is expressed with the `<x:expect-assert>` element.

For more details about how to write Schematron tests and various samples, see https://github.com/xspec/xspec/wiki/Writing-Scenarios-for-Schematron#writing-tests.

Adding a Catalog to an XSpec Transformation

If your Schematron needs a catalog, you can add one to the XSpec transformation by doing one of the following:

- If you are using a project *(on page 191)* in Oxygen XML Developer Eclipse plugin, create a `catalog.xml` file in the project directory. This catalog will then be loaded automatically.
- Edit *(on page 762)* the **Run XSpec Test** transformation scenario, go to the **Parameters** tab *(on page 731)*, and set the value of the `catalog` parameter to the location of your catalog file.

Editing Schematron Quick Fixes

Oxygen XML Developer Eclipse plugin provides support for editing the Schematron *Quick Fixes (on page 303)*. 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 Eclipse 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 303)*.

Related Information:

Oxygen XML Blog: Schematron Checks to Help Technical Writing

Schematron Quick Fixes (SQF)

Oxygen XML Developer Eclipse plugin provides support for Schematron *Quick Fixes (on page 1328)* (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.

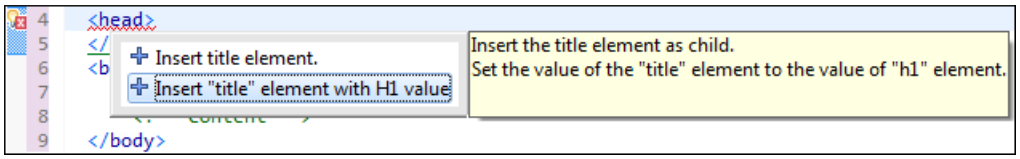
A typical use case is using Schematron *Quick Fixes* to assist content authors with common editing tasks. For example, you can use Schematron rules to automatically report certain validation warnings

(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, see the following blog post: https://blog.oxygenxml.com/topics/SchematronBCs.html.

Displaying the Schematron Quick Fix Proposals

The defined Schematron *Quick Fixes* are displayed on validation errors in **Text** mode.

Figure 263. Example of a Schematron Quick Fix



Related Information:

- Editing Schematron Quick Fixes *(on page 606)*
- Schematron Quick Fix Specifications
- Presenting Schematron Validation Issues *(on page 591)*

Defining Schematron Quick Fixes

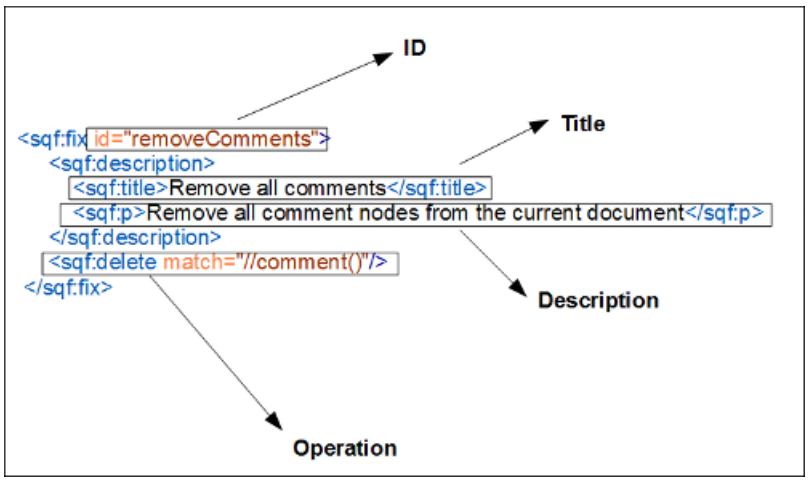
You can define and customize Schematron *Quick Fixes (on page 1328)* directly in the current Schematron file or in a separate Schematron file. The Schematron *Quick Fixes* are an extension of the Schematron language and they allow you to define fixes for Schematron error messages. You can reference the *Quick Fixes* using the `@sqf:fix` attribute inside the `<assert>` or `<report>` elements (for example: `<assert test="title" sqf:fix="removeComments">Remove comments</assert>`).

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 `<sqf: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 `<sqf:title>` element.
- **Description** - Defined by the text in the paragraphs (`<sqf:p>`) of the `<sqf:description>` element.
- **Operations** - The following basic types of operations (elements) *(on page 609)* 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 264. Schematron Quick Fix Components



The assertion message that generates the *Quick Fix* is added as the `<sqf:description>` of the problem to be fixed. The `<sqf:title>` is presented as the name of the *Quick Fix*. The content of the paragraphs (`<sqf:p>`) within the `<sqf: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 613)*.

<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:copy-of>

Used to copy the selected nodes that are specified by the `@select` attribute. The element with its attribute is treated as an `xsl:copy-of` with a `@select` attribute, as defined in the XSLT specification.

<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 Eclipse 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 (on page 1328)*: **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 Eclipse 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` attribute as "element", and specify the element *QName (on page 1328)* 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` attribute as "attribute", and specify the attribute *QName (on page 1328)* 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` attribute 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 `<sqf: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` attribute as "comment".
- **Insert Processing Instruction** - To insert a processing instruction, use the `<sqf:add>` element, set the value of the `@node-type` attribute as "pi" or "processing-instruction", and specify the name of the processing instruction in the `@target` attribute.

Delete

The `<sqf:delete>` element allows you to remove any type of node (such as elements, attributes, text, comments, or processing instructions). To specify nodes for deletion, the `<sqf:delete>` element can include a `@match` attribute that is an XPath expression (the default value is `.`). If the `@match` attribute is not included, it deletes the context node of the Schematron rule.

An Example of the `<sqf:delete>` Element:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2"
    xmlns:sqf="http://www.schematron-quickfix.com/validator/process">
    <pattern>
        <rule context="*[@xml:lang]">
            <report test="@xml:lang" sqf:fix="remove_lang">
                The attribute "xml:lang" is forbidden.</report>
            <sqf:fix id="remove_lang">
                <sqf:description>
                    <sqf:title>Remove "xml:lang" attribute</sqf:title>
                </sqf:description>
                <sqf:delete match="@xml:lang"/>
            </sqf:fix>
        </rule>
    </pattern>
</schema>
```

Replace

The `<sqf:replace>` element allows you to replace nodes. Similar to the `<sqf:delete>` element, it can include a `@match` attribute. Otherwise, it replaces the context node of the rule. The `<sqf:replace>`

element has three tasks. It identifies the nodes to be replaced, defines the replacing nodes, and defines their content.

An Example of the `<sqf:replace>` Element:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
    xmlns:sqf="http://www.schematron-quickfix.com/validator/process"
 queryBinding="xslt2">
    <pattern>
        <rule context="title">
            <report test="exists(ph)" sqf:fix="resolvePh" role="warn">
                ph element is not allowed in title.</report>
            <sqf:fix id="resolvePh">
                <sqf:description>
                    <sqf:title>Change the ph element into text</sqf:title>
                </sqf:description>
                <sqf:replace match="ph">
                    <value-of select="."/>
                </sqf:replace>
            </sqf:fix>
        </rule>
    </pattern>
</schema>
```

Other Attributes for Replace Operations:

- **node-type** - Determines the type of the replacing node. The permitted values include:
  - **keep** - Keeps the node type of the node to be replaced.
  - **element** - Replaces the node with an element.
  - **attribute** - Replaces the node with an attribute.
  - **pi** - Replaces the node with a processing instruction.
  - **comment** - Replaces the node with a comment.
- **target** - By using a *QName (on page 1328)* 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.
- **flags** - Specifies flags to control the interpretation of the regular expression (given in the `@regex` attribute).

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(., 'Oxygen', 'i')"
 sqf:fix="changeWord">The oXygen word is not allowed</sch:report>
            <sqf:fix id="changeWord">
                <sqf:description>
                    <sqf:title>Replace word with product</sqf:title>
                </sqf:description>
                <sqf:stringReplace regex="Oxygen" flags="i"><ph keyref="product"/>
                </sqf:stringReplace>
            </sqf:fix>
        </sch:rule>
    </sch:pattern>
</sch:schema>
```

Related Information:

User Entry SQF Operation *(on page 613)*

Restricting Quick Fix Operations *(on page 613)*

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 Eclipse plugin will display a dialog box for each one where 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>`. Note that the `@default` attribute defines a default value for the operation by using an XPath expression (as in the example below) and its value will be presented in the user entry dialog box.

An Example of the `<sqf:user-entry>` Element:

```
<sqf:fix id="editTitle">
    <sqf:description>
        <sqf:title>Edit the journal title</sqf:title>
    </sqf:description>
    <sqf:user-entry name="newTitle" default="@title">
        <sqf:description>
            <sqf:title>Edit the title:</sqf:title>
        </sqf:description>
    </sqf:user-entry>
    <sqf:replace match="@title" target="title" node-type="keep" select="$newTitle"/>
</sqf:fix>
```

Related Information:

Basic Schematron Quick Fix Operations *(on page 609)*

Restricting Quick Fix Operations

To restrict a *Quick Fix (on page 1328)* 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 609)*

Formatting/Indenting Content Inserted by SQF Operations

Content that is inserted by the **Add**, **Replace**, or **String Replace** Schematron *Quick Fix (on page 1328)* 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 609)*

Executing Schematron Quick Fixes in Other Documents

You can apply Schematron *Quick Fixes (on page 1328)* 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">
```

Generate Multiple Similar Quick Fixes

You can generate the same Schematron *Quick Fix (on page 1328)* for multiple matches. To do this, you can add the `@use-for-each` attribute inside the `<sqf:fix>` element and for each match of the XPath expression in the value of the `@use-for-each` attribute, a Quick Fix will be presented to the user. The XPath expression does not change the context of the Quick Fix. If you want to access the current match from the XPath expression, you can use the `$sqf:current` variable.

Example:

Suppose you want to restrict the user from entering more than 4 list items in a list. The following example presents an error on any list that has more than 4 list items and offers a Quick Fix with multiple proposals where the user would specify which list item to remove.

```
<sch:rule context="ul">
    <sch:report test="count(li) gt 4" sqf:fix="removeAnyItem">
        The list cannot contain more than 4 entries.
    </sch:report>
    <sqf:fix id="removeAnyItem" use-for-each="1 to count(li)">
        <sqf:description>
            <sqf:title>Remove item #<sch:value-of select="$sqf:current"/></sqf:title>
        </sqf:description>
        <sqf:delete match="li[$sqf:current]"/>
    </sqf:fix>
</sch:rule>
```

Localizing SQF Messages

Oxygen XML Developer Eclipse plugin provides support for presenting Schematron Quick Fix messages in multiple languages. The language used for the SQF messages is the language specified in the '**Message Language** option in the **Schematron** preferences page *(on page 125)*. If you want to provide an alternative message for a specific language, you can reference IDs or key values for the specific alternate text phrase. In Oxygen XML Developer Eclipse plugin, the alternate text phrase is defined in a `<sch:diagnostic>` element and it can be used in conjunction with `<sch:assert>` or `<sch:report>` elements.

Example:

The following example presents a quick fix with a different message depending on whether the user's language is English or German.

```
<sch:rule context="dog">
  <sch:assert test="bone" diagnostics="d_en d_de" sqf:fix="addBone" >
```

```
    A dog should have a bone.</sch:assert>

  <sqf:fix id="addBone">
    <sqf:description>
      <sqf:title ref="fix_en fix_de" xml:lang="en">Add a bone</sqf:title>
      <sqf:p ref="fix_desc_en fix_desc_de" xml:lang="en">Add bone element as child</sqf:p>
    </sqf:description>
    <sqf:add node-type="element" target="bone"/>
  </sqf:fix>
</sch:rule>
....
<sch:diagnostics xml:lang="en">
  <sch:diagnostic id="d_en"> A dog should have a bone. </sch:diagnostic>
  <sch:diagnostic id="fix_en"> Add a bone </sch:diagnostic>
  <sch:diagnostic id="fix_desc_en">Add a bone element as child</sch:diagnostic>
</sch:diagnostics>
<sch:diagnostics xml:lang="de">
  <sch:diagnostic id="d_de"> Ein Hund sollte ein Bein haben. </sch:diagnostic>
  <sch:diagnostic id="fix_de"> Fugen Sie einen Knochen hinzu </sch:diagnostic>
  <sch:diagnostic id="fix_desc_de"> Fugen Sie ein Knochenelement als
    untergeordnetes Element hinzu </sch:diagnostic>
</sch:diagnostics>
```

Integrating SQF in a Framework and Sharing Them

You can use Schematron *Quick Fixes (on page 1328)* to assist your content authors by imposing rules for an entire *framework (on page 1325)* (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 (on page 1325)*, follow these steps:

1. Create a folder structure for an extended framework and save it somewhere on disk where you have full write access (for example, `custom_frameworks/dita-extension`).
2. In that new folder structure, create another folder that will contain all of your custom Schematron files (for example, `custom_frameworks/dita-extension/rules`).
3. Define the Schematron *Quick Fix* for a rule *(on page 607)* in an existing or new Schematron file and save it in the folder you created in step 2.
4. Open the **Preferences** dialog box *(on page 36)* and go to **Document Type Association > Locations** *(on page 51)*. In this preferences page, add the path to your `custom_frameworks` folder in the **Additional frameworks directories** list, then click **OK** or **Apply** to save your changes.
5. Go to the **Document Type Association** preferences page *(on page 50)* and select a *framework* configuration (for example, **DITA**) and use the **Extend** button to create an extension for it.
6. Give the extension an appropriate name (for example, *DITA - Custom*), select **External** for the **Storage** option, and specify an appropriate path to your framework configuration file (for example, `path/to/.../custom_frameworks/dita-extension/dita-extension.framework`).
7. Make whatever changes you desire to the extension, then go to the **Validation** tab, edit the default validation scenario (select the scenario and click the **Edit** button), and add an extra validation unit to it (one that uses your custom Schematron file that includes the SQF).
8. Click **OK** to close the dialog box and then **OK** or **Apply** to save the changes to the **Document Type Association** preferences page *(on page 50)*.
9. 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 308)*.
10. Open a document in your *framework* and test the new rule and *Quick Fix*.
11. You can continue to refine the Schematron and develop additional rules as needed.

Sharing Schematron Quick Fixes

To share Schematron Quick Fixes with other members of your team, you simply need to share the framework where you integrated the SQF.

Related Information:

- Defining Schematron Quick Fixes *(on page 607)*
- Basic Schematron Quick Fix Operations *(on page 609)*
- Associating a Schema in Validation Scenarios Defined in the Document Type *(on page 308)*

Validating Schematron Quick Fixes

By default, Schematron *Quick Fixes (on page 1328)* 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 *(on page 36)*, go to **Editor > Document Checking**, and deselect the **Enable automatic validation** option *(on page 84)*.

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 Information:

- Validating XML Documents Against a Schema *(on page 279)*
- Validation Scenario *(on page 287)*
- Presenting Validation Errors in Text Mode *(on page 281)*

Content Completion in SQF

Oxygen XML Developer Eclipse plugin helps you edit Schematron *Quick Fixes (on page 1328)* embedded in a Schematron document by offering proposals that are valid at the cursor position in a *Content Completion Assistant (on page 1324)*. It can be manually activated with the **Ctrl + Space (Command + Space on OS X)** shortcut.

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 `@path`, `@select`, `@context`, `@subject`, and `@test` attributes, depending on the Schematron options *(on page 124)* that are set in Preferences pages. If the Saxon namespace (`xmlns:saxon="http://icl.com/saxon"`) or the Saxon 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 (on page 1328)* ID in a Schematron document, Oxygen XML Developer Eclipse 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 *(on page 36)* 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 *(on page 248)*.

Searching and Refactoring Operations in SQF

Search Actions

The following search actions can be applied on *Quick Fix (on page 1328)* 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 currently 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 currently 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:

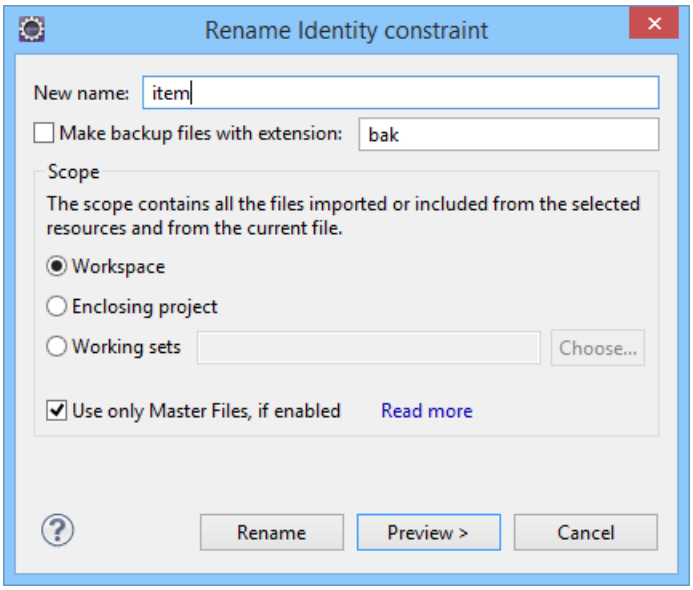
Rename Component

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 265. Rename Identity Constraint Dialog Box



Embedding Schematron Quick Fixes in Relax NG or XML Schema

Schematron *Quick Fixes (on page 1328)* 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 Embedding Schematron Rules in XML Schema or RELAX NG *(on page 595)*.

Oxygen XML Developer Eclipse 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 *(on page 279)*.
2. For the **Schema type**, choose `XML Schema` or `Relax NG`.
3. Select 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

```
<grammar
    xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:sch="http://purl.oclc.org/dsdl/schematron" >
    <sch:pattern>
      <sch:rule context="...">
        <sch:assert test="..." sqf:fix="fixId">Message.</sch:assert>
     <sqf:fix id="fixId">
       .....
     </sqf:fix>
      </sch:rule>
     </sch:pattern>
     <start>
      ..............
     </start>
</grammar>
```

Tip: For more extensive examples, see the samples in the *[OXYGEN_INSTALL_DIR]/samples/schematron* folder.

Related Information:

Embedding Schematron Rules in XML Schema or RELAX NG *(on page 595)*

Defining Schematron Quick Fixes *(on page 607)*

Editing HTML Documents

Oxygen XML Developer Eclipse plugin provides a special framework for editing HTML files (*html* or *htm* file extensions) with a variety of specialized editing features, including validation, content completion, syntax highlighting, HTML-specific actions, and more.

Oxygen XML Developer Eclipse plugin also includes a built-in XHTML framework *(on page 660)* (files with the *http://www.w3.org/1999/xhtml* namespace or with the *xhtml* or *xht* file extension) that has a full set of features (full editing support, document templates, enhanced CSS rendering, specific actions, validation,

content completion, transformation scenarios, and more). Oxygen XML Developer Eclipse plugin also includes support for importing HTML files as an XML document *(on page 1206)*.

For more information about HTML editing support, watch our video demonstration:

https://www.youtube.com/embed/-GObGytf8eY

Related Information:

XHTML Document Type (Framework) *(on page 660)*

HTML Editor

Oxygen XML Developer Eclipse plugin includes a specialized HTML editor and various editing features for files that have the `html` or `htm` file extensions. The encoding is detected automatically based on the value specified in the `@charset` attribute of the `<meta>` element.

Note: If an HTML document has an XHTML namespace, or there is an XSD schema declared, or there is a PUBLIC ID specified in a DOCTYPE, or there is a SYSTEM ID with a value other than *"about:legacy-compat"*, then the document will be opened as an XHTML file.

New Document Template

Oxygen XML Developer Eclipse plugin includes a new document template to help you get started creating HTML content. It is available when creating new documents from templates *(on page 177)* and can be found in the **New Document** folder or by typing *html* in the search field.

Text Mode Editor

You can edit HTML files in the **Text** editing mode *(on page 222)* using all of its useful features *(on page 222)*. It includes content completion *(on page 624)* based on a special HTML schema, syntax highlighting *(on page 624)*, a specialized Outline view *(on page 626)* that presents the structure, folding support *(on page 625)*, and more.

HTML documents support formatting and indenting single or multiple documents to make them more readable. The formatting works even if the document is not XML well-formed and it also works on embedded CSS or JavaScript code. The HTML formatting details are similar to those for XML documents. For details, see Formatting and Indenting XML Documents *(on page 251)*.

HTML-Specific Contextual Menu Actions

There are some specialized actions (available in the contextual menu when you right-click anywhere in the current HTML document) that invoke features unique to HTML documents. These contextual menu actions include:

View in Browser/System Application

Opens the HTML document in your default browser.

Minify HTML

Compresses the HTML document by removing unnecessary white spaces, without affecting the functionality of the document, but significantly reducing the loading time in Web browsers.

HTML to XML Well-formed

Converts the currently edited HTML document to be XML well-formed. This means that unclosed tags will be properly closed, unquoted attribute values will be quoted, and more. This is helpful if, for example, you use XSLT stylesheets while applying transformations on HTML documents (since the transformation will require the HTML document to be XML well-formed).

HTML Validation

Oxygen XML Developer Eclipse plugin includes a built-in default validator used for validating HTML documents. It is based upon the W3C HTML Validator and the HTML documents are validated against the W3C HTML5 specification. The validator in Oxygen XML Developer Eclipse plugin only supports HTML5 structure. It presents the errors in the editor similar to XML documents *(on page 281)*. It also checks the embedded CSS content and the warnings and errors are presented similar to the CSS editor *(on page 522)*.

Validating HTML Against a Schematron

It is also possible to validate HTML documents against a Schematron schema. Besides the default HTML validator, Oxygen XML Developer Eclipse plugin also includes a built-in *HTML Schematron Validator* engine. There are several ways to validate an HTML document against a Schematron:

- **Configure a Validation Scenario** - You can create or edit a validation scenario *(on page 287)*, change the **File type** column to *HTML Document*, change the **Validation engine** column to *HTML Schematron Validator*, and specify the Schematron document in the **Schema** column.
- **Manually Validate a Single Document** - You can use the use the **Validate with** action from the **Validation** drop-down menu on the toolbar. This opens a dialog box where you can specify the Schematron document to validate the current document against.
- **Batch Validate Multiple Documents** - You can select multiple HTML documents in the **Navigator** view, right-click, and use the use the **Validate with schema** action from the **Validate** submenu. This opens a dialog box where you can specify the Schematron document to validate the selected documents against.

Notes:

- The Schematron must use the HTML5 namespace to reference the elements from the instance.
- Implicit HTML elements (i.e. `<html>`, `<body>`, `<tbody>`) must be included in an XPath expression in the Schematron document, even if they are missing from the HTML document.

Tip: The Oxygen XML Developer Eclipse plugin installation directory includes a `samples` folder that contains numerous sample files to help you learn about features, certain file types, and XML technologies. For example, inside the `samples` folder, there is an `html` folder with a `schematron` subfolder where you can find some samples that illustrate HTML validation against a Schematron schema.

Validating HTML Against Other Types of Schema

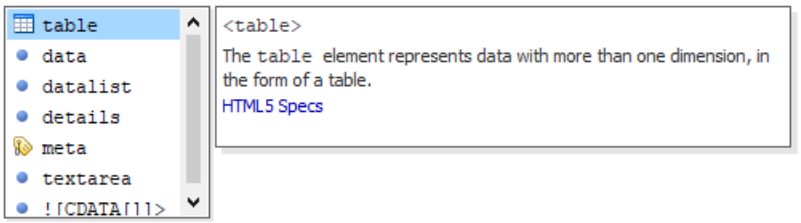
If your HTML document is XML well-formed, you could also configure a validation scenario to validate it as an XML document against other types of schemas. You would create or edit a validation scenario *(on page 287)*, make sure the **File type** column is set to *XML Document*, select the appropriate **Validation engine**, and specify the schema document in the **Schema** column.

HTML Content Completion Assistant

Oxygen XML Developer Eclipse plugin includes an intelligent *Content Completion Assistant (on page 1324)* that offers proposals for inserting HTML structures that are valid at the current editing location. Content completion is even available for CSS and JavaScript code that is embedded in an HTML document.

The *Content Completion Assistant* is enabled by default. To disable it, open the **Preferences** dialog box *(on page 36)*, go to **Editor > Content Completion**, and deselect the **Enable content completion** option *(on page 78)*.

Figure 266. Content Completion Assistant in HTML



Using the Content Completion in HTML

For HTML documents, the *Content Completion Assistant* uses a built-in schema and the list of proposals depend on the RELAX NG schema specified in the HTML framework. Using the content completion feature is the same as with any other XML document. For more details, see:

- Using the Content Completion Assistant in Text Mode *(on page 234)*

Code Templates in the Content Completion

Oxygen XML Developer Eclipse plugin includes a set of built-in code templates for HTML documents that can be selected from the *Content Completion Assistant*. The code templates are displayed with a symbol in the content completion list. You can also define your own code templates and share them with others. For more information, see Code Templates *(on page 238)*.

Content Completion for XPath Expressions

When entering XPath expressions in the **XPath Builder** view, the *Content Completion Assistant* is available as you type to help you compose query patterns.

Syntax Highlighting in HTML Documents

Oxygen XML Developer Eclipse plugin supports syntax highlighting in **Text** mode to make it easier to read the semantics of the structured content by displaying each type of code in different colors and fonts.

For HTML documents, it handles attributes without quotes, unclosed or void elements, and it also offers highlighting for embedded CSS or JavaScript content.

To customize the colors or styles used for the syntax highlighting colors for HTML files, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
3. Select and expand the **XHTML** 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 101)*

Folding in HTML

In a large HTML document, elements can be collapsed so that only the needed data remains in focus. The folding features available for XML documents *(on page 231)* are also available in HTML documents, but it also provides folding for nested elements that are not closed.

Minifyng HTML Documents

Minification (or compression) of an HTML document is the practice of removing unnecessary white spaces, without affecting the functionality of the document, but significantly reducing the loading time in Web browsers. While a minified HTML document gains in terms of execution performance, it is more difficult to read.

To minify an HTML document, right-click anywhere in the editor for an HTML document that is open in **Text** mode (or right-click an HTML document in the **Navigator** view and select the **Minify HTML** action. This opens a dialog box with the following options:

Output file

Use this option to set the name and location of the resulting compressed/minified HTML document.

Remove comments

If selected (default), all the HTML comments and also the comments from embedded CSS or JavaScript code blocks will be removed from the resulting output file.

Compress on a single line

If selected (default), the resulting output file will consist of a single line, as all the *'new line'* characters from the source document are removed.

Open output file in editor

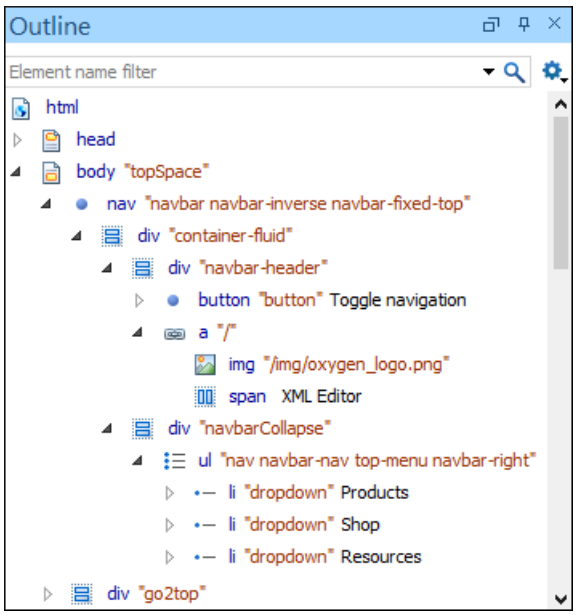
If selected (default), the resulting output file will be opened in Oxygen XML Developer Eclipse plugin.

When you click **OK**, the resulting HTML document is a compressed version of the original file for the purpose of enhanced performance, while losing some readability. The source HTML document is not affected.

HTML Outline View

The **Outline** view for HTML documents displays the structure of the HTML document you are editing. By default, it is displayed on the left side of the editor. In addition to the normal features available in the **Outline** view for XML documents *(on page 240)*, the HTML **Outline** view also handles void elements, elements that are not closed, or attributes without quotes, and presents the tree structure of the HTML document correctly.

Figure 267. HTML Outline View



Querying HTML Documents with XPath

Oxygen XML Developer Eclipse plugin provides a dedicated **XPath Builder** view *(on page 1119)* that allows you to compose complex XPath expressions and execute them over HTML documents (even if they are not well-formed according to XML standards). The **XPath Builder** view offers content completion as you type to help you compose expressions.

Editing Markdown Documents

Markdown was created as a lightweight markup language with plain text formatting syntax designed to provide 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 Eclipse 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 among most Markdown applications, the editor in Oxygen XML Developer Eclipse 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 (on page 1324)* and converting Markdown documents to DITA topics.
- Specialized syntax rules to combine popular syntax features from several specifications.

Markdown Editor

Oxygen XML Developer Eclipse plugin provides an intuitive, dynamic, and easy-to-use Markdown editor. It is a split-screen 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.

Markdown Text Editor Pane (Left Side)

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 Eclipse 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 Eclipse plugin uses an integration of rules *(on page 634)* 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 Eclipse 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 *(on page 632)*.
- **Automatic Spell Checking** - The Markdown editor supports the Oxygen XML Developer Eclipse plugin automatic spell checking feature *(on page 213)* that reports possible misspelled words as you type. You simply need to select the **Automatic spell check** option in the **Spell Check** preferences page *(on page 98)*, then click the **Select editors** button and select **Markdown Editor**.
- **Helpful Toolbar and Contextual Menu Actions** - A variety of unique actions *(on page 629)* 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.

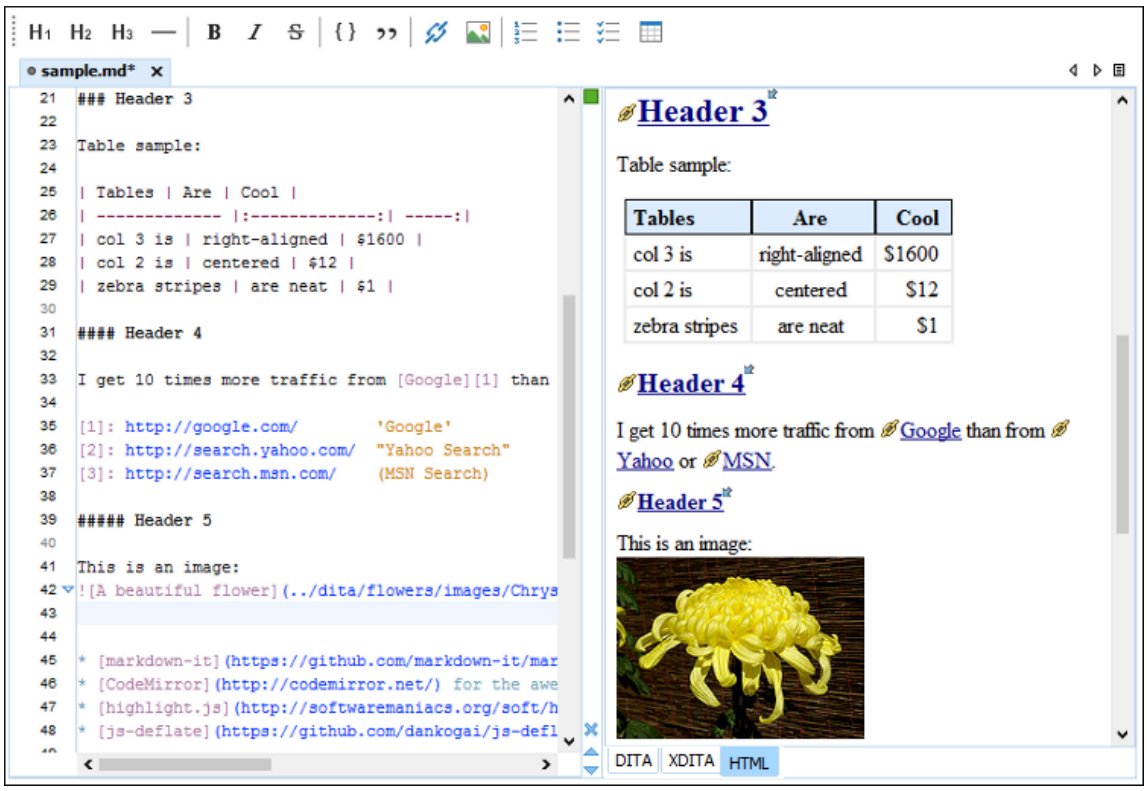
WYSIWYG Preview Pane (Right Side)

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**, **XDITA** (Lightweight DITA XML), 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 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.
- **Synchronous caret and scroll synchronization** - Moving the cursor in the editor area will display the corresponding move in the *Preview* area.
- **Export Options** - The **DITA** tab includes a contextual menu action to export (convert) the current Markdown document to a DITA topic *(on page 632)*. The **XDITA** tab includes a contextual menu action to export (convert) the current Markdown document to a Lightweight DITA topic *(on page 632)*. Similarly, the **HTML** tab includes a contextual menu action to export (convert) it to an XHTML document *(on page 632)*.
- **Automatic Validation** - As you edit Markdown documents, they are validated automatically *(on page 632)*. 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 *(on page 248)* 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 Eclipse plugin.

Figure 268. Markdown Editor



Related Information:

- Markdown Editor Syntax Rules and Specifications *(on page 634)*
- Actions Available in the Markdown Editor *(on page 629)*
- Working with Markdown Documents in DITA
- Creating New Markdown Documents *(on page 629)*

Creating New Markdown Documents

To create a new Markdown document in Oxygen XML Developer Eclipse plugin, follow these steps:

1. Click the **New** button on the toolbar or select **File > New**.
2. Select the **Markdown** document template.
3. Click **Next**.
4. Choose the storage path and 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 *(on page 627)*.

Related Information:

- Markdown Editor *(on page 627)*

Actions Available in the Markdown Editor

Aside from the actions that are available in Oxygen XML Developer Eclipse 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 available on the Markdown toolbar when editing Markdown documents:

Header (1st Level)

Inserts an *atx-style* first-level header *(on page 634)* at the cursor position.

Header (2st Level)

Inserts an *atx-style* second-level header *(on page 634)* at the cursor position.

Header (3rd Level)

Inserts an *atx-style* third-level header *(on page 634)* at the cursor position.

Horizontal Rule

Inserts a horizontal rule *(on page 635)* at the cursor position.

Bold (Strong)

Marks the selected text with bold *(on page 636)*.

Italic (Emphasis)

Marks the selected text with italics *(on page 635)*.

Strikethrough

Marks the selected text with a strikethrough *(on page 636)*.

Code Block

Inserts (or surrounds selected text in) a codeblock *(on page 639)*.

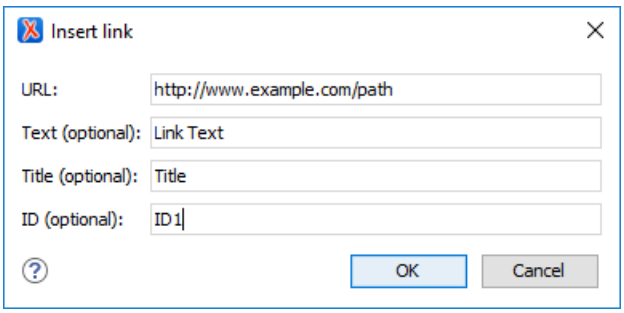
Blockquote

Inserts a blockquote *(on page 638)* at the cursor position.

Insert Link

Opens the **Insert Link** dialog box that allows you to define a link *(on page 636)* to insert at the cursor position.

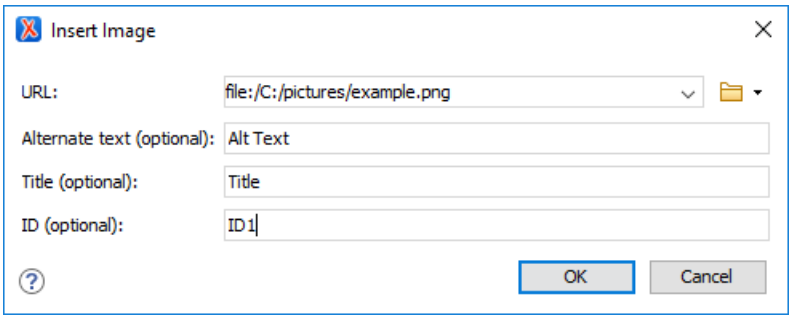
Figure 269. Insert Link Dialog Box



Insert Image

Opens the **Insert Image** dialog box that allows you to define an image *(on page 638)* to insert at the cursor position. You can type the URL of the image you want to insert or use browsing actions in the **Browse** drop-down menu.

Figure 270. Insert Link Dialog Box



Insert Ordered List

Inserts an ordered list *(on page 641)* at the cursor position. Three child list items are also automatically inserted by default. You can also use this action to convert selected content to an ordered list.

Insert Unordered List

Inserts an unordered list *(on page 641)* at the cursor position. Three child list items are also automatically inserted by default. You can also use this action to convert selected content to an unordered list.

Insert Task List

Inserts a task list *(on page 642)* at the cursor position. Three child list items are also automatically inserted by default. You can also use this action to convert selected content to a task list.

Insert Table

Inserts a table *(on page 642)* 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 XDITA Topic

Converts the current Markdown document into a Lightweight DITA XML topic.

Export as HTML

Converts the current Markdown document into an XHTML 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 627)*

Working with Markdown Documents in DITA

Markdown Editor Syntax Rules and Specifications *(on page 634)*

Syntax Highlighting in the Markdown Editor

Oxygen XML Developer Eclipse plugin supports syntax highlighting in the Markdown editor to make it easier to read the semantics of the structured content by displaying each type of XML code in different colors and fonts.

To customize the colors or styles used for the syntax highlighting colors for Markdown documents, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*.
2. Go to **Editor > Syntax Highlight** *(on page 101)*.
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 101)*

Markdown Editor *(on page 627)*

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 *(on page 248)* at the bottom of the editor.

Examples of the type of errors that will be reported include headers being in the wrong order or the syntax of a document begins with something other than a 1st level header.

Validating Markdown Documents with Schematron

It is possible to validate Markdown documents with Schematron rules. There are two ways to create an association between Markdown documents and Schematron files:

- You can configure an association using the **Markdown** preferences page *(on page 109)*. You can specify a Schematron file to validate converted HTML content, as well as one to validate converted DITA content.
- You can create a Schematron association for Markdown documents by adding a catalog mapping *(on page 312)* for one of the following URIs:
  - **http://www.oxygenxml.com/schematron/validation/markdown-as-html** - The obtained Schematron will be applied over HTML conversions.
  - **http://www.oxygenxml.com/schematron/validation/markdown-as-dita** - The obtained Schematron will be applied over DITA conversions.

  The catalog mapping is a fallback in case the Schematron validation is disabled in the **Markdown** preferences page *(on page 109)* or the path to the Schematron file is empty.

  ⚠ **Warning:** If you are using a custom version of DITA-OT *(on page 47)*, the mapping information might not be generated properly, causing issues with the Schematron validation. For example, error locations may not be accurate or synchronization may fail.

Tip: Inside the samples folder, there is a `schematron-validation` folder with some files you can use to see how Schematron validation can be done with Markdown files. The path of the folder is: `[OXYGEN_INSTALL_DIR]/samples/markdown/schematron-validation/`.

Related Information:

Markdown Editor *(on page 627)*

Working with Markdown Documents in DITA

Oxygen XML Developer Eclipse 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 DITA topic will look after conversion. Likewise, the **XDITA** tab in the *Preview* pane provides a visualization of how a

Lightweight DITA topic will look after conversion. Keys that are defined in the root map are also resolved in the *Preview* pane.

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.

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 Eclipse plugin comes with a sample ditamap project for converting Markdown to DITA. Go to the **Navigator** view *(on page 192)*, open the `sample.xpr` project, and navigate to the `dita/markdown-dita` folder.

Related Information:

- Markdown Editor *(on page 627)*
- Actions Available in the Markdown Editor *(on page 629)*
- Markdown Editor Syntax Rules and Specifications *(on page 634)*
- Automatic Validation in Markdown Documents *(on page 632)*
- Markdown DITA Syntax Reference

Markdown Editor Syntax Rules and Specifications

The Markdown editor in Oxygen XML Developer Eclipse 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.

This topic lists the Oxygen XML Developer Eclipse plugin implementation of the most commonly used syntax rules.

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 (they also need to be preceded and followed by a blank line). 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 (`<br>`. 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 reference 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") inline link and title.
  ```

  Without a title:

  ```
  Text with [example link text](http://www.example.com/path) inline link without a title.
  ```

  Relative path:

  ```
  Text with [example link text](/relative_path/) inline link with relative path.
  ```

- **Reference Links**

  Reference-type links use a second set of square brackets that include a label (link identifier) to reference the target for the link (link identifier 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;&#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 two types of links (*inline* and *reference*). In both cases, the syntax for images begins 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") inline image and a title.
  ```

  Without a title:

  ```
  Text with ![Alt text](/path/to/img.jpg) 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 clearer) if you put a > before every line.

- **Example: Blockquotes**

  ```
  > This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet,
  > consectetuer 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 consectetuer libero luctus adipiscing.
```

- **Example: Nested Blockquotes**

  Blockquotes can be nested by adding additional levels of > characters.

```
> This is the first level of quoting.
>
> > This is nested blockquote.
>
> Back to the first level.
```

- **Example: Blockquotes with Other Markdown Elements**

  Blockquotes can also contain other Markdown elements (such as headers, lists, and code blocks).

```
> ## 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:
```

```
This is a code block
```

```
```

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 Eclipse 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 (on page 638)* and *code blocks (on page 639)* 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 three 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, consectetuer 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 consectetuer 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 a hyphen followed by a space followed by 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  |
  ```

- **Joining Cells (Span a Cell Over Multiple Columns)**

  You can join cells horizontally (span a cell over multiple columns) by using multiple consecutive pipe characters (`|`) to the right of the particular cell. The number of consecutive pipes indicate the number of columns the cell will span (`||` for two, `|||` for three, and so on).

  ```
  | First Header | Second Header | Third Header | Fourth Header |
  | ------------ | ------------- | ------------ | ------------- |
  | Content Cell | *Cell Span Over 3 Columns* |||
  ```

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, `<b>`), they are treated as literal characters and escaping is not necessary.

Literal Character	Escaping Code
<	<
>	>
&	&

Footnotes

The Markdown editor in Oxygen XML Developer Eclipse 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 627)*

Actions Available in the Markdown Editor *(on page 629)*

9.

Built-in XML Frameworks (Document Types)

Oxygen XML Developer Eclipse plugin includes a variety of specialized editors, views, and features that are dynamic according to the type of document that you open or create. Oxygen XML Developer Eclipse plugin includes fully supported built-in *frameworks (on page 1325)* for the most popular XML document types (DITA, DocBook, TEI, XHTML, JATS) with a full set of features, as well as support for other document types with more generic features *(on page 222)*.

The built-in *frameworks (on page 1325)* are defined according to a set of rules and a variety of settings that improve editing capabilities for its particular file type. These settings include:

- A default grammar used for validation and content completion in **Text** mode.
- Built-in transformation scenarios used for publishing XML documents.
- *XML Catalogs (on page 1329)* used for mapping resources.
- New document templates to make it easy to create XML documents.

DocBook 4 Document Type (Framework)

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 *(on page 177)* and they can be found in: **Framework Templates > DocBook 4**.

The default templates for DocBook 4 documents are located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/docbook/templates/Docbook 4` folder.

Default Schema for Validation and Content Completion

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 (on page 1329)*, `catalog.xml`, is stored in *`[OXYGEN_INSTALL_DIR]/frameworks/docbook/`*.

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes numerous built-in DocBook transformation scenarios that allow you to transform DocBook 4 documents to a variety of outputs, such as WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, and EPUB. All of them are listed in the **DocBook 4** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

For more information, see the DocBook Transformation Scenarios *(on page 686)* section.

Resources

-
- DocBook Specifications

Related Information:

Editing XML Documents in Text Mode *(on page 222)*

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 means for pulling in the target data for each document. The database document is static and all the document data is pulled in dynamically.

   Tip: Oxygen XML Developer Eclipse plugin includes a built-in new document template called **DocBook Targetset Map** available in the **New from templates** wizard *(on page 177)* that will help you get started.

   Example: 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 agtargets SYSTEM "file:///doc/adminguide/target.db">
<!ENTITY reftargets SYSTEM "file:///doc/man/target.db">
]>
<targetset>
  <targetsetinfo>
    Description of this target database document,
    which is for the examples in olink doc.
  </targetsetinfo>
  <!-- Site map for generating relative paths between documents -->
  <sitemap>
    <dir name="documentation">
      <dir name="guides">
        <dir name="mailuser">
          <document targetdoc="MailUserGuide"
                    baseuri="userguide.html">
            &ugtargets;
          </document>
        </dir>
        <dir name="mailadmin">
          <document targetdoc="MailAdminGuide">
            &agtargets;
```

```
          </document>
        </dir>
      </dir>
      <dir name="reference">
        <dir name="mailref">
          <document targetdoc="MailReference">
            &reftargets;
          </document>
        </dir>
      </dir>
    </dir>
  </sitemap>
</targetset>
```

4. Generate the target data files by executing a DocBook transformation scenario.
   Before applying the transformation, you need to edit the transformation scenario, go to 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.
   Example: An example of a `target.db` file:

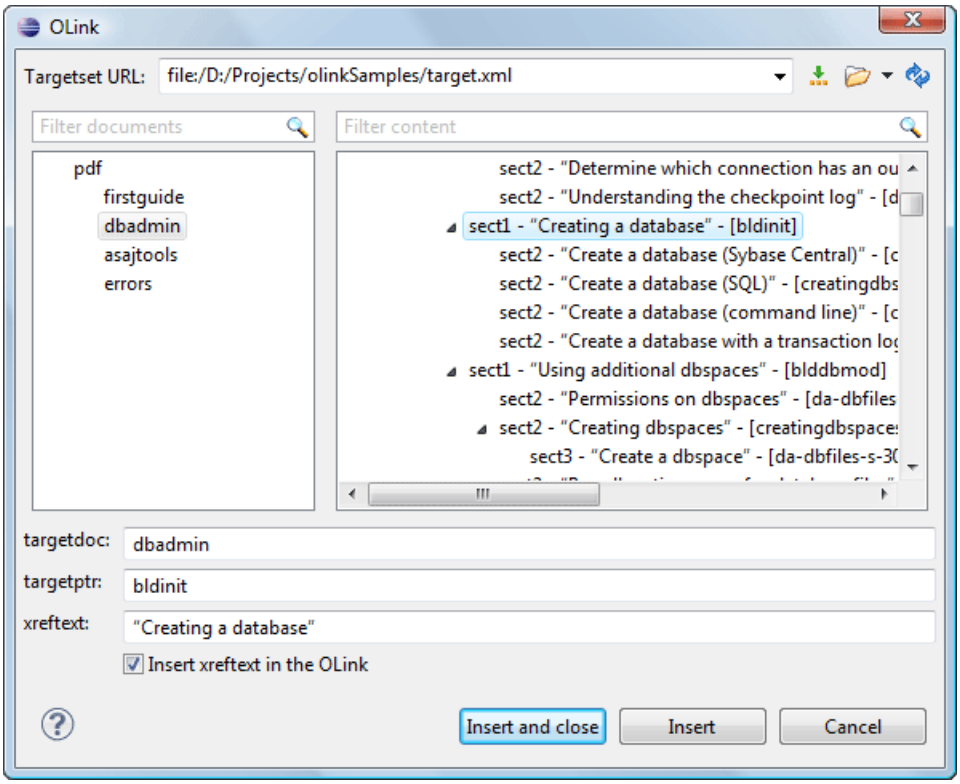
```
<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">
 <ttl>Administering User Accounts</ttl>
 <xreftext>How to administer user accounts</xreftext>
 <div element="part" href="#d5e4" number="I">
  <ttl>First Part</ttl>
  <xreftext>Part I, “First Part”</xreftext>
  <div element="chapter" href="#d5e6" number="1">
    <ttl>Chapter Title</ttl>
    <xreftext>Chapter 1, Chapter Title</xreftext>
    <div element="sect1" href="#src_chapter" number="1" targetptr="src_chapter">
      <ttl>Section1 Title</ttl>
      <xreftext>xreflabel_here</xreftext>
    </div>
  </div>
 </div>
</div>
```

5. Insert `<olink>` elements in the DocBook documents.
   When editing a DocBook XML document in **Author** mode, the **Insert OLink** action is available in the **Link** drop-down menu from the toolbar. This action opens the **Insert OLink** dialog box that allows you to select the target of an `<olink>` from the list of all possible targets from a specified target database document (specified in the **Targetset URL** field). Once a **Targetset URL** is selected, the structure of the target documents is presented. For each target document (`@targetdoc`), its content is displayed, allowing you to easily identify the appropriate `@targetptr`. You can also use the search fields

to quickly identify a target. If you already know the values for the `@targetdoc` and `@targetptr` attributes, you can insert them directly in the corresponding fields.

Example: 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 271. 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 (Framework)

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 *(on page 177)* and they can be found in: **Framework Templates > DocBook 5 > DocBook 5.0** and **Framework Templates > DocBook 5 > DocBook 5.1**.

New document templates for both DocBook 5 documents are located in the *[OXYGEN_INSTALL_DIR]/*`frameworks/docbook/templates/Docbook5.0` folder.

New document templates for both DocBook 5.1 documents are located in the *[OXYGEN_INSTALL_DIR]/*`frameworks/docbook/templates/Docbook5.1` folder.

Default Schema for Validation and Content Completion

The default schema that is used if one is not detected is `docbookxi.rng` and it is stored in *[OXYGEN_INSTALL_DIR]*`/frameworks/docbook/5.0/rng/` (or for DocBook 5.1 in *[OXYGEN_INSTALL_DIR]*`/frameworks/docbook/5.1/rng/`). Other types of schemas for various DocBook versions are also located in various folders inside the *[OXYGEN_INSTALL_DIR]*`/frameworks/docbook/` directory.

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes numerous built-in DocBook transformation scenarios that allow you to transform DocBook 5 documents to a variety of outputs, such as WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, and EPUB. Oxygen XML Developer Eclipse plugin also includes a DocBook 5.1 transformation scenario for *Assembly* documents *(on page 655)*. All of them are listed in the **DocBook 5** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

For more information, see the DocBook Transformation Scenarios *(on page 686)* section.

Resources

- DocBook 5.0 (and older) Specifications
- DocBook 5.1 Specifications
- DocBook 5.1: The Definitive Guide

Related Information:

Editing XML Documents in Text Mode *(on page 222)*

DocBook 5.1 Assembly *(on page 655)*

DocBook 5.1 Topic *(on page 656)*

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 means for pulling in the target data for each document. The database document is static and all the document data is pulled in dynamically.

   Tip: Oxygen XML Developer Eclipse plugin includes a built-in new document template called **DocBook Targetset Map** available in the **New from templates** wizard *(on page 177)* that will help you get started.

   Example: 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 agtargets SYSTEM "file:///doc/adminguide/target.db">
   <!ENTITY reftargets SYSTEM "file:///doc/man/target.db">
   ```

```
]>
<targetset>
  <targetsetinfo>
    Description of this target database document,
    which is for the examples in olink doc.
  </targetsetinfo>
  <!-- Site map for generating relative paths between documents -->
  <sitemap>
    <dir name="documentation">
      <dir name="guides">
        <dir name="mailuser">
          <document targetdoc="MailUserGuide"
                    baseuri="userguide.html">
            &ugtargets;
          </document>
        </dir>
        <dir name="mailadmin">
          <document targetdoc="MailAdminGuide">
            &agtargets;
          </document>
        </dir>
      </dir>
      <dir name="reference">
        <dir name="mailref">
          <document targetdoc="MailReference">
            &reftargets;
          </document>
        </dir>
      </dir>
    </dir>
  </sitemap>
</targetset>
```

4. Generate the target data files by executing a DocBook transformation scenario.

   Before applying the transformation, you need to edit the transformation scenario, go to 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.

   Example: An example of a `target.db` file:

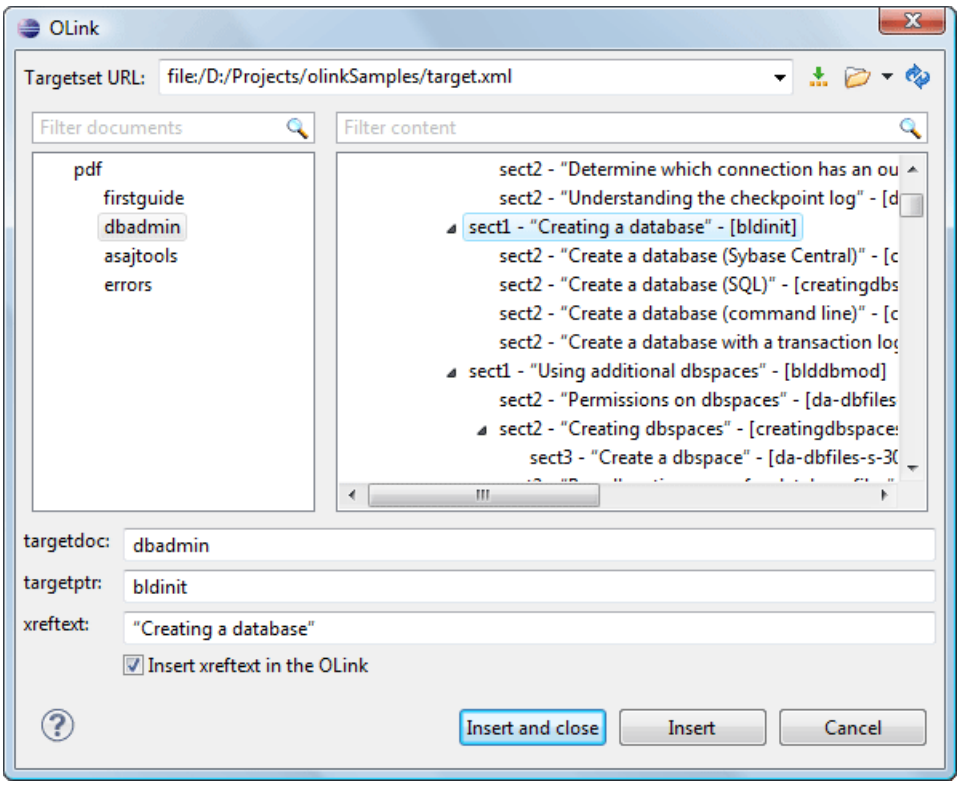
```
<div element="book" href="#MailAdminGuide" number="1" targetptr="user_accounts">
 <ttl>Administering User Accounts</ttl>
 <xreftext>How to administer user accounts</xreftext>
 <div element="part" href="#d5e4" number="I">
```

```
  <ttl>First Part</ttl>
  <xreftext>Part I, “First Part”</xreftext>
  <div element="chapter" href="#d5e6" number="1">
    <ttl>Chapter Title</ttl>
    <xreftext>Chapter 1, Chapter Title</xreftext>
    <div element="sect1" href="#src_chapter" number="1" targetptr="src_chapter">
      <ttl>Section1 Title</ttl>
      <xreftext>xreflabel_here</xreftext>
    </div>
  </div>
 </div>
</div>
```

5. Insert `<olink>` elements in the DocBook documents.
   When editing a DocBook XML document in **Author** mode, the **Insert OLink** action is available in the **Link** drop-down menu from the toolbar. This action opens the **Insert OLink** dialog box that allows you to select the target of an `<olink>` from the list of all possible targets from a specified target database document (specified in the **Targetset URL** field). Once a **Targetset URL** is selected, the structure of the target documents is presented. For each target document (`@targetdoc`), its content is displayed, allowing you to easily identify the appropriate `@targetptr`. You can also use the search fields to quickly identify a target. If you already know the values for the `@targetdoc` and `@targetptr` attributes, you can insert them directly in the corresponding fields.

   Example: 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 272. 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 5.1 Assembly

The DocBook *Assembly* document type was introduced with DocBook 5.1 and it is used to define the hierarchy and relationships for a collection of resources. It is especially helpful for topic-oriented authoring scenarios since it assembles a set of resources (such as DocBook 5.1 *topics (on page 656)*) to form a hierarchical structure for a larger publication.

An *Assembly* document usually has four major parts:

- **Resources** - Identifies a collection of resources (such as topics). An *Assembly* may identify one or more collections.
- **Structure** - Identifies an artifact to be assembled. A document in this case is the particular collection of resources (such as topics) that forms the documentation. Within the `<structure>` element, an `<output>` element can be used to identify the type of output to be generated and `<module>` elements can be used to identify the resources to be included. An *Assembly* may identify one or more *structures*.
- **Relationships** - Identifies relationships between resources. These relationships may be manifested in any number of *structures* during assembly. An *Assembly* may identify any number of relationships.

- **Transformations** - Identifies transformations that can be applied during assembly. An *Assembly* may identify any number of transformations.

For detailed information about the DocBook *Assembly* document type, see DocBook 5.1: The Definitive Guide - DocBook Assemblies.

File Definition

A file is considered to be an *Assembly* when the root name is `assembly`.

Default Document Templates

A default **Assembly** document template is available when creating new documents from templates *(on page 177)* and it can be found in: **Framework Templates > DocBook 5 > DocBook 5.1**.

The default template for DocBook Assembly documents is located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/docbook/templates/Docbook5.1` folder.

Default Schema for Validation and Content Completion

The default schema that is used if one is not detected is `docbookxi.rng` and it is stored in *[OXYGEN_INSTALL_DIR]*`/frameworks/docbook/5.1/rng/`.

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes a built-in transformation scenario that can be applied on an *Assembly* file to generate an *assembled* (merged) DocBook file. The scenario is called **DocBook Assembly** and is found in the **DocBook 5** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Resources

- DocBook 5.1: The Definitive Guide - DocBook Assemblies
- DocBook 5.1 Specifications
- Sample files: *[OXYGEN_INSTALL_DIR]*`/samples/docbook/v5/assembly/`

DocBook 5.1 Topic

The DocBook *Topic* document type was introduced with DocBook 5.1 and it is used as a modular unit of documentation. It is similar to the concept of the DITA *Topic* and can be used as modular resources in conjunction with DocBook 5.1 *Assembly (on page 655)* documents.

For detailed information about the DocBook *Topic* document type, see DocBook 5.1: The Definitive Guide - Topic.

File Definition

A DocBook file is considered to be a *Topic* when the root name is `topic`.

Default Document Templates

A default **Topic** document template is available when creating new documents from templates *(on page 177)* and it can be found in: **Framework Templates > DocBook 5 > DocBook 5.1**.

The default template for DocBook Assembly documents is located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/docbook/templates/Docbook5.1` folder.

Default Schema for Validation and Content Completion

The default schema that is used if one is not detected is `docbookxi.rng` and it is stored in *[OXYGEN_INSTALL_DIR]*`/frameworks/docbook/5.1/rng/`.

Transformation Scenarios

Since DocBook *Topics* are modular resources, they are *assembled* and transformed in the **DocBook Assembly** transformation process *(on page 656)*. You can also use any of the built-in DocBook transformation scenarios to transform individual DocBook Topics to a variety of outputs, such as PDF, HTML, EPUB, and more. They are found in the **DocBook 5** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Resources

- DocBook 5.1: The Definitive Guide - Topic
- DocBook 5.1 Specifications
- Sample files: *[OXYGEN_INSTALL_DIR]*`/samples/docbook/v5/assembly/`

Related Information:

DocBook 5.1 Assembly *(on page 655)*

DocBook Targetset Document Type (Framework)

DocBook *Targetset* documents are used to resolve cross references with the DocBook *Olink*.

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 *(on page 177)* and it can be found in: **Framework Templates > DocBook Targetset**.

The default template for DocBook Targetset documents is located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/docbook/templates/Targetset` folder.

Default Schema for Validation and Content Completion

The default schema, `targetdatabase.dtd`, for this type of document is stored in *`[OXYGEN_INSTALL_DIR]`*`/frameworks/docbook/xsl/common/`.

Related Information:

DocBook Specifications

DITA Topics Document Type (Framework)

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 a `@DITAArchVersion` attribute 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 *(on page 51)* is selected from the **Document Type Association** preferences page *(on page 50)*.

Default Document Templates

There are a variety of default *DITA topic* templates available when creating new documents from templates *(on page 177)* and they can be found in various folders inside: **Framework Templates > DITA**.

The default templates for DITA topic documents are located in the *`[OXYGEN_INSTALL_DIR]`*`/frameworks/dita/templates/topic` folder.

Default Schema for Validation and Content Completion

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 *XML Catalogs (on page 1329)* 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`

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes built-in transformation scenarios for transforming individual DITA Topics to HTML5, XHTML, or PDF output. They can be found in the **DITA** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

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 222)*

DITA Map Document Type (Framework)

DITA maps (on page 1324) 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 a `@class` attribute 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 *(on page 51)* from the **Document Type Association** preferences page *(on page 50)* is selected.

Default Document Templates

There are a variety of default *DITA map* templates available when creating new documents from templates *(on page 177)* and they can be found in various folders inside: **Framework Templates > DITA Map**.

The default templates for *DITA map* documents are located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/sita/templates/map` folder.

Default Schema for Validation and Content Completion

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 *XML Catalogs (on page 1329)* for the *DITA map* document type are as follows:

- *[OXYGEN_INSTALL_DIR]*`/frameworks/dita/catalog.xml`
- *DITA-OT-DIR*`/catalog-dita.xml`

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes numerous built-in transformation scenarios that allow you to transform *DITA maps* to a variety of outputs, such as WebHelp, PDF, ODF, XHTML, EPUB, and CHM. All of them are listed in the **DITA Map** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

For more information, see the DITA Map Transformation Scenarios section.

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 222)*

XHTML Document Type (Framework)

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 is `<html>`.

Default Document Templates

There are a variety of default *XHTML* templates available when creating new documents from templates *(on page 177)* and they can be found in: **Framework Templates > XHTML**.

The default templates for XHTML documents are located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/xhtml/templates/` folder.

Default Schema for Validation and Content Completion

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 *XML Catalogs (on page 1329)* 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-compat.xml`

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes built-in transformation scenarios that allow you to transform XHTML documents to several types of DITA document types (topic, task, concept, reference). They can be found in the **XHTML** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Related Information:

Editing HTML Documents *(on page 621)*

Editing XML Documents in Text Mode *(on page 222)*

XHTML Specifications

TEI P5 Document Type (Framework)

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 *(on page 177)* and they can be found in: **Framework Templates > TEI P5**.

The default templates for TEI P5 documents are located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/tei/templates/TEI P5` folder.

Default Schema for Validation and Content Completion

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 *XML Catalogs (on page 1329)* 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`

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes built-in transformation scenarios that allow you to transform TEI P5 documents to a variety of outputs, such as PDF, XHTML, EPUB, DOCX, and ODT. They can be found in the **TEI P5** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Resources

-
- TEI: P5 Guidelines

Related Information:

Editing XML Documents in Text Mode *(on page 222)*

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 (on page 1325)* for Oxygen XML Developer Eclipse 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 Eclipse plugin `frameworks` folder.

TEI ODD Document Type (Framework)

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 *(on page 177)* and they can be found in: **Framework Templates > TEI ODD**.

The default template is located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/tei/templates/TEI ODD` folder.

Default Schema for Validation and Content Completion

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 *XML Catalogs (on page 1329)* 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`

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes built-in transformation scenarios that allow you to transform TEI ODD documents to a variety of outputs, such as PDF, XHTML, EPUB, DOCX, ODT, RNG, DTD, and XML Schema. They can be found in the **TEI ODD** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Resources

-
- TEI: Getting Started with ODD

Related Information:

Editing XML Documents in Text Mode *(on page 222)*

jTEI Document Type (Framework)

The *jTEI (Journal of the Text Encoding Initiative)* document type is a highly restrictive customization (only about 80 elements are included) of the TEI P5 *framework*.

File Definition

A file is considered to be a *jTEI* document when the root element is named *TEI*, it is in the namespace *http://www.tei-c.org/ns/1.0*, and the `@rend` attribute is set to *"jTEI"*.

Default Document Templates

There is a default **jTEI Article** template available when creating new documents from templates *(on page 177)* and they can be found in: **Framework Templates > TEI JTEI**.

The default template is located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/tei/templates/TEI jTEI` folder.

Default Schema for Validation and Content Completion

The default schema that is used if one is not detected is `tei_jtei.rng` and it is stored in *[OXYGEN_INSTALL_DIR]*`/frameworks/tei/xml/tei/custom/schema/relaxng/`.

Default XML Catalogs

The default *XML Catalogs (on page 1329)* for jTEI 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`

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes built-in transformation scenarios that allow you to transform jTEI documents to PDF and ODT. They can be found in the **TEI JTEI** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Resources

-
- jTEI Article Guidelines

Related Information:

Editing XML Documents in Text Mode *(on page 222)*

JATS Document Type (Framework)

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 *(on page 177)* and they can be found in: **Framework Templates > JATSKit - NISO JATS and NLM BITS**

The default templates for JATS documents are located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/jats/templates/` folder.

Default Schema for Validation and Content Completion

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 (on page 1329)*, `jatskit-catalog.xml`, is stored in *[OXYGEN_INSTALL_DIR]*`/frameworks/jats/lib/schemas/`.

Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes built-in transformation scenarios that allow you to transform JATS documents to a variety of outputs, such as PDF, HTML, and EPUB. They can be found in the **JATSKit** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Resources

-
- NLM Journal Archiving and Interchange Tag Suite

Related Information:

Editing XML Documents in Text Mode *(on page 222)*

EPUB Document Type (Framework)

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 *(on page 1126)* in the **Navigator** view or main editing pane.

Three distinct *frameworks (on page 1325)* 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 Eclipse plugin supports OPF 2.0, OPF 3.0, and OPF 3.1.

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 *(on page 177)* and they can be found the following folders in **Framework Templates**: **NCX**, **OCF**, **OPF 2.0**, **OPF 3.0**, and **OPF 3.1**.

- 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.
- The default template for the **OPF 3.1** document type is located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/opf/templates/3.1` folder.

Default Schema

The default schema files for the various types of EPUB document types are located in the following directories:

- The default schema files for the **NCX** document types are located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/ncx/schemas` folder.
- The default schema files for the **OCF** document types are located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/ocf/schemas` folder.
- The default schema files for the **OPF 2.0** document type is located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/opf/schemas/2.0` folder.
- The default schema files for the **OPF 3.0** document type is located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/opf/schemas/3.0` folder.
- The default schema files for the **OPF 3.1** document type is located in the *[OXYGEN_INSTALL_DIR]*`/frameworks/opf/schemas/3.1` folder.

Related Information:

Working with Archive Files *(on page 1128)*

Other Supported Document Types

Along with the fully supported built-in *frameworks* (document types) *(on page 646)*, Oxygen XML Developer Eclipse plugin also provides limited support (including document 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 Eclipse plugin include:

- EPUB (NCX, OCF, OPF 2.0, 3.0, & 3.1) *(on page 665)* - A standard for e-book files.
- OOXML *(on page 1128)* - An XML-based file format for representing spreadsheets, charts, presentations, and word processing documents.
- ODF *(on page 1128)* - An free and open-source XML-based file format for electronic office documents, such as spreadsheets, charts, presentations, and word processing documents.
- DocBook Targetset *(on page 657)* - For resolving cross-references when using *olinks*.
- XSLT Stylesheets *(on page 363)* - A document type that provides a visual mode for editing XSLT stylesheets.
- WSDL *(on page 500)* - Web Services Description Language is an XML language for describing the functionality offered by a web service.
- Schematron *(on page 589)* - 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) *(on page 303)* - An extension of the ISO standard Schematron, allows developers to define *Quick Fixes (on page 1328)* for Schematron errors.
- StratML (Part 1 & 2) *(on page 582)* - Part 1 and 2 of the Strategy Markup Language specification.
- XProc *(on page 588)* - A document type for processing XProc script files.
- XML Schema *(on page 406)* - Documents that provide support for editing annotations.
- XLIFF (1.2, 2.0, 2.1) *(on page 582)* - XML Localization Interchange File Format is a standard for passing data between tools during a localization process.
- XQuery *(on page 485)* - The common query language for XML.
- CSS *(on page 521)* - Cascading Style Sheets is a language used for describing the look and formatting of a document.
- Relax NG Schema *(on page 525)* - A schema language that specifies a pattern for the structure and content of an XML document.
- NVDL Schema *(on page 542)* - Namespace Validation Dispatching Language allows you to specify sections of XML documents to be validated against various schemas.
- JSON *(on page 550)* - JavaScript Object Notation is a lightweight data-interchange format.
- Markdown *(on page 626)* - A lightweight markup language with plain text formatting syntax that can be converted to HTML or DITA.
- JavaScript *(on page 583)* - Programming language of HTML and the Web.

- XMLSpec - 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 XML - 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* - An *XML Catalog (on page 1329)* document that describes a mapping between external entity references and locally-cached equivalents.
- Other Non-XML Files *(on page 190)*- Oxygen XML Developer Eclipse plugin also includes a simple text editor and a variety of helpful features for creating and editing non-XML files.

10.
Publishing

XML documents can be transformed into a variety of user-friendly output formats that can be viewed by end-users. This process is known as a *transformation*.

Oxygen XML Developer Eclipse plugin includes numerous built-in transformation possibilities 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 Eclipse 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 a transformation to any format is that your source document is well-formed.

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.

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.

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 Eclipse 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. Oxygen XML Developer Eclipse plugin includes preconfigured built-in transformation scenarios *(on page 670)*, but you can also create new transformation scenarios *(on page 691)*.

When creating new transformation scenarios, the types that are available 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. For more information, see XML Transformation with XSLT *(on page 691)* and XML Transformation with XQuery *(on page 707)*.

- **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. For more information, see XSLT Transformation *(on page 732)*.
- **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. For more information, see XQuery Transformation *(on page 754)*.
- **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. For more information, see SQL Transformation *(on page 762)*.
- **DITA-OT Scenarios** - This type of scenario provides the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Developer Eclipse 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. For more information, see DITA-OT Transformation *(on page 716)*.
- **ANT Scenarios** - This type of scenario contains the location of an Ant build script, as well as other transform parameters. For more information, see Ant Transformation *(on page 729)*.

Note:

Status messages generated during the transformation process are displayed in the **Console** view *(on page 220)* (the **Enable Oxygen consoles** option *(on page 112)* must be selected in the **View** preferences page *(on page 112)*).

Built-in Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes preconfigured built-in transformation scenarios that are used for common transformations. They can be found in the various sections in the **Configure Transformation Scenario(s)** dialog box *(on page 765)* or **Transformation Scenarios** view *(on page 770)*. All the built-in *document types (frameworks) (on page 1325)* that are included in Oxygen XML Developer Eclipse plugin have various transformation scenarios in their specific sections, including the most popular *frameworks*, such as DITA, DocBook, TEI, XHTML, JATS, OOXML, and more.

To obtain the desired output, use one of the following actions from the toolbar or **Transform** submenu in the contextual menu of the **Navigator** view *(on page 192)*:

- **Apply Transformation Scenario(s) (Alt + Shift + T, T (Command + Alt + T, T on OS X))** - If you have associated transformation scenarios for the current document, this action will simply apply the association *(on page 764)* and begin the transformation process. If an association is not detected, this action will open the **Configure Transformation Scenario(s)** dialog box *(on page 765)* where you can choose the scenarios you want to apply.

- **Configure Transformation Scenario(s) (Alt + Shift + T, C (Command + Alt + T, C on OS X))** - This action will open the **Configure Transformation Scenario(s)** dialog box *(on page 765)* where you can choose the scenarios you want to apply.

Note:

- You can apply a transformation even if the current document is not associated with a transformation scenario.
- If the document contains an `xml-stylesheet` processing instruction that references an XSLT stylesheet (commonly used to display the document in web browsers), Oxygen XML Developer Eclipse 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.

Related Information:

Creating New Transformation Scenarios *(on page 691)*

Editing a Transformation Scenario *(on page 762)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

Applying Associated Transformation Scenarios *(on page 764)*

Transformation Scenarios View *(on page 770)*

DITA Map Transformation Scenarios

Built-in transformation scenarios allow you to transform *DITA maps (on page 1324)* to a variety of outputs, such as WebHelp, PDF, ODF, XHTML, EPUB, CHM, Kindle, and MS Word. Oxygen XML Developer Eclipse plugin also includes a special **Run DITA-OT Integrator** *(on page 684)* that can be used to integrate a DITA-OT plugin and a **DITA Map Metrics Report** transformation that generates a statistics report for your DITA map. All of them are listed in the **DITA Map** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

A variety of transformations scenarios are available for *DITA maps (on page 1324)*:

- Built-in transformation scenarios allow you to transform a *DITA map* to a variety of outputs, such as WebHelp, PDF, ODF, XHTML, EPUB, CHM, Kindle, and MS Word.
- **Run DITA-OT Integrator** *(on page 684)* - 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. It contains 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*.
- Processing instructions.
- External links.
- All `@outputclass` attribute values gathered from the DITA project.

Important: If you have cross references that point to content outside the scope of the DITA map, that referenced content will not be counted. For example, if you have links to topics that are not included in the DITA map hierarchy, the content in those topics is ignored when generating the statistics.

Related Information:

Editing a Transformation Scenario *(on page 762)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

Applying Associated Transformation Scenarios *(on page 764)*

DITA Topic Transformation Scenarios

DITA Map WebHelp Responsive Transformation

DITA content can be transformed into several types of WebHelp Responsive systems (with or without a feedback section). The WebHelp Responsive layout and features *(on page 774)* are designed to adapt to any device and screen size to provide an optimal viewing and interaction experience. Oxygen XML Developer Eclipse plugin also provides numerous possibilities for customizing the WebHelp Responsive output *(on page 848)*.

WebHelp Responsive Transformation Scenario

To publish a *DITA map (on page 1324)* as **WebHelp Responsive** output, follow these steps:

1. Select the **Configure Transformation Scenario(s)** action from the 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 built-in publishing templates *(on page 816)* that you can use for the layout of your WebHelp system output. You can also create your own publishing templates or edit existing ones *(on page 848)*.
   - 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.
   - Feedback Tab - This tab is for those who want to add the **Oxygen Feedback** comments component at the bottom of each WebHelp page so that you can interact with your readers.
   - 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.

Result: When the **DITA Map WebHelp Responsive** transformation is complete, the output is automatically opened in your default browser.

General Parameters for Customizing WebHelp Responsive Output

To customize a transformation scenario, you can edit various parameters, including the following most commonly used ones:

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`).

editlink.remote.ditamap.url

Use this parameter in conjunction with `editlink.web.author.url` to add an *Edit* link next to the topic title in the WebHelp output. When a user clicks the link, the topic is opened in Oxygen XML Web Author where they can make changes that can be saved to a file server. The value should be set as the custom URL of the *main DITA map*. For example, a GitHub custom URL might look like this: `https://getFileContent/oxyengxml/userguide/master/UserGuide.ditamap`.

editlink.web.author.url

This parameter needs to be used in conjunction with `editlink.remote.ditamap.url` to add an *Edit* link next to the topic title in the WebHelp output. When a user clicks the link, the topic is opened in Oxygen XML Web Author where they can make changes that can be saved to a file server. The value should be set as the URL of the Web Author installation. For example: `https://www.oxygenxml.com/oxygen-xml-web-author/`.

editlink.present.only.path.to.topic

When this parameter is set to "true", the DITA topic path is displayed to the right of each topic title in the WebHelp Responsive output. Also, when this parameter is used, the **editlink.ditamap.edit.url**, **editlink.remote.ditamap.url**, and **editlink.web.author.url** parameters are ignored.

fix.external.refs.com.oxygenxml (Only supported when the DITA-OT transformation process is started from Oxygen XML Developer Eclipse 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).

force.unique

When set to `true` (default value), the transformation will be forced to create unique output files for each instance of a resource when a map contains multiple references to a single topic.

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 for indexing Japanese content in the WebHelp pages. The encoding for the dictionary must be **UTF8**.

webhelp.search.enable.pagination

Specifies whether or not search results will be displayed on multiple pages. Allowed values are `yes` or `no`.

webhelp.search.page.numberOfItems

Specifies the number of search results items displayed on each page. This parameter is only used when the **webhelp.search.enable.pagination** parameter is enabled.

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 Oxygen WebHelp Responsive

webhelp.fragment.feedback

You can integrate **Oxygen Feedback** with your WebHelp Responsive output to provide a comments area at the bottom of each page where readers can offer feedback. When you create an **Oxygen Feedback** site configuration, an HTML fragment is generated during the final step of the creation process and that fragment should be set as the value for this parameter.

webhelp.default.collection.type.sequence

Specifies if the **sequence** value will be used by default when the `@collection-type` attribute is not specified. This option is helpful if you want to have *Next* and *Previous* navigational buttons generated for all HTML pages. Allowed values are **no** (default) and **yes**.

webhelp.enable.search.autocomplete

Specifies if the *Autocomplete* feature is enabled in the WebHelp search text field. The default value is `yes`.

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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment.

Important: This parameter should only be used if you are using a valid, purchased license of Oxygen XML Developer Eclipse plugin (do not use it with a trial license).

webhelp.fragment.head

In the generated output it inserts a given XHTML fragment in the `<head>` element. The value of the parameter can be either a **well-formed** 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 a **well-formed** 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.search.stop.words.exclude

Specifies a list of words that will be excluded from the default list of *stop words* that are filtered out before the search processing. Use comma separators to specify more than one word (for example: `if,for,is`).

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.publication.toc

Specifies if a table of contents will be presented on the left side of each topic in the output. The default value is `yes`.

webhelp.show.topic.toc

Specifies if a topic table of contents will be presented on the right side of each topic in the output. This table of contents contains links to each `<section>` within the current topic that contains an `@id` attribute and the section corresponding to the current scroll position is highlighted. 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.publication.toc.hide.chunked.topics

Specifies if the table of contents will contain links for *chunked* topics. The default value is `yes`.

webhelp.publication.toc.links

Specifies which links will be included in the table of contents. The possible values are:

- **chapter** (default) - The TOC will include links for the current topic, its children, its siblings, and its direct ancestor (including the direct ancestor's siblings), and the parent chapter.
- **topic** - The TOC will only include links for the current topic and its direct children.
- **all** - The TOC will include all links.

webhelp.publication.toc.tooltip.position

By default, if a topic contains a `<shortdesc>` element, its content is displayed in a tooltip when the user hovers over its link in the table of contents. This parameter controls whether or not this tooltip is displayed and its position relative to the link. The possible values are:

- **left** (default)
- **right**
- **top**
- **bottom**

- **hidden** - The tooltip will not be displayed.

webhelp.top.menu.depth

Specifies the maximum depth level of the topics that will be included in the top menu. The default value is `3`. A value of `0` means that the menu has unlimited depth.

webhelp.topic.collapsible.elements.initial.state

Specifies the initial state of collapsible elements (tables with titles, nested topics with titles, sections with titles, index term groups). The possible values are `collapsed` or `expanded` (default value).

Related Information:

Customizing WebHelp Responsive Output *(on page 848)*

Layout and Features *(on page 774)*

DITA Map PDF - based on HTML5 & CSS Transformation

Oxygen XML Developer Eclipse plugin includes a built-in **DITA Map PDF - based on HTML5 & CSS** transformation scenario based on a *DITA-OT CSS-based PDF Publishing plugin* that converts DITA maps to PDF using a CSS-based processing engine and an HTML5 intermediate format. Oxygen XML Developer Eclipse plugin comes bundled with a built-in CSS-based PDF processing engine called **Oxygen PDF Chemistry**. Oxygen XML Developer Eclipse plugin also supports some third-party processors.

For those who are familiar with CSS, this makes it very easy to style and customize the PDF output of your DITA projects without having to work with *xsl:fo* customizations. This transformation also includes some built-in publishing templates that you can use for the layout of your PDF output and you can create your own templates or edit existing ones.

The following CSS-based PDF processors can be used:

- **Oxygen PDF Chemistry** - A built-in processor that is bundled with Oxygen XML Developer Eclipse plugin. For more information, see the Oxygen PDF Chemistry User Guide. This is the supported processor.
- **Prince Print with CSS** (not included in the Oxygen XML Developer Eclipse plugin installation kit) - A third-party component that needs to be purchased from http://www.princexml.com.
- **Antenna House Formatter** (not included in the Oxygen XML Developer Eclipse plugin installation kit) - A third-party component that needs to be purchased from http://www.antennahouse.com/antenna1/formatter/.

How to Create the Transformation Scenario

To create a **DITA Map PDF - based on HTML5 & CSS** transformation scenario, follow these steps:

1. Click the **Configure Transformation Scenario(s)** button.
2. Select the **DITA Map PDF - based on HTML5 & CSS** transformation scenario.
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 built-in publishing templates that you can use for the layout of your WebHelp system output. You can also create your own publishing templates by saving one from the gallery and changing it.
- **Parameters** Tab - This tab includes numerous parameters that can be set to customize the transformation.
- **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. In the **Parameters** tab, configure any of the following parameters (if applicable):
   - **args.css** - Specifies a path to a custom CSS to be used in addition to those specified in the publishing template.
   - **css.processor.type**- This is where you choose the processor type. You can select between **Oxygen PDF Chemistry**, **Prince XML**, or **Antenna House**.
   - **css.processor.path.chemistry** (if you are using the **Oxygen PDF Chemistry** processor) - Specifies the path to the **Oxygen PDF Chemistry** executable file that will be run to generate the PDF. If this parameter is not set, the transformation will use the processor specified in the **CSS-based Processors** preferences page *(on page 118)*.
   - **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`.
   - **figure.title.placement** - Controls the position of the figure title relative to the image. Allowed values are "top" and "bottom", "top" is the default
5. Click **OK** and run the transformation scenario.

Customizing the Output

For information about customizing the output, see CSS-based DITA to PDF Customization *(on page 951)*.

Related Information:

Editing a Transformation Scenario *(on page 762)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

Oxygen PDF Chemistry User Guide

CSS-based DITA to PDF Customization *(on page 951)*

DITA Map PDF - based on XSL-FO Transformation

Oxygen XML Developer Eclipse plugin comes bundled with the DITA Open Toolkit that provides a mechanism for converting *DITA maps (on page 1324)* to PDF output.

Creating a DITA Map PDF - based on XSL-FO Transformation Scenario

To create a *DITA Map PDF - based on XSL-FO* transformation scenario, follow these steps:

1. Click the **Configure Transformation Scenario(s)** button.
2. Select **DITA Map PDF - based on XSL-FO** and click the **Edit** button (or use the **Duplicate** button if your *framework (on page 1325)* 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.
   - **editlink.present.only.path.to.topic** - When this parameter is set to "true", the DITA topic path is displayed to the right of each topic title in the PDF output.
4. Click **OK** and then the **Apply Associated** button to run the transformation scenario.

Related Information:

XSL FO-based DITA to PDF Customization *(on page 1107)*

DITA Map MS Office Word Transformation

Oxygen XML Developer Eclipse plugin comes bundled with a transformation scenario that allows you to convert *DITA maps (on page 1324)* to Microsoft Office Word documents. It utilizes the **DITA to Word plugin** created by Jarno Elovirta. This *plugin* contains a Word document named **Normal.docx** (located in: `[OXYGEN_INSTALL_DIR]/frameworks/dita/DITA-OT3.x/plugins/com.elovirta.ooxml/resources`) that is used by the transformation scenario as a template to generate the final Word document.

Tip: You can make general modifications to the **Normal.docx** template file to alter the published output.

Configuring the Transformation Scenario

To configure a **DITA Map to MS Office Word** transformation scenario, follow these steps:

1. Open the DITA map.
2. Click the **Configure Transformation Scenario(s)** button.
3. Select **DITA Map MS Office Word**.

4. For advanced customizations, in the **Parameters** tab you can use any of the following parameters that are unique to this transformation scenario to specify paths to files that affect the output in various ways:
    - **dotx.file** - Specifies the path to a Word template file (`.docx`) that will be used in the transformation to generate the final Word document. Set this parameter if you want to use a different template file other than the **Normal.docx** file that is used by default.
    - **document.flat.xsl** - Specifies the path to a pre-process clean-up stylesheet.
    - **core.xsl** - Specifies the path to a core metadata stylesheet.
    - **custom.xsl** - Specifies the path to a custom metadata stylesheet.
    - **document.xsl** - Specifies the path to a main document stylesheet.
    - **comments.xsl** - Specifies the path to a comments stylesheet.
    - **numbering.xsl** - Specifies the path to a list and title numbering stylesheet.
    - **footnotes.xsl** - Specifies the path to a footnote stylesheet.
    - **document.xml.xsl** - Specifies the path to a document relations metadata stylesheet.
    - **inkscape.exec** - Specifies the path to an Inkscape (open-source vector graphics editor) executable file.
5. Click **OK** and run the transformation scenario.

Result: The result of the transformation will automatically be opened in your system's default word processing application (such as Microsoft Word).

Related Information:

Editing a Transformation Scenario *(on page 762)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

DITA Map CHM (Compiled HTML Help) Transformation

To perform a *Compiled HTML Help (CHM)* transformation, Oxygen XML Developer Eclipse plugin needs `Microsoft HTML Help Workshop` to be installed on your computer. Oxygen XML Developer Eclipse plugin automatically detects if `HTML Help Workshop` is installed 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 Eclipse plugin uses the `htmlhelp.locale` parameter to correctly display specific characters of different languages in the output of the *Compiled HTML Help (CHM)* transformation. By default, the **DITA Map CHM** transformation scenario that comes bundled with Oxygen XML Developer Eclipse plugin has the `htmlhelp.locale` parameter set to `en-US`.

To customize this parameter, follow this procedure:

1. 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 **DITA Map CHM** transformation scenario and click the **Edit** button.
3. In the **Parameter** tab, search for the `htmlhelp.locale` parameter and change its value to the desired language tag.

   Note: 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.

Customizing the CHM Output

There are several possibilities available for customizing the CHM output:

- You can use a custom CSS stylesheet to customize how the HTML content is rendered in the output:
  1. Create the custom CSS.
  2. Select the **DITA Map CHM** transformation scenario and click the **Edit** button.
  3. In the **Parameter** tab, set the `args.css` parameter to point to the location of your custom CSS and make sure the `args.copy.css` parameter is set to **yes** to instruct the transformation to copy the custom CSS to the output folder.
  4. Run the transformation.
- If you are familiar with XSLT, there are two XSLT stylesheets that are used in the transformation to compile various settings and components in the CHM output. They are found in the following directory: `OXYGEN_INSTALL_DIR/frameworks/dita/DITA-OT3.x/plugins/org.dita.htmlhelp/xsl/map2htmlhelp`. The files are as follows:
  - **map2hhcImpl.xsl** - This file is used to compile the table of contents.
  - **map2hhpImpl.xsl** - This file contains information for compiling the CHM and various settings that are read by the *HTML Help Workshop* when creating the output.

DITA Map Kindle Transformation

Oxygen XML Developer Eclipse plugin requires *KindleGen* to generate Kindle output from *DITA maps (on page 1324)*. To install *KindleGen* for use by Oxygen XML Developer Eclipse 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 Eclipse plugin and open a *DITA map*.
4. Click the **Configure Transformation Scenario(s)** button.
5. Select the **DITA Map Kindle** transformation and click 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.

Run DITA-OT Integrator Transformation

Oxygen XML Developer Eclipse plugin comes bundled with a transformation scenario designed to integrate DITA-OT plugins *(on page 1327)*. These DITA-OT *plugins* are used for various customizations. It is called **Run DITA-OT Integrator** and is found in the **DITA Map** section of the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

⚠ **Attention:** The integration will be performed on the DITA-OT version specified in the **DITA Open Toolkit** section of the **DITA** preferences page *(on page 47)*.

Running the Transformation Scenario

To integrate a DITA-OT *plugin*, follow these steps:

1. If Oxygen XML Developer Eclipse plugin was installed in the default location, you may need to restart and run it as an administrator.
2. Select the **Apply Transformation Scenario(s)** or **Configure Transformation Scenario(s)** *(on page 765)* action from the **DITA Maps Manager** toolbar (you could also use the same action on the main toolbar or open the **Transformation Scenarios** view *(on page 770)*).
3. Select the **Run DITA-OT Integrator** transformation scenario. If the *integrator* is not visible, select the **Show all scenarios** action that is available in the **Settings** drop-down menu.
4. Apply the scenario *(on page 764)*.
5. Check the **Results** panel at the bottom of the application to make sure the build was successful.
6. Restart Oxygen XML Developer Eclipse plugin with your normal permissions.

Related Information:

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

Solving 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 *(on page 192)*.
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 *(on page 716)*.

4. If the transformation results in errors or warnings, they are displayed in the **Results** panel *(on page 248)* 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 the **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**.

DITA Topic Transformation Scenarios

Oxygen XML Developer Eclipse plugin includes built-in transformation scenarios for transforming individual DITA Topics to HTML5, XHTML, or PDF output. They can be found in the **DITA** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

The available transformations scenarios for individual DITA topics include:

- **DITA HTML5** - This DITA-OT transformation scenario generates HTML5 output from a single DITA topic.
- **DITA XHTML** - This DITA-OT transformation scenario generates XHTML output from a single DITA topic. This was the first transformation scenario created for the DITA Open Toolkit and it originally served as the basis for all HTML-based transformations.
- **DITA PDF - based on HTML5 & CSS** - This transformation scenario converts individual DITA topics to PDF using a CSS-based processing engine and an HTML5 intermediate format. Oxygen XML Developer Eclipse plugin comes bundled with a built-in CSS-based PDF processing engine called **Oxygen PDF Chemistry**. Oxygen XML Developer Eclipse plugin also supports some third-party processors.

For those who are familiar with CSS, this makes it very easy to style and customize the PDF output of your DITA projects without having to work with *xsl:fo* customizations. Another advantage of this transformation scenario is that you can use the same customization CSS *(on page 981)* or publishing template *(on page 962)* that you use for converting entire DITA maps.

- **DITA PDF - based on XSL-FO** - This DITA-OT transformation scenario converts individual DITA topics to PDF using an *xsl:fo* processor.

Related Information:

Editing a Transformation Scenario *(on page 762)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

Applying Associated Transformation Scenarios *(on page 764)*

DITA Map Transformation Scenarios

DocBook Transformation Scenarios

Built-in transformation scenarios allow you to transform DocBook documents to a variety of outputs, such as WebHelp, PDF, HTML, HTML Chunk, XHTML, XHTML Chunk, and EPUB. Oxygen XML Developer Eclipse plugin also includes a DocBook 5.1 transformation scenario for *Assembly* documents. All of them are listed in the **DocBook 4** and **DocBook 5** sections in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Related Information:

Editing a Transformation Scenario *(on page 762)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

Applying Associated Transformation Scenarios *(on page 764)*

DocBook to WebHelp Classic Transformation

DocBook documents can be transformed into several types of WebHelp systems (with or without a feedback section). The WebHelp Classic layout and features *(on page 915)* are designed for desktop systems and include a familiar classical style. Oxygen XML Developer Eclipse plugin also provides numerous possibilities for customizing the WebHelp Classic output *(on page 927)*.

WebHelp Classic Transformation Scenario

To publish a DocBook 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** or **DocBook 5** section.
3. Click **Apply associated**.

Result: When the **DocBook WebHelp Classic** transformation is complete, the output is automatically opened in your default browser.

WebHelp Classic with Feedback Transformation Scenario

To publish a DocBook 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** or **DocBook 5** section.
3. Click **Apply associated**.
4. Enter the documentation product ID and the documentation version.

Result: 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 System.

For more information about the feedback-enabled WebHelp system, watch our video demonstration:

https://www.youtube.com/embed/eoQ2uxHvppE

Customizing DocBook WebHelp Transformation Scenarios

To customize a DocBook WebHelp transformation scenario, you can edit various parameters, including the following most commonly used ones:

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 exert is an example for adding a Facebook™ widget:

```
<div id="facebook">
  <div id="fb-root"/>
  <script>
    <!-- (function(d, s, id) { var js, fjs = d.getElementsByTagName(s)[0];
      if (d.getElementById(id)) return;
      js = d.createElement(s); js.id = id;
      js.src = "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
      fjs.parentNode.insertBefore(js, fjs); }
       (document, 'script', 'facebook-jssdk')); -->
    </script>
    <div data-share="true" data-show-faces="true" data-action="like"
       data-layout="standard" class="fb-like"/>
</div>
```

webhelp.footer.include

Specifies whether or not to include footer in each WebHelp page. Possible values: `yes`, `no`. If set to `no`, no footer is added to the WebHelp pages. If set to `yes` and the `webhelp.footer.file` parameter has a value, then the content of that file is used as footer. If the `webhelp.footer.file` has no value then a default **Oxygen** 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.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 WebHelp Classic output. It can be one of the built-in skin CSS from the `OXYGEN_INSTALL_DIR\frameworks\docbook\xsl\com.oxygenxml.webhelp.classic\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:

Customizing WebHelp Classic Output *(on page 927)*

DocBook to DITA Transformation

Oxygen XML Developer Eclipse 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 762)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

DocBook to PDF Transformation

Oxygen XML Developer Eclipse plugin includes a built-in transformation scenario that is designed to convert DocBook content to PDF.

To convert a DocBook document to PDF, 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 PDF** transformation scenario in the **DocBook 4** or **DocBook 5** section.
3. Click the **Apply associated** button to run the transformation.

For information about customizing the PDF output for DocBook content, see DocBook to PDF Output Customization *(on page 1117)*.

Related Information:

Editing a Transformation Scenario *(on page 762)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

DocBook to PDF Output Customization *(on page 1117)*

DocBook to EPUB Transformation

Oxygen XML Developer Eclipse plugin includes a built-in transformation scenario that is designed to convert DocBook content to EPUB. The EPUB specification recommends the use of *OpenType* fonts (recognized by their `.otf` file extension) whenever 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 *(on page 765)*, select the **DocBook EPUB** transformation scenario in the **DocBook 4** or **DocBook 5** section, 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 PDF (Show Change Tracking and Comments)

Oxygen XML Developer Eclipse plugin includes a built-in transformation scenario that is designed to show tracked changes and comment in DocBook to PDF output.

To include comments and *tracked changes* (stored within your DocBook 5 documents) in the PDF output, follow these steps:

1. Click the **Configure Transformation Scenario(s)** button.
2. Select **DocBook PDF (Show Change Tracking and Comments)** in the **DocBook 5** section.
3. If you need to configure the transformation, click the **Edit** *(on page 762)* or **Duplicate** *(on page 764)* button, make your changes to the scenario, and click **OK**.
4. Click 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.

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 new scenarios that are available in Oxygen XML Developer Eclipse 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**.
- Go to **Window > Show View** and select **Transformation Scenarios** to display this view *(on page 770)*. Click the **New Scenario** drop-down menu button and select **XML transformation with XSLT**.

Both 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 *(on page 691)* or edit an existing one *(on page 762)*, 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** *(on page 148)* button, or the browsing actions 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 *(on page 143)*, 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 *(on page 693)* 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 *(on page 706)*. 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** *(on page 148)* button, or the browsing actions 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 selected, the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction. By default, this option is deselected 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 Eclipse plugin for performing a transformation. These include the built-in engines and the external engines defined in the Custom Engines preferences page *(on page 130)*. 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 *(on page 695)* for the current transformation scenario. To configure the same options globally, go to the Saxon-HE/PE/EE preferences page *(on page 140)*. For the current transformation scenario, these **Advanced options** override the options configured in that preferences page.

Parameters

Opens a **Configure parameters** dialog box *(on page 693)* 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 *(on page 693)*. If the XSLT transformation engine is custom-defined, you cannot 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 *(on page 695)* that define extension Java functions or extension XSLT elements used in the transformation.

Additional XSLT stylesheets

Opens a dialog box for adding XSLT stylesheets *(on page 695)* 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 *(on page 695)*, 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 selected, the parameter value is evaluated as an XPath expression before starting the XSLT transformation.

Example:

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 *(on page 148)* (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 *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is selected, 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 *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is selected, 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 available 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 148)*

XSLT Extensions

The **Extensions** button opens a dialog box that allows you to specify the *JARS (on page 1326)* 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.

Tip: You can specify the path to the resources using wildcards (for example, `${oxygenHome}/lib/*.jar`).

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 click 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 *(on page 148)* 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 *(on page 140)* 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.

Saxon-HE/PE/EE Options

The advanced options for Saxon 9.9.1.5 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. If this value is also specified in a configuration file *(on page 696)*, the value in this option takes precedence.

Template ("-it")

A Saxon-specific option that sets the name of the initial XSLT template to be executed. If this value is also specified in a configuration file *(on page 696)*, the value in this option takes precedence.

Tip: If your stylesheet includes `<xsl:template name="xsl:initial-template">`, Oxygen XML Developer Eclipse plugin will automatically detect and use it as the initial template, so this option is not needed in this case.

Use a configuration file ("-config")

Select this option if you want to use a Saxon 9.9.1.5 configuration file that will be executed for the XSLT transformation and validation processes. You can specify the path to the configuration file by entering it in the **URL** field, or by using the **Insert Editor Variables** button, or using the browsing actions in the **Browse** drop-down list.

Debugger trace into XPath expressions (applies to debugging sessions)

Instructs the XSLT Debugger *(on page 1227)* to *step into* XPath expressions.

Enable Optimizations ("-opt")

This option is selected by default, which means that optimization is enabled. If not selected, the optimization is suppressed, which is helpful when reducing the compiling time is important, optimization conflicts with debugging, or optimization causes extension functions with side-effects to behave unpredictably.

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")

Specifies 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")

Specifies 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.

Saxon-PE/EE Options

The following advanced options are specific for Saxon 9.9.1.5 Professional Edition (PE) and Enterprise Edition (EE) only:

Register Saxon-JS extension functions and instructions

Registers the Saxon-CE extension functions and instructions when compiling a stylesheet using the Saxon 9.9.1.5 processors.

Note: Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Developer Eclipse 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 selected, 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.

Enable assertions ("-ea")

In XSLT 3.0, you can use the `<xsl:assert>` element to make assertions in the form of XPath expressions, causing a dynamic error if the assertion turns out to be false. If this option is selected, XSLT 3.0 `<xsl:assert>` instructions are enabled. If it is not selected (default), the assertions are ignored.

Saxon-EE Options

The advanced options that are specific for Saxon 9.9.1.5 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 *(on page 36)* and go to **XML > XML Parser > XML Schema** and use the **Default XML Schema version** option *(on page 126)*.

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 allows for parsing the source documents with strict schema-validation enabled.
- **Lax schema validation ("lax")** - If an XML Schema is provided, this mode allows for 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. Selecting 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.

Enable streaming mode

Selecting this option will allow an XSLT to run in streaming mode. It is not selected by default.

Other Options

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.

Using Saxon Integrated Extension Functions

Saxon, the transformation and validation engine used by Oxygen XML Developer Eclipse plugin, can be customized by adding custom functions (called Integrated Extension Functions) that can be called from XPath.

To define such a function, follow these steps:

1. Create a file with a Java class that extends `net.sf.saxon.lib.ExtensionFunctionDefinition`. Here is an example:

```
private static class ShiftLeft extends ExtensionFunctionDefinition {
  @Override
  public StructuredQName getFunctionQName() {
    return new StructuredQName("eg", "http://example.com/saxon-extension", "shift-left");
  }
  @Override
  public SequenceType[] getArgumentTypes() {
   return new SequenceType[] {SequenceType.SINGLE_INTEGER, SequenceType.SINGLE_INTEGER};
  }
  @Override
  public SequenceType getResultType(SequenceType[] suppliedArgumentTypes) {
   return SequenceType.SINGLE_INTEGER;
  }
  @Override
  public ExtensionFunctionCall makeCallExpression() {
   return new ExtensionFunctionCall() {
   public SequenceIterator call(SequenceIterator[] arguments, XPathContext context)
                    throws XPathException {
       long v0 = ((IntegerValue)arguments[0].next()).longValue();
    long v1 = ((IntegerValue)arguments[1].next()).longValue();
    long result = v0<<v1;
        return Value.asIterator(Int64Value.makeIntegerValue(result));
      }
    };
  }
}
```

2. Compile the class and add it to a JAR file.
3. Add a file called **net.sf.saxon.lib.ExtensionFunctionDefinition** that contains the fully qualified name of the Java class in the `META-INF/services/` folder of the JAR file.

Note: To add more function definitions in the same JAR file, you need to add their fully qualified names on different lines.

To enable Oxygen XML Developer Eclipse plugin to pick up your custom function definition, the JAR file should be added to the classpath of the transformer. Here are some possibilities:

- If you develop a framework, you just need to link the JAR file in the **Classpath** tab *(on page 55)*.
- In a validation scenario *(on page 288)*, you can use the **Extensions** button to open a dialog box where you can add libraries.
- In a transformation scenario, you can use the **Extensions** button in the **XSLT** tab *(on page 693)* to open a dialog box where you can add libraries.

FO Processor Tab (XSLT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 115)*.

Output Tab (XSLT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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** *(on page 148)* button, or the **Browse** button.

Open in Browser/System Application

If selected, Oxygen XML Developer Eclipse 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** *(on page 148)* button, or the **Browse** button.

Open in editor

When this is option is selected, 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 Eclipse plugin displays the transformation result in an XML viewer panel at the bottom of the application window with syntax highlighting *(on page 101)*.
- **XHTML** - This option is only available if **Open in Browser/System Application** is not selected. If selected, Oxygen XML Developer Eclipse 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 selecting this option 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.

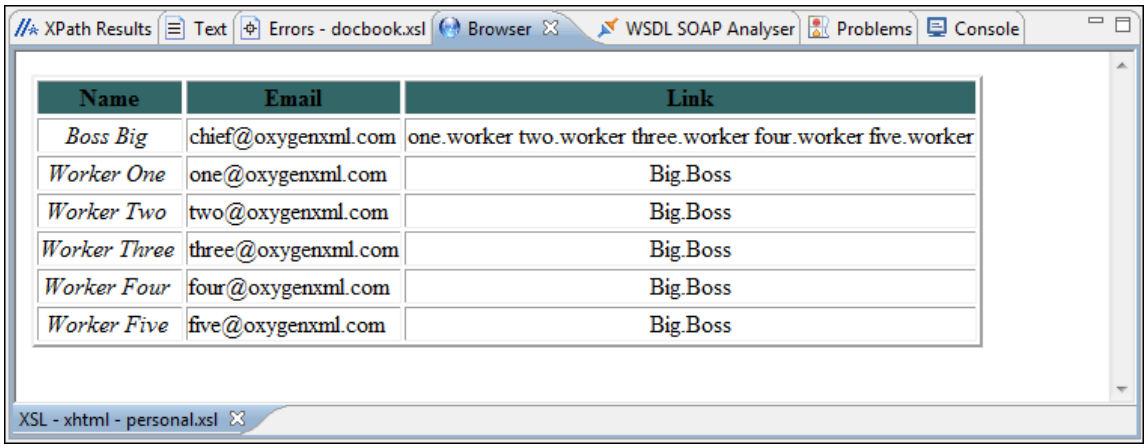
- **Image URLs are relative to** - If **Show in results view as XHTML** is selected, this option specifies the path used to resolve image paths contained in the transformation result. You can specify the path by using the text field, the **Insert Editor Variables** *(on page 148)* button, or the **Browse** button.

⚠ **Attention:** If your input XSLT contains `<xsl:result-document>` elements, then the secondary results will be saved to the specified URIs while the principal result is specified in this **Output** tab. For more information, see: https://www.w3.org/TR/xslt-30/#element-result-document.

Oxygen XML Developer Eclipse plugin Browser View

The Oxygen XML Developer Eclipse 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.

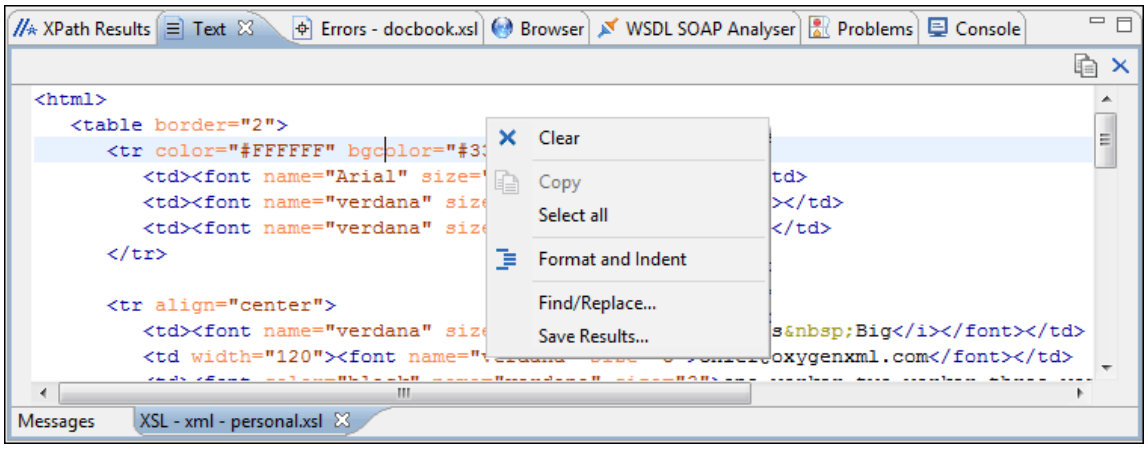
Figure 273. Browser View



Oxygen XML Developer Eclipse plugin Text View

The Oxygen XML Developer Eclipse 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 274. 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 (on page 1327)*) the content in this view.

Find/Replace

Allows you to perform search and replace operations.

Save Results

Saves the content in the view to a file in text format.

Configuring an XSLT Processor for Generating Output

This section explains how to configure an XSLT processor and extensions for such a processor in Oxygen XML Developer Eclipse plugin.

Supported XSLT Processors

Oxygen XML Developer Eclipse plugin includes the following XSLT processors:

- **Xalan 2.7.2** - 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 9.9.1.5 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 9.9.1.5 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 Eclipse 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 XML Schema 1.0 or 1.1. This can be configured in Preferences *(on page 126)*.

Note: Oxygen XML Developer Eclipse 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 Eclipse 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 9.9.1.5 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 Eclipse plugin.

- **Saxon-CE (Client Edition)** is Saxonica's implementation of XSLT 2.0 for use on web browsers. Oxygen XML Developer Eclipse plugin provides support for editing stylesheets that contain Saxon-CE extension functions and instructions. This support improves the validation, content completion, and syntax highlighting.

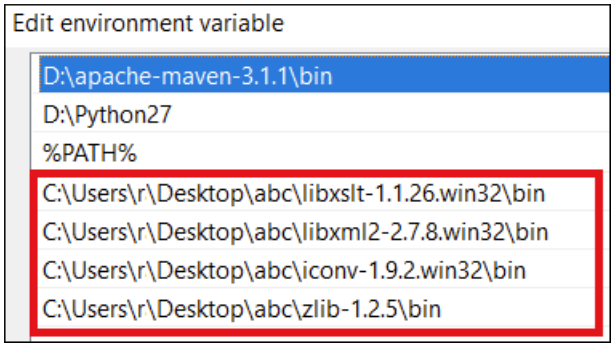
  Note: Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Developer Eclipse plugin only for developing the Saxon-CE stylesheet, leaving the execution part to a web browser. See more details about executing such a stylesheet on Saxonica's website.

  Note: A specific template, named **Saxon-CE stylesheet**, is available in the **New from Templates** wizard *(on page 177)*.
- **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.

  Oxygen XML Developer Eclipse plugin uses **Libxslt** through its command-line tool (*Xsltproc*). Depending on your operating system, you must download the Libxslt libraries on your machine from http://xmlsoft.org/XSLT/downloads.html and place them in a local folder. Then you need to update the `PATH` environmental variable to contain the parent folder where the `xsltproc` executable is located.

  Tip: As an example, a Windows installation of the Xsltproc engine would follow these steps:

1. Go to http://ftp.zlatkovic.com/libxml.en.html and download the following ZIP files: **iconv-1.9.2.win32.zip**, **libxml2-2.7.8.win32.zip**, **libxslt-1.1.26.win32.zip**, **zlib-1.2.5.win32.zip**.
2. Unzip all of them into the same folder of your choice.
3. Edit the `PATH` environment variable and add the `bin` folder for all four archives:



4. Restart Oxygen XML Developer Eclipse plugin.

Result: You can now use the **xsltproc** processor as an XSLT engine in the XSLT transformation scenario.

Note: The Xsltproc processor can be configured from the **XSLTPROC** options page *(on page 143)*.

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 (on page 1329)* files of the built-in document types (DocBook, TEI, DITA, etc.) are not handled properly by LIBXML if Oxygen XML Developer Eclipse 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 *(on page 691)* and validation of XSLT stylesheets *(on page 366)*.

  Oxygen XML Developer Eclipse 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 *(on page 691)* and validation of XSLT stylesheets *(on page 366)*.

  Oxygen XML Developer Eclipse plugin performs XSLT transformations and validations using the .NET *Framework* XSLT implementation (*System.Xml.Xsl.XslTransform* class) through the **nxslt** command-line utility. The **nxslt** version included in Oxygen XML Developer Eclipse 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 Eclipse 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 *(on page 36)* 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 *(on page 130)*.
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 Eclipse plugin linked message *(on page 285)*, 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 130)*

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 9.9.1.5** - 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 *(on page 693)* in the **Edit scenario** dialog box.

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 to have an XQuery file query 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**.
- Go to **Window > Show View** and select **Transformation Scenarios** to display this view *(on page 770)*. Click the **New Scenario** drop-down menu button and select **XML transformation with XQuery**.

Both 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 *(on page 691)* or edit an existing one *(on page 762)*, 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** *(on page 148)* button, or the browsing actions 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 *(on page 143)*, 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** *(on page 148)* button, or the browsing actions 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 Eclipse plugin for performing a transformation. These include the built-in engines and the external engines defined in the Custom Engines preferences page *(on page 130)*. 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 *(on page 710)* for the current transformation scenario. To configure the same options globally, go to the Saxon-HE/PE/EE preferences page *(on page 140)*. For the current transformation scenario, these **Advanced options** override the options configured in that preferences page.

Parameters

Opens the **Configure parameters** dialog box *(on page 708)* 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 *(on page 710)* 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 selected, the parameter value is evaluated as an XPath expression before starting the XQuery transformation.

Example:

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 *(on page 148)* (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 *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is selected, 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 *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is selected, 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 available 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 148)*

XQuery Extensions

The **Extensions** button is used to specify the *JAR (on page 1326)* 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 click 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 *(on page 134)* 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.

Saxon-HE/PE/EE Options

The advanced options for Saxon 9.9.1.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

Use a configuration file ("-config")

Sets a Saxon 9.9.1.5 configuration file that is used for XQuery transformation and validation scenarios.

Enable Optimizations ("-opt")

This option is selected by default, which means that optimization is enabled. If not selected, the optimization is suppressed, which is helpful 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.

Recoverable errors ("-warnings")

Specifies 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")

Specifies 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.

Saxon-PE/EE Options

The following advanced options are specific for Saxon 9.9.1.5 Professional Edition (PE) and Enterprise Edition (EE) only:

Allow calls on extension functions ("-ext")

If selected, calls on external functions are allowed. Selecting 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.

Saxon-EE Options

The advanced options that are specific for Saxon 9.9.1.5 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 allows for parsing the source documents with strict schema-validation enabled.
- **Lax schema validation ("lax")** - If an XML Schema is provided, this mode allows for 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. Selecting 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.

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 selected, backup versions for any XML files updated with an XQuery Update are generated. This option is available when the **Enable XQuery update** option is selected.

Other Options

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.

FO Processor Tab (XQuery Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 115)*.

Output Tab (XQuery Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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

Selecting 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** *(on page 148)* button, or the **Browse** button.

Open in Browser/System Application

If selected, Oxygen XML Developer Eclipse 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** *(on page 148)* button, or the **Browse** button.

Open in editor

When this is option is selected, 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 Eclipse plugin displays the transformation result in an XML viewer panel at the bottom of the application window with syntax highlighting *(on page 101)*.

- **XHTML** - This option is only available if **Open in Browser/System Application** is not selected. If selected, Oxygen XML Developer Eclipse 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 selecting this option 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** *(on page 148)* button, or the **Browse** button.

XML to PDF Transformation with CSS

This type of transformation uses the **Oxygen PDF Chemistry** processing engine to obtain PDF output by applying CSS styling to the edited XML document. This scenario is useful for those who are familiar with CSS and want to obtain PDF output as its final form.

To create an **XML to PDF transformation with CSS** 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 to PDF transformation with CSS**.
- Go to **Window > Show View** and select **Transformation Scenarios** to display this view *(on page 770)*. Click the **New Scenario** drop-down menu button and select **XML to PDF transformation with CSS**.

Both 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.

For more information about the **Oxygen PDF Chemistry** processing engine and numerous tips for customizing the output, see the Oxygen Chemistry User Guide.

CSS Tab (XML to PDF Transformation with CSS)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **CSS** 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** *(on page 148)* button, or the browsing actions 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.

CSS URL

Optionally, you can use this option to specify the location of a custom CSS file to be applied to the transformation. If this option is left blank, only the CSS referenced directly from the document will be applied. You can specify the path by using the text field, its history drop-down, the **Insert Editor Variables** *(on page 148)* button, or the browsing actions in the **Browse** drop-down list.

Apply CSS stylesheets set in the current framework

If selected, CSS stylesheets that are specified in the framework (in the Document Type configuration CSS subtab *(on page 56)*) are applied to the transformation in addition to any CSS referenced directly in the document or specified in the CSS URL field *(on page 715)*.

Note: If CSS files are specified in multiple ways, the transformation applies the CSS in the following order (from lowest priority to highest):

- CSS files that are specified in the framework (in the Document Type configuration CSS subtab *(on page 56)*).
- CSS files referenced directly in the document.
- CSS files specified in the CSS URL field *(on page 715)*.

Processor options link

Opens the **CSS-based Processors** preferences page *(on page 118)* where you can configure some options for generating PDF output.

Output Tab (XML to PDF Transformation with CSS)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Output** tab contains the following options:

Output File section

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** *(on page 148)* button, or the **Browse** button.

Open in Browser/System Application

If selected, Oxygen XML Developer Eclipse plugin automatically opens the result of the transformation in a system application associated with the PDF file type (for example, in Windows *PDF* files are often opened in *Acrobat Reader*).

Debugging section

Dump the intermediate annotated XML

Select this option to include (dump) the intermediate, annotated XML file in the same location as the output file. This can be used for debugging purposes.

Dump the FO file

Select this option to include (dump) the FO file (before it is converted to PDF) in the same location as the output file. This can be used for debugging purposes.

Console options link

Opens the **CSS-based Processors** preferences page *(on page 118)* where you can configure some options for generating PDF output.

DITA-OT Transformation

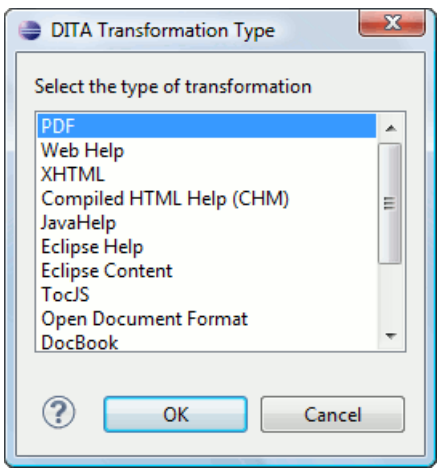
This type of transformation specifies the parameters for an Ant transformation that executes a DITA-OT build script. Oxygen XML Developer Eclipse plugin includes a built-in version of Ant and a built-in version of DITA-OT, but other versions can be set in the scenario.

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 **DITA Maps Manager** toolbar, main toolbar, or the **XML** menu. Then click the **New** button and select **DITA-OT Transformation**.
- Go to **Window > Show View** and select **Transformation Scenarios** to display this view *(on page 770)*. Click the **New Scenario** drop-down menu button and select **DITA-OT Transformation**.

Both methods open the **DITA Transformation Type** dialog box that presents the list of possible outputs.

Figure 275. 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.

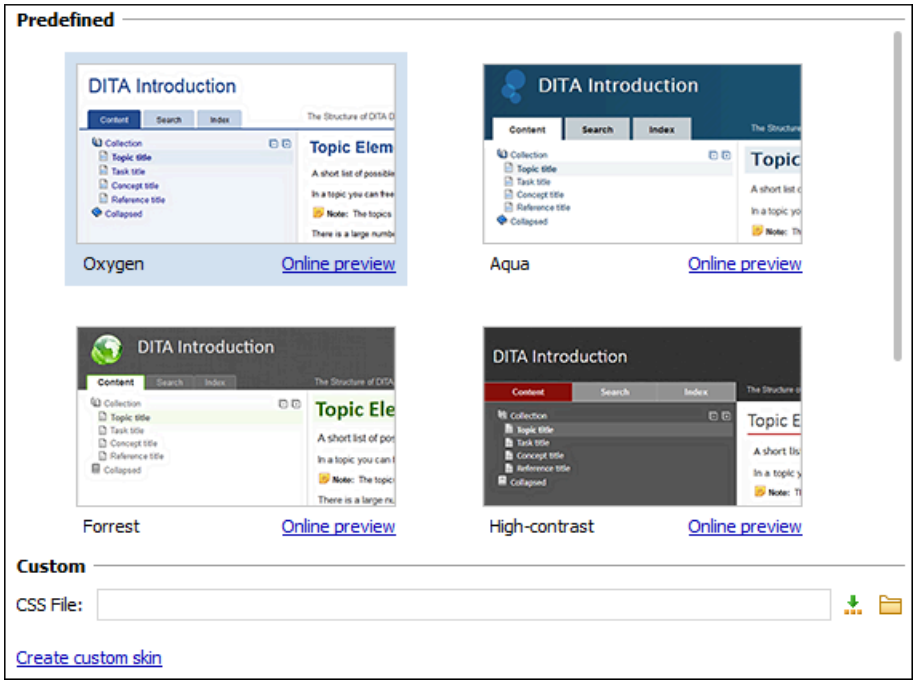
Skins Tab (DITA-OT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 built-in 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 276. Skins Tab



The **Skins** tab includes the following sections:

Built-in Skins

This section presents the built-in skins that are included in Oxygen XML Developer Eclipse plugin. The built-in 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 Eclipse plugin website, click 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

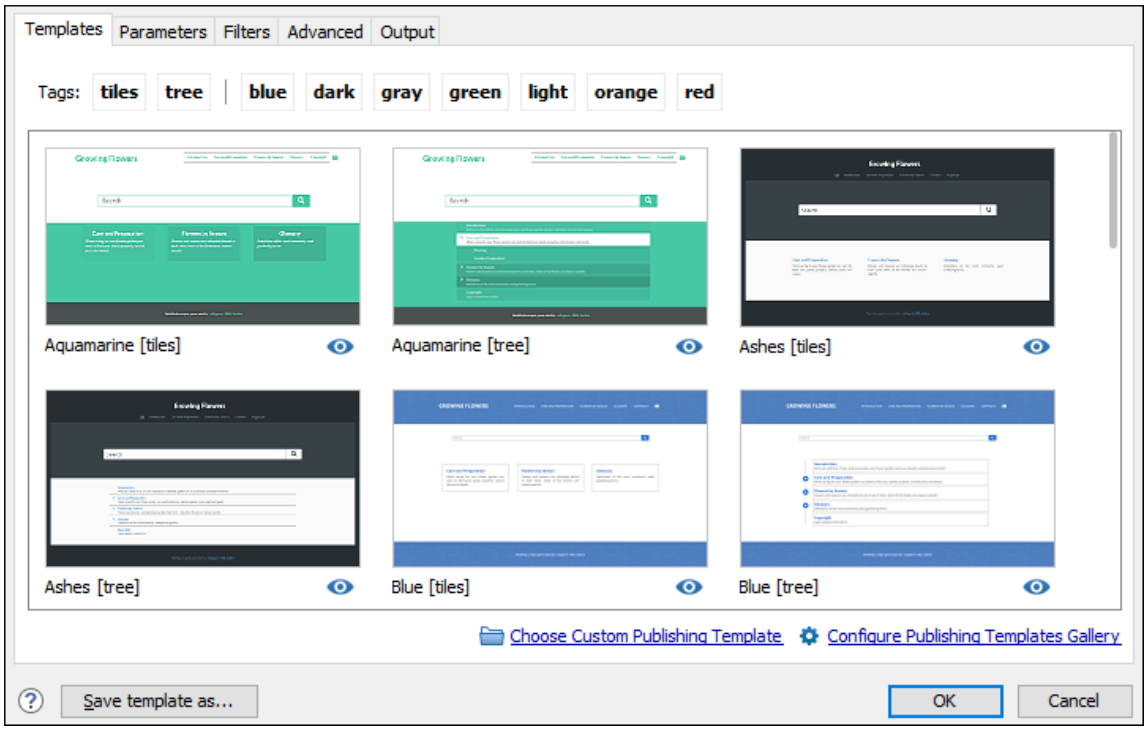
Use this link to open the **WebHelp Skin Builder** *(on page 927)* tool.

Templates Tab (DITA-OT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 **PDF - based on HTML5 & CSS** output types and it provides a set of built-in publishing templates *(on page 816)*. You can use one of them to publish your documentation or as a starting point for a new publishing template.

Figure 277. Templates Tab



Filtering and Previewing Templates

You can click on the tags at the top of the pane to filter the templates and narrow your search. Each built-in template also includes an **Online preview** icon in the bottom-right corner that opens a webpage in your

default browser providing a sample of how the main page will look when that particular template is used to generate the output.

Built-in Templates Locations

Oxygen XML Developer Eclipse plugin scans the following locations to find the built-in templates to display in the dialog box:

- **WebHelp Responsive Templates** - All built-in WebHelp Responsive publishing templates are stored in the following directory: *DITA-OT-DIR*`/plugins/com.oxygenxml.webhelp.responsive/templates`.
- **PDF - based on HTML5 & CSS** - All built-in PDF publishing templates are stored in the following directories:
  - *DITA-OT-DIR*`/plugins/com.oxygenxml.pdf.css/templates`
  - *DITA-OT-DIR*`/plugins/com.oxygenxml.webhelp.responsive/templates`

Custom Templates Locations

Oxygen XML Developer Eclipse plugin scans the locations specified in the **DITA > Publishing** preferences page *(on page 49)* to find custom templates to display in the dialog box. You can access that preferences page directly from the **Template** tab by clicking on the **Configure Publishing Templates Gallery** link.

Selecting Custom Templates

Once you are finished configuring your template, you can click the **Choose Custom Publishing Template** link to select your template.

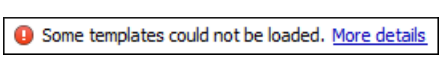
You can also add your custom templates *(on page 850)* to the list of templates displayed in the **Templates** tab. To do this, store them in a directory, then click the **Configure Publishing Templates Gallery** link to open the **DITA > Publishing** preferences page *(on page 49)* where you can add that directory to the list. All the templates contained in your template directory will be displayed in the preview pane along with all the built-in templates.

Save Template As Button

You can use the **Save template as** button (at the bottom-left of the transformation dialog box) to export the currently selected template into a new template package that can be used as a starting point to create your own custom template *(on page 968)*. Clicking this button will open a template package configuration dialog box that contains some options and displays the parameters that will be exported to your template package.

Template Errors

When the **Templates** tab is opened, all templates (built-in and custom) are loaded and validated. Specifically, certain elements in the template descriptor file are checked for validity. If errors are encountered that prevents the template from loading, the following message will be displayed toward the bottom of the dialog box:



If you click the **More details** link, a window will open with more information about the encountered error. For example, it might offer a hint that the element is missing from the expected descriptor file structure.

Also, if a template could be loaded, but certain elements could not be found in the descriptor file, a warning icon (⚠) will be displayed on the template's image (in the **Templates** tab of the transformation dialog box). For example, this happens if a valid preview-image element cannot be found.

Sharing Publishing Template

To share a publishing template with others, following these steps:

1. Copy your template in a new folder.
2. Go to **Options > Preferences > DITA > Publishing** *(on page 49)* and add that new folder to the list.
3. Switch the option as the bottom of that preferences page to **Project Options**.
4. Share your project file (`.xpr`).

For more information about customizing publishing templates, watch our video demonstration:

https://www.youtube.com/embed/zNmXfKWXwO8

Related Information:

Publishing Templates *(on page 816)*

Publishing Template Package Contents for PDF Customizations *(on page 963)*

Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

Template Package Configuration Dialog Box

The **Save template as** button (at the bottom-left of the transformation dialog box for **WebHelp Responsive** or **PDF - based on HTML5 & CSS** transformations) can be used to export the currently selected template into a new template package that can be used as a starting point to create your own custom template *(on page 968)*. The result will be a ZIP archive that contains a template descriptor file and other resources (such as CSS files) that were attached to the selected template.

Clicking the **Save template as** button opens a template package configuration dialog box contains the following options and components:

Name

Required field used to specify the name for the new template. This will become the text value of the `<name>` element in the template descriptor file. This information is displayed as the name of the template in the transformation scenario dialog box.

Description

Optional field used to specify a template description. This will become the text value of the `<description>` element in the template descriptor file. This information is displayed when the user hovers over the template in the transformation scenario dialog box.

Parameter Table

This table displays the parameters that will be exported. Only certain relevant parameters are exported. The parameters and their values will be inserted in the `<parameters>` section of the

template descriptor file. If any of the parameter values point to a file path that references a template resource (such as CSS files, custom HTML fragments, images), those resources will automatically be copied to the new template package and their references will be changed accordingly.

Note: Additional resources that are referenced in CSS files or other resources will not be copied to the new template package, so you will need to copy them manually and update their references in the template descriptor file.

Include WebHelp Customization

The same publishing template package can contain both a WebHelp Responsive and PDF customization and you can use the same template in both types of transformations (**DITA Map WebHelp Responsive** *(on page 672)* or **DITA Map to PDF - based on HTML5 & CSS** *(on page 679)*). This option specifies that the custom template will include a WebHelp Responsive customization.

Include HTML Page Layout Files

For **WebHelp Responsive** customizations, select this option if you want to copy the default *HTML Page Layout Files (on page 830)* into your template package. They are helpful if you want to change the structure of the generated HTML pages.

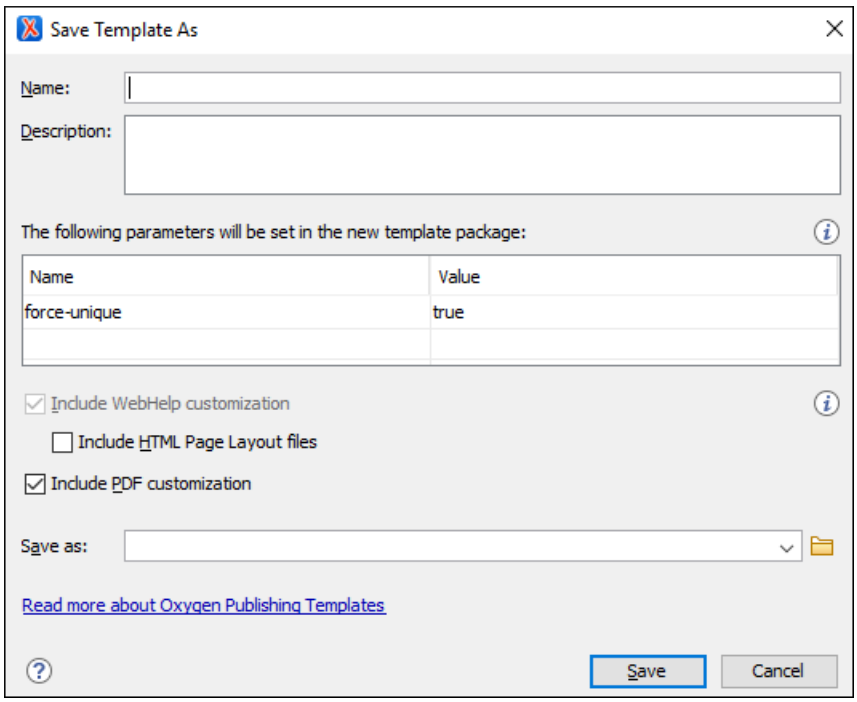
Include PDF Customization

The same publishing template package can contain both a WebHelp Responsive and PDF customization and you can use the same template in both types of transformations (**DITA Map WebHelp Responsive** *(on page 672)* or **DITA Map to PDF - based on HTML5 & CSS** *(on page 679)*). This option specifies that the custom template will include a PDF customization.

Save as

Use this field to specify the name and path of the ZIP file where the template will be saved.

Figure 278. Template Package Configuration Dialog Box



Related Information:

Publishing Templates *(on page 816)*

Publishing Template Package Contents for PDF Customizations *(on page 963)*

Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

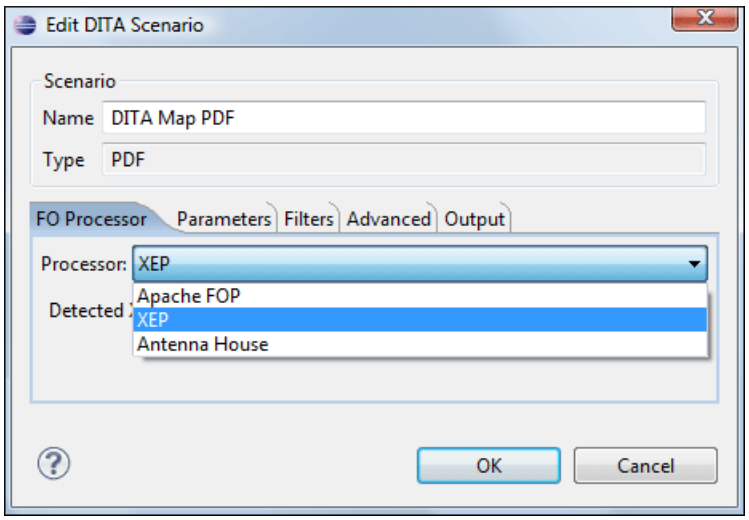
FO Processor Tab (DITA-OT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 279. FO Processor Configuration Tab



You can choose one of the following processors:

Apache FOP

The default processor that comes bundled with Oxygen XML Developer Eclipse plugin.

XEP

The RenderX XEP processor. If XEP is already installed, Oxygen XML Developer Eclipse 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 *(on page 115)*.
- 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 Eclipse plugin installation directory.

Antenna House

The Antenna House (AH Formatter) processor. If Antenna House is already installed, Oxygen XML Developer Eclipse 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 Eclipse 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 115)*

XSL-FO (Apache FOP) Processor for Generating PDF Output *(on page 746)*

Parameters Tab (DITA-OT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 148)* 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 *(on page 149)* 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** *(on page 148)* 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 available only for new parameters that have been added to the list.

Parameters Contributed by an Oxygen Publishing Template

Transformation parameters that are defined in an *Oxygen Publishing Template (on page 962)* descriptor file are displayed in italics. After creating a publishing template *(on page 968)* and adding it to the templates gallery *(on page 850)*, when you select the template in the **Templates** tab, the **Parameters** tab will automatically be updated to include the parameters defined in the template descriptor file.

Related Information:

DITA Open Toolkit Documentation

Feedback Tab (DITA-OT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, a configuration dialog box allows you to customize the transformation with various options in several tabs.

The **Feedback** tab is for those who want to provide a way for users to offer feedback and ask questions in the published output and it is available for the **DITA Map WebHelp Responsive** transformation type. To add a comments component in the output, you need to use **Oxygen Feedback** to create a site configuration for the website where your WebHelp output is published and use this **Feedback** tab to instruct the transformation to install the comments component at the bottom of each WebHelp page.

When you create a site configuration in the **Oxygen Feedback** administration interface, an HTML fragment is generated during the final step of the creation process. You need to click the **Edit** button at the bottom-right of this tab to open a dialog box where you will paste the generated HTML fragment. The HTML fragment can also be set in an **Oxygen Publishing Template** *(on page 962)*, either as an HTML fragment extension point *(on page 825)* or as a transformation parameter *(on page 823)* (the `webhelp.fragment.feedback` parameter). If the fragment is specified in multiple places, the order of precedence (from highest to lowest) is:

- The fragment specified directly in the **Feedback** tab.
- The fragment specified in a publishing template as an HTML fragment extension point.
- The fragment specified in a publishing template as a transformation parameter.

Filters Tab (DITA Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 (on page 1324)*, 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** *(on page 148)* button, or the browsing actions in the **Browse** drop-down list. You can find out more about constructing a *DITAVAL* file in the DITA Documentation.

Note: If a filter file is specified in the `args.filter` parameter (in the **Parameters** tab), the filters are combined (neither file takes precedence over the other).

Exclude from output all elements with any of the following attributes

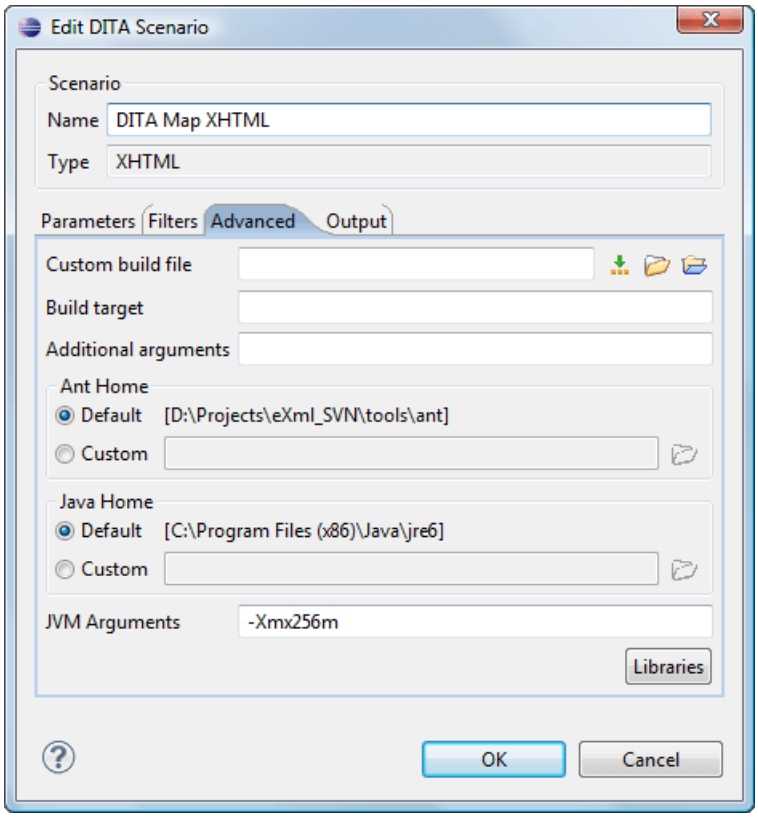
By using the **New**, **Edit**, or **Delete** buttons at the bottom of the pane, you can configure a list of attributes (name and value) to exclude all elements that contain any of these attributes from the output.

Advanced Tab (DITA-OT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 280. 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** *(on page 148)* 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 Eclipse plugin.

Note: It may be possible that the used Java version is incompatible with the DITA Open Toolkit engine. For example, 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 Eclipse 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 (on page 1326)* files or additional class paths) to be used by the transformer.

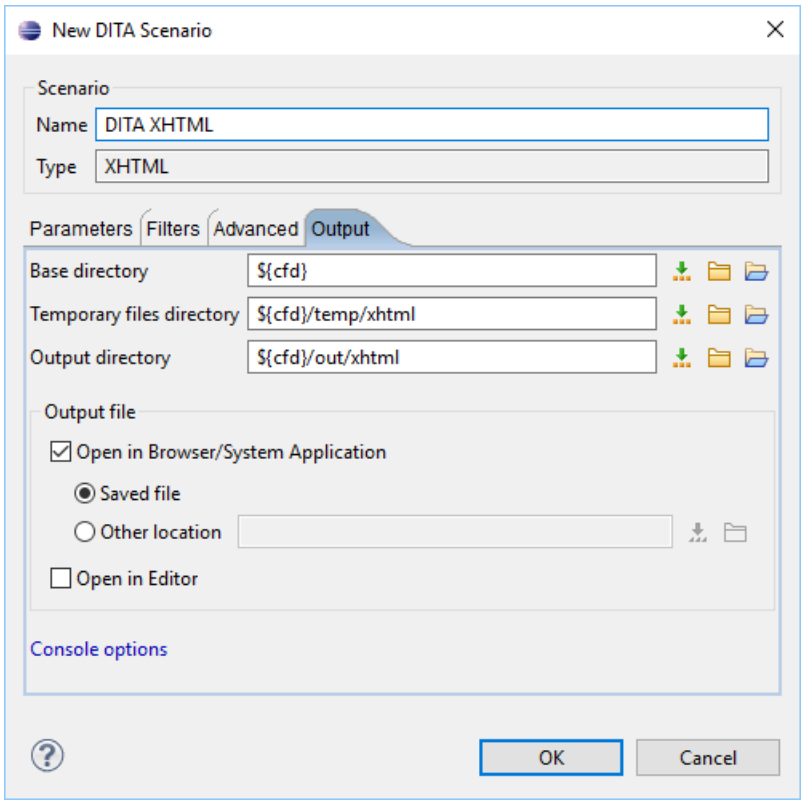
Tip: You can specify the path to the additional libraries using wildcards (for example, `${oxygenHome}/lib/*.jar`).

Output Tab (DITA-OT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 281. 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** *(on page 148)* 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** *(on page 148)* 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** *(on page 148)* button, or the **Browse** button.

Note: If the *DITA map (on page 1324)* or topic is opened from a remote location or a ZIP file, the parameters must specify absolute paths.

Open in Browser/System Application

If selected, Oxygen XML Developer Eclipse 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** *(on page 148)* button, or the **Browse** button.

Open in editor

When this is option is selected, 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).

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 Eclipse plugin runs an Ant transformation scenario as an external process that executes the Ant build script with the built-in Ant distribution (*Apache Ant (on page 1323)* version 1.9.8) that is included with the application, or optionally with a custom Ant distribution configured in the scenario.

Tip: Certain Ant tasks require additional JAR libraries (for example, Ant *mail* tasks). The additional libraries can be added by editing the Ant transformation scenario, and in the **Output** tab, click the **Libraries** button *(on page 730)* in the bottom right corner. This opens a dialog box where you can add JAR libraries. For a list of library dependencies, see https://ant.apache.org/manual/install.html#librarydependencies.

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**.
- Go to **Window > Show View** and select **Transformation Scenarios** to display this view *(on page 770)*. Click the **New Scenario** drop-down menu button and select **ANT transformation**.

Both methods open the transformation configuration 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 *(on page 691)* or edit an existing one *(on page 762)*, 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** *(on page 148)* 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** *(on page 148)* 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 Eclipse 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 Eclipse 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 (on page 1326)* files or additional class paths) to be used by the transformer.

Tip: You can specify the path to the additional libraries using wildcards (for example, `${oxygenHome}/lib/*.jar`).

Parameters Tab (Ant Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 148)* 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 *(on page 149)* 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** *(on page 148)* 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 available only for new parameters that have been added to the list.

Output Tab (Ant Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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** *(on page 148)* 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 in the message panel at the bottom of the editor. The following options are available:

- **When build fails** - Displays the console output log only 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**.
- Go to **Window > Show View** and select **Transformation Scenarios** to display this view *(on page 770)*. Click the **New Scenario** drop-down menu button and select **XSLT transformation**.

Both 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 *(on page 691)* or edit an existing one *(on page 762)*, 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** *(on page 148)* button, or the browsing actions 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 *(on page 143)*, 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 *(on page 693)* 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 *(on page 706)*. 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** *(on page 148)* button, or the browsing actions 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 selected, the scenario applies the stylesheet specified explicitly in the XML document with the `xml-stylesheet` processing instruction. By default, this option is deselected 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 Eclipse plugin for performing a transformation. These include the built-in engines and the external engines defined in the Custom Engines preferences page *(on page 130)*. 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 *(on page 695)* for the current transformation scenario. To configure the same options globally, go to the Saxon-HE/PE/EE preferences page *(on page 140)*. For the current transformation scenario, these **Advanced options** override the options configured in that preferences page.

Parameters

Opens a **Configure parameters** dialog box *(on page 693)* 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 *(on page 693)*. If the XSLT transformation engine is custom-defined, you cannot 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 *(on page 695)* that define extension Java functions or extension XSLT elements used in the transformation.

Additional XSLT stylesheets

Opens a dialog box for adding XSLT stylesheets *(on page 695)* 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 *(on page 695)*, 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 selected, the parameter value is evaluated as an XPath expression before starting the XSLT transformation.

Example:

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 *(on page 148)* (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 *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is selected, 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 *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is selected, 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 available 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 148)*

XSLT Extensions

The **Extensions** button opens a dialog box that allows you to specify the *JARS (on page 1326)* 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.

Tip: You can specify the path to the resources using wildcards (for example, `${oxygenHome}/lib/*.jar`).

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 click 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 *(on page 148)* 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 *(on page 140)* 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.

Saxon-HE/PE/EE Options

The advanced options for Saxon 9.9.1.5 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. If this value is also specified in a configuration file *(on page 696)*, the value in this option takes precedence.

Template ("-it")

A Saxon-specific option that sets the name of the initial XSLT template to be executed. If this value is also specified in a configuration file *(on page 696)*, the value in this option takes precedence.

Tip: If your stylesheet includes `<xsl:template name="xsl:initial-template">`, Oxygen XML Developer Eclipse plugin will automatically detect and use it as the initial template, so this option is not needed in this case.

Use a configuration file ("-config")

Select this option if you want to use a Saxon 9.9.1.5 configuration file that will be executed for the XSLT transformation and validation processes. You can specify the path to the configuration file by entering it in the **URL** field, or by using the **Insert Editor Variables** button, or using the browsing actions in the **Browse** drop-down list.

Debugger trace into XPath expressions (applies to debugging sessions)

Instructs the XSLT Debugger *(on page 1227)* to *step into* XPath expressions.

Enable Optimizations ("-opt")

This option is selected by default, which means that optimization is enabled. If not selected, the optimization is suppressed, which is helpful when reducing the compiling time is important,

optimization conflicts with debugging, or optimization causes extension functions with side-effects to behave unpredictably.

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")

Specifies 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")

Specifies 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.

Saxon-PE/EE Options

The following advanced options are specific for Saxon 9.9.1.5 Professional Edition (PE) and Enterprise Edition (EE) only:

Register Saxon-JS extension functions and instructions

Registers the Saxon-CE extension functions and instructions when compiling a stylesheet using the Saxon 9.9.1.5 processors.

Note: Saxon-CE, being JavaScript-based, was designed to run inside a web browser. This means that you will use Oxygen XML Developer Eclipse 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 selected, 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.

Enable assertions ("-ea")

In XSLT 3.0, you can use the `<xsl:assert>` element to make assertions in the form of XPath expressions, causing a dynamic error if the assertion turns out to be false. If this option is selected, XSLT 3.0 `<xsl:assert>` instructions are enabled. If it is not selected (default), the assertions are ignored.

Saxon-EE Options

The advanced options that are specific for Saxon 9.9.1.5 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 *(on page 36)* and go to **XML > XML Parser > XML Schema** and use the **Default XML Schema version** option *(on page 126)*.

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 allows for parsing the source documents with strict schema-validation enabled.
- **Lax schema validation ("lax")** - If an XML Schema is provided, this mode allows for 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. Selecting 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.

Enable streaming mode

Selecting this option will allow an XSLT to run in streaming mode. It is not selected by default.

Other Options

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.

Using Saxon Integrated Extension Functions

Saxon, the transformation and validation engine used by Oxygen XML Developer Eclipse plugin, can be customized by adding custom functions (called Integrated Extension Functions) that can be called from XPath.

To define such a function, follow these steps:

1. Create a file with a Java class that extends `net.sf.saxon.lib.ExtensionFunctionDefinition`. Here is an example:

```
private static class ShiftLeft extends ExtensionFunctionDefinition {
  @Override
  public StructuredQName getFunctionQName() {
    return new StructuredQName("eg", "http://example.com/saxon-extension", "shift-left");
  }
  @Override
  public SequenceType[] getArgumentTypes() {
   return new SequenceType[] {SequenceType.SINGLE_INTEGER, SequenceType.SINGLE_INTEGER};
  }
  @Override
  public SequenceType getResultType(SequenceType[] suppliedArgumentTypes) {
   return SequenceType.SINGLE_INTEGER;
  }
  @Override
```

```
  public ExtensionFunctionCall makeCallExpression() {
   return new ExtensionFunctionCall() {
   public SequenceIterator call(SequenceIterator[] arguments, XPathContext context)
                    throws XPathException {
       long v0 = ((IntegerValue)arguments[0].next()).longValue();
    long v1 = ((IntegerValue)arguments[1].next()).longValue();
    long result = v0<<v1;
        return Value.asIterator(Int64Value.makeIntegerValue(result));
      }
    };
  }
}
```

2. Compile the class and add it to a JAR file.
3. Add a file called **net.sf.saxon.lib.ExtensionFunctionDefinition** that contains the fully qualified name of the Java class in the `META-INF/services/` folder of the JAR file.

   Note: To add more function definitions in the same JAR file, you need to add their fully qualified names on different lines.

To enable Oxygen XML Developer Eclipse plugin to pick up your custom function definition, the JAR file should be added to the classpath of the transformer. Here are some possibilities:

- If you develop a framework, you just need to link the JAR file in the **Classpath** tab *(on page 55)*.
- In a validation scenario *(on page 288)*, you can use the **Extensions** button to open a dialog box where you can add libraries.
- In a transformation scenario, you can use the **Extensions** button in the **XSLT** tab *(on page 693)* to open a dialog box where you can add libraries.

FO Processor Tab (XSLT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 115)*.

Output Tab (XSLT Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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** *(on page 148)* button, or the **Browse** button.

Open in Browser/System Application

If selected, Oxygen XML Developer Eclipse 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** *(on page 148)* button, or the **Browse** button.

Open in editor

When this is option is selected, 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 Eclipse plugin displays the transformation result in an XML viewer panel at the bottom of the application window with syntax highlighting *(on page 101)*.
- **XHTML** - This option is only available if **Open in Browser/System Application** is not selected. If selected, Oxygen XML Developer Eclipse 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 selecting this option 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** *(on page 148)* button, or the **Browse** button.

Attention: If your input XSLT contains `<xsl:result-document>` elements, then the secondary results will be saved to the specified URIs while the principal result is specified in this **Output** tab. For more information, see: https://www.w3.org/TR/xslt-30/#element-result-document.

Configuring an XSLT Processor for Generating Output

This section explains how to configure an XSLT processor and extensions for such a processor in Oxygen XML Developer Eclipse plugin.

Supported XSLT Processors

Oxygen XML Developer Eclipse plugin includes the following XSLT processors:

- **Xalan 2.7.2** - 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 9.9.1.5 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 9.9.1.5 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 Eclipse 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 XML Schema 1.0 or 1.1. This can be configured in Preferences *(on page 126)*.

  Note: Oxygen XML Developer Eclipse 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 Eclipse 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 9.9.1.5 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 Eclipse plugin.
- **Saxon-CE (Client Edition)** is Saxonica's implementation of XSLT 2.0 for use on web browsers. Oxygen XML Developer Eclipse 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 Eclipse 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.

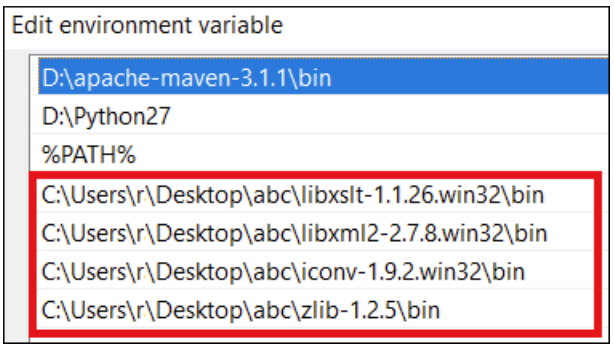
  Note: A specific template, named **Saxon-CE stylesheet**, is available in the **New from Templates** wizard *(on page 177)*.
- **Xsltproc (libxslt)** - Libxslt is the XSLT C library developed for the Gnome project. **Libxslt** is based on *libxml2*, the XML C library developed for the Gnome project. It also implements most of the EXSLT set of processor-portable extensions, functions, and some of Saxon's evaluate and expression extensions.

  Oxygen XML Developer Eclipse plugin uses **Libxslt** through its command-line tool (*Xsltproc*). Depending on your operating system, you must download the Libxslt libraries on your machine from http://

xmlsoft.org/XSLT/downloads.html and place them in a local folder. Then you need to update the `PATH` environmental variable to contain the parent folder where the `xsltproc` executable is located.

Tip: As an example, a Windows installation of the Xsltproc engine would follow these steps:

1. Go to http://ftp.zlatkovic.com/libxml.en.html and download the following ZIP files: **iconv-1.9.2.win32.zip**, **libxml2-2.7.8.win32.zip**, **libxslt-1.1.26.win32.zip**, **zlib-1.2.5.win32.zip**.
2. Unzip all of them into the same folder of your choice.
3. Edit the `PATH` environment variable and add the `bin` folder for all four archives:



4. Restart Oxygen XML Developer Eclipse plugin.

Result: You can now use the **xsltproc** processor as an XSLT engine in the XSLT transformation scenario.

Note: The Xsltproc processor can be configured from the **XSLTPROC** options page *(on page 143)*.

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 (on page 1329)* files of the built-in document types (DocBook, TEI, DITA, etc.) are not handled properly by LIBXML if Oxygen XML Developer Eclipse 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 *(on page 691)* and validation of XSLT stylesheets *(on page 366)*.

  Oxygen XML Developer Eclipse 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 *(on page 691)* and validation of XSLT stylesheets *(on page 366)*.

  Oxygen XML Developer Eclipse plugin performs XSLT transformations and validations using the .NET *Framework* XSLT implementation (*System.Xml.Xsl.XslTransform* class) through the **nxslt** command-line utility. The **nxslt** version included in Oxygen XML Developer Eclipse 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 Eclipse 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 *(on page 36)* 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 *(on page 130)*.
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 Eclipse plugin linked message *(on page 285)*, 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 130)*

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 9.9.1.5** - 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 *(on page 693)* in the **Edit scenario** dialog box.

XSL-FO (Apache FOP) Processor for Generating PDF Output

The Oxygen XML Developer Eclipse plugin installation package is distributed with the Apache FOP that is a Formatting Objects processor for transforming your XML documents to PDF. *FOP* is a print and output independent formatter driven by XSL Formatting Objects. *FOP* is implemented as a Java application that reads a formatting object tree and renders the resulting pages to a specified output.

To see the version of the built-in XSL-FO processor for your installation, go to **Help > About > Libraries** and search for *Apache FOP*.

Other FO processors can be configured in the **FO Processors** preferences page *(on page 115)*.

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 *(on page 747)*.

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 *(on page 36)*, 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 built-in scenario called **DocBook PDF**, edit the XSLT parameters *(on page 691)* and set the font name (for example, **Arial Unicode MS**) to the `body.font.family` and `title.font.family` parameters.
   - For TEI documents you can start with the built-in scenario called **TEI PDF**, edit the XSLT parameters *(on page 691)* and set the font name (for example, **Arial Unicode MS**) to the `bodyFont` and `sansFont` parameters.
   - 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 `<physical-font>` element that has the `char-set="default"` attribute must contain the name of the font (for example, **Arial Unicode MS**)
     - *DITA-OT-DIR*`/plugins/org.dita.pdf2.fop/fop/conf/fop.xconf` - An `<auto-detect>` element must be inserted in the `<fonts>` element, which is inside the `<renderer>` element that has the `mime="application/pdf"` attribute:

       ```
       <renderer mime="application/pdf">
         . . .
         <fonts>
             <auto-detect/>
         </fonts>
         . . .
       </renderer>
       ```

Add a Font to the Built-in FO Processor - Advanced Version

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: On Windows, 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, it is enough to copy it in the `Fonts` directory. If a 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 *(on page 746)*.

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.
2. Register the font in the FOP configuration.

   Note: DITA PDF transformations have their own `fop.xconf` (*`DITA-OT-DIR`*`/plugins/org.dita.pdf2.fop/fop/conf/fop.xconf`). If the font is not installed in the system, it needs to be referenced in the `fop.xconf`.

   a. For information about registering the font in the FOP Configuration, see: https://xmlgraphics.apache.org/fop/2.3/fonts.html.
   b. Open the **Preferences** dialog box *(on page 36)*, go to **XML > XSLT/FO/XQuery > FO Processors**, and enter the path of the FOP configuration file in the **Configuration file** text field.
3. 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.

   DocBook Example: For DocBook documents, you can start with the built-in scenario called **DocBook PDF**, edit the XSLT parameters *(on page 691)*, and set the font name (for example, *Arialuni*) to the `body.font.family` and `title.font.family` parameters.

   TEI Example: For TEI documents, you can start with the built-in scenario called **TEI PDF**, edit the XSLT parameters *(on page 691)*, and set the font name (for example, *Arialuni*) to the `bodyFont` and `sansFont` parameters.

   DITA Example: 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 `<physical-font>` element that has the `char-set="default"` attribute must contain the name of the font.
   - *`DITA-OT-DIR`*`/plugins/org.dita.pdf2/fop/conf/fop.xconf` - A `<font>` element must be inserted in the `<fonts>` element, which is inside the `<renderer>` element that has the `mime="application/pdf"` attribute.

   For more information, see: https://xmlgraphics.apache.org/fop/2.1/fonts.html.

Adding Libraries to the Built-in FO Processor (XML with XSLT and FO)

Starting with Oxygen XML Developer Eclipse plugin version 20.0, both hyphenation and PDF image support are enabled by default in the built-in Apache FO processor. For older version of Oxygen XML Developer Eclipse plugin, use the following procedures to enable such support.

Adding Hyphenation Support for XML with XSLT Transformation Scenarios

If you want to add newer hyphenation libraries or you are using an older version of Oxygen XML Developer Eclipse plugin, follow this procedure:

1. Create a folder called `fop` in the *`[OXYGEN_INSTALL_DIR]`*`/lib` folder.

2. Download the compiled *JAR (on page 1326)* from OFFO.
3. Copy the `fop-hyph.jar` file into the *[OXYGEN_INSTALL_DIR]*`/lib/fop` folder.
4. Restart Oxygen XML Developer Eclipse plugin.

Adding Support for PDF Images

To add support for PDF images in an older version of Oxygen XML Developer Eclipse plugin, 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 Eclipse plugin.

How to Enable Debugging for FO Processor Transformations

If you encounter errors when running PDF transformations that use an FO processor, it is possible to enable debugging/logging to help you identify the problem. To enable debugging/logging for FO processing, follow this procedure:

1. Locate and edit the following configuration file: *[OXYGEN_INSTALL_DIR]*`/tools/config/log4j2.xml`.

   Note: You need write access to this folder, so if you do not have administrator permissions, you may first need to copy the file to another location where you have write access.
2. Edit the `<Root>` element (inside the `<Loggers>` element), change its level to **debug**, and save the file.
3. Restart Oxygen XML Developer Eclipse plugin and re-run the transformation.

   Tip: To make it easier to analyze the data in the logs, it is recommended that you use a small input file when trying to reproduce the problem.
4. Once you are finished with the debugging session, remember to edit the `log4j2.xml` file and change the `<priority>` element back to its original value. Otherwise, performance could be affected.

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**.
- Go to **Window > Show View** and select **Transformation Scenarios** to display this view *(on page 770)*. Click the **New Scenario** drop-down menu button and select **XProc transformation**.

Both 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.

Related Information:

Integrating an External XProc Engine *(on page 753)*

Editing XProc Scripts *(on page 588)*

XProc Tab

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 XProc 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** *(on page 148)* button, or the browsing actions 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 *Add-on for Calabash XProc engine* or a custom engine that is configured in the **XProc Preferences** page *(on page 128)*.

Inputs Tab (XProc Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 148)* and custom editor variables *(on page 155)* 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 *(on page 148)* and custom editor variables *(on page 155)* can be used to specify the URL.

Delete

Removes the selected port from the list. It is available only for new ports that have been added to the list.

Parameters Tab (XProc Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 148)* and custom editor variables *(on page 155)* 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 *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 148)* 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 *(on page 248)*.

Edit

Opens an **Edit** dialog box that allows you to edit an existing output port and its URL. An editor variable *(on page 148)* 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 *(on page 248)*.

Delete

Removes the selected output port from the list. It is available 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

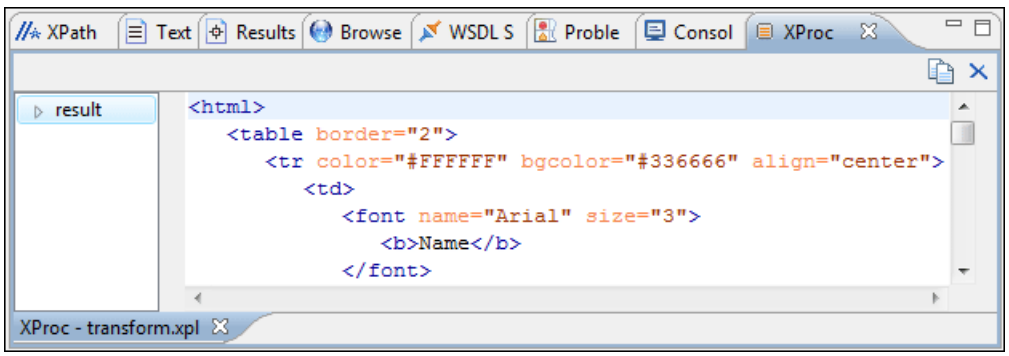
If this option is selected, you can specify a file to be opened at the end of the XProc transformation in the browser or system application that is associated with the file type. You can specify the path by using the text field, its history drop-down, the **Insert Editor Variables** *(on page 148)* button, or the browsing actions 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 282. XProc Transformation Results View



Options Tab (XProc Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 148)* and custom editor variables *(on page 155)* can be used for specifying the URI. This section provides more information about the editor variables that can be used.

Calabash XProc Processor for Generating PDF Output

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 Eclipse plugin.

Integrating an External XProc Engine

Oxygen XML Developer Eclipse plugin includes a bundled version of the *Calabash* XProc engine that can be used for XProc transformations and validation, but you can also integrate other external XProc engines. When you edit an XProc transformation scenario, there is a **Processor** drop-down menu where you can select the XProc engine to be used for the transformation.

If you do not need the external XProc engine to be used for automatic validation or pass parameters/ports and it is not Java-based, you can simply add the external engine by using the **XProc** preferences page *(on page 128)*. Otherwise, if the external engine is Java-based, or it has validation support, or it can receive parameters or ports passed from the transformation, you need to integrate it using the plugin extension procedure below.

For example, there is a public project on GitHub that is an implementation for integrating Morgana XProc with Oxygen XML Developer Eclipse plugin: https://github.com/xml-project/support-for-xmleditor. Also, 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) (on page 1326)* from the classes you created.
4. Create an `engine.xml` file according to the `engine.dtd` file. The attributes of the `<engine>` element are as follows:
    a. **name** - The name of the XProc engine.
    b. **description** - A short description of the XProc engine.
    c. **class** - The complete name of the class that implements *ro.sync.xml.transformer.xproc.api.XProcTransformerInterface*.
    d. **version** - The version of the integration.
    e. **engineVersion** - The version of the integrated engine.
    f. **vendor** - The name of the vendor / implementer.
    g. **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 new folder (for example, named `MyXprocEngine`) and place the `engine.xml` and all the libraries necessary to run the new integration in that folder.
6. Place that new folder (e.g. `MyXprocEngine`) inside a new plugin folder. This new plugin folder should also contain a `plugin.xml` file that points to the new engine folder (e.g. `MyXprocEngine`).
   The `plugin.xml` file would look like this:

```
<plugin
 id="morgana.xproc.addon"
 name="Contribute Morgana XProc"
 description="Contribute Morgana XProc"
 version="1.0"
 vendor="Syncro Soft"
 class="ro.sync.exml.plugin.Plugin"
 classLoaderType="preferReferencedResources">
 <extension type="AdditionalXProcEngine" path="MyXprocEngine/"></extension>
</plugin>
```

Related Information:

Editing XProc Scripts *(on page 588)*

Creating an XProc Transformation Scenario *(on page 749)*

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**.
- Go to **Window > Show View** and select **Transformation Scenarios** to display this view *(on page 770)*. Click the **New Scenario** drop-down menu button and select **XQuery transformation**.

Both 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 *(on page 691)* or edit an existing one *(on page 762)*, 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** *(on page 148)* button, or the browsing actions 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 *(on page 143)*, 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** *(on page 148)* button, or the browsing actions 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 Eclipse plugin for performing a transformation. These include the built-in engines and the external engines defined in the Custom Engines preferences page *(on page 130)*. 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 *(on page 710)* for the current transformation scenario. To configure the same options globally, go to the Saxon-HE/PE/EE preferences page *(on page 140)*. For the current transformation scenario, these **Advanced options** override the options configured in that preferences page.

Parameters

Opens the **Configure parameters** dialog box *(on page 708)* 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 *(on page 710)* 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 selected, the parameter value is evaluated as an XPath expression before starting the XQuery transformation.

Example:

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 *(on page 148)* (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 *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is selected, 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 *(on page 148)* can be inserted in the text box using the **Insert Editor Variables** button. If the **Evaluate as XPath** option is selected, 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 available 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 148)*

XQuery Extensions

The **Extensions** button is used to specify the *JAR (on page 1326)* 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 click 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 *(on page 134)* 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.

Saxon-HE/PE/EE Options

The advanced options for Saxon 9.9.1.5 Home Edition (HE), Professional Edition (PE), and Enterprise Edition (EE) are as follows:

Use a configuration file ("-config")

Sets a Saxon 9.9.1.5 configuration file that is used for XQuery transformation and validation scenarios.

Enable Optimizations ("-opt")

This option is selected by default, which means that optimization is enabled. If not selected, the optimization is suppressed, which is helpful 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.

Recoverable errors ("-warnings")

Specifies 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")

Specifies 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.

Saxon-PE/EE Options

The following advanced options are specific for Saxon 9.9.1.5 Professional Edition (PE) and Enterprise Edition (EE) only:

Allow calls on extension functions ("-ext")

If selected, calls on external functions are allowed. Selecting 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.

Saxon-EE Options

The advanced options that are specific for Saxon 9.9.1.5 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 allows for parsing the source documents with strict schema-validation enabled.
- **Lax schema validation ("lax")** - If an XML Schema is provided, this mode allows for 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. Selecting 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.

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 selected, backup versions for any XML files updated with an XQuery Update are generated. This option is available when the **Enable XQuery update** option is selected.

Other Options

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.

FO Processor Tab (XQuery Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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 *(on page 115)*.

Output Tab (XQuery Transformations)

When you create a new transformation scenario *(on page 691)* or edit an existing one *(on page 762)*, 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

Selecting 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** *(on page 148)* button, or the **Browse** button.

Open in Browser/System Application

If selected, Oxygen XML Developer Eclipse 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** *(on page 148)* button, or the **Browse** button.

Open in editor

When this is option is selected, 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 Eclipse plugin displays the transformation result in an XML viewer panel at the bottom of the application window with syntax highlighting *(on page 101)*.
- **XHTML** - This option is only available if **Open in Browser/System Application** is not selected. If selected, Oxygen XML Developer Eclipse 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 selecting this option 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** *(on page 148)* 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**.
- Go to **Window > Show View** and select **Transformation Scenarios** to display this view *(on page 770)*. Click the **New Scenario** drop-down menu button and select **SQL transformation**.

Both 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** *(on page 148)* button, or the browsing actions 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 *(on page 40)*.

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: Since transformation scenarios that are associated with built-in *frameworks (on page 1325)* are read-only, to edit one of these scenarios you will need to duplicate it and edit the duplicated scenario *(on page 764)*.

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.

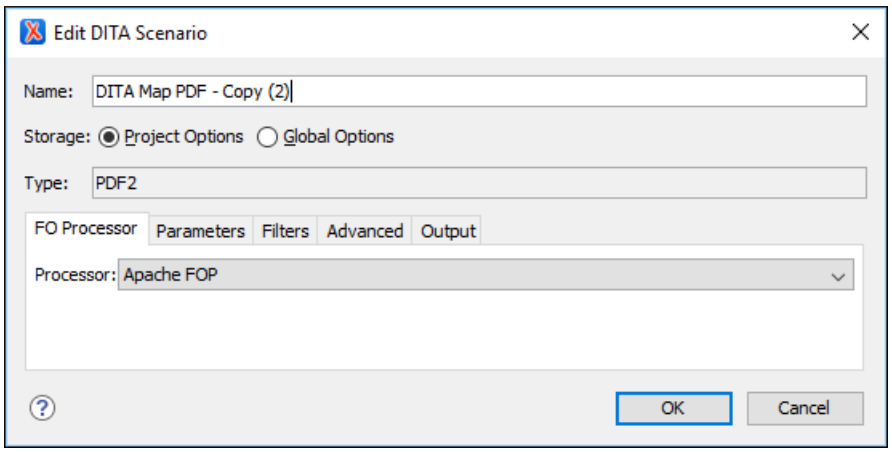
Step Result: The **Configure Transformation Scenario(s)** dialog box *(on page 765)* is opened.

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 *(on page 770)* 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.

Figure 283. Edit Scenarios Configuration Dialog Box



Transformation Types

The **Configure Transformation Scenario(s)** dialog box *(on page 765)* contains a **Type** column that shows you the transformation type for each of the listed scenarios. Each type of transformation contains 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:
  - Templates Tab
  - 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 *(on page 730)*
  - Parameters Tab *(on page 731)*
  - Output Tab *(on page 731)*
- **XSLT** - This type of transformation includes configurable options in the following tabs:
  - XSLT Tab *(on page 692)*
  - FO Processor Tab *(on page 700)*

    - Output Tab *(on page 700)*
- **XProc** - This type of transformation includes configurable options in the following tabs:
    - XProc Tab *(on page 750)*
    - Inputs Tab *(on page 750)*
    - Parameters Tab *(on page 751)*
    - Outputs Tab *(on page 751)*
    - Options Tab *(on page 752)*
- **XQuery** - This type of transformation includes configurable options in the following tabs:
    - XQuery Tab *(on page 707)*
    - FO Processor Tab *(on page 712)*
    - Output Tab *(on page 712)*

Related Information:

Creating New Transformation Scenarios *(on page 691)*

Duplicating a Transformation Scenario *(on page 764)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

Applying Associated Transformation Scenarios *(on page 764)*

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 built-in 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 *(on page 765)* 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 *(on page 770)* 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 *(on page 763)*.

Related Information:

Creating New Transformation Scenarios *(on page 691)*

Editing a Transformation Scenario *(on page 762)*

Applying Associated Transformation Scenarios

If you have associated transformation scenarios for the current document (in the **Configure Transformation Scenario(s)** dialog box *(on page 765)* or **Transformation Scenarios** view *(on page 770)*), Oxygen XML Developer Eclipse plugin included an **Apply Transformation Scenario(s)** action that allows you to quickly apply the associated transformation scenarios on the current document. Note that if an association is not detected, this action will open the **Configure Transformation Scenario(s)** dialog box *(on page 765)* where you can choose the scenarios you want to apply.

The **Apply Transformation Scenario(s)** action can be initiated from any of the following methods:

- Use the **Alt + Shift + T, T (Command + Alt + T, T on OS X))** keyboard shortcut.
- Click the **Apply Transformation Scenario(s)** button on the main toolbar.
- Click the **Apply Transformation Scenario(s)** button on the toolbar in the **Transformation Scenarios** view *(on page 770)*.
- Right-click a file in the **Navigator** view *(on page 192)* and select **Transform > Apply Transformation Scenario(s)**.
- Use the **Apply Associated** button in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Related Information:

Creating New Transformation Scenarios *(on page 691)*

Editing a Transformation Scenario *(on page 762)*

Configure Transformation Scenario(s) Dialog Box *(on page 765)*

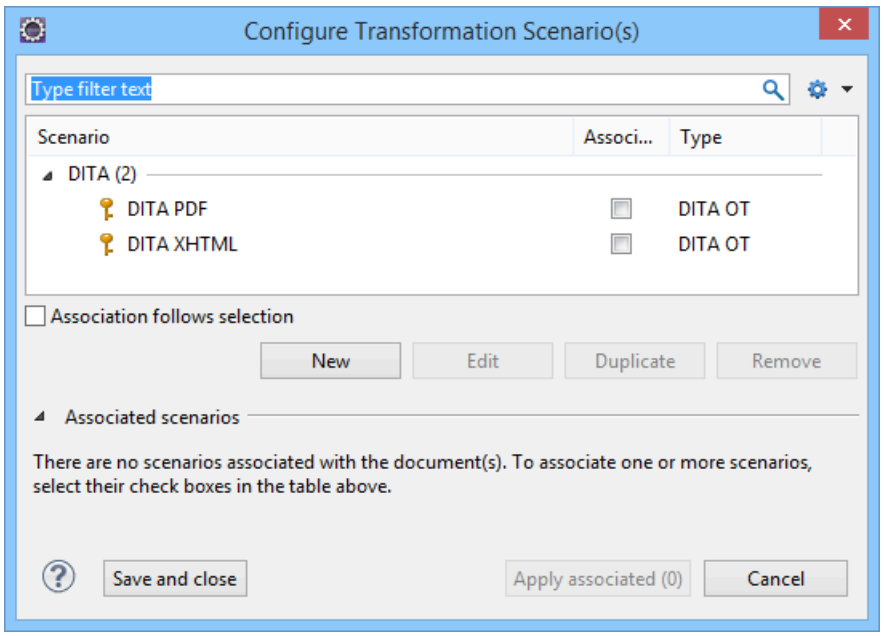
Transformation Scenarios View *(on page 770)*

Configure Transformation Scenario(s) Dialog Box

You can use the **Configure Transformation Scenarios(s)** dialog box for editing existing transformation scenarios *(on page 762)* or creating new ones *(on page 691)*.

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 284. Configure Transformation Scenario(s) Dialog Box



Top Section

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 Eclipse 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 Eclipse 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 Eclipse plugin adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export selected scenarios individually. Oxygen XML Developer Eclipse plugin creates a `scenarios` file that contains the exported scenarios. This is useful if you want to share scenarios with others or export them to another computer.

Middle Section

The middle section of the dialog box displays the scenarios that you can apply to the current document. The following columns are used to display the transformation scenarios:

- **Association** - The checkboxes 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** - If the **Show Type** contextual menu option is selected, this column displays the type of the transformation scenario. For further details about the types of transformation scenarios that are available in Oxygen XML Developer Eclipse plugin, see the Transformation Types section *(on page 763)*.
- **Storage** - If the **Show Storage** contextual menu option is selected, this column displays where a transformation scenario is stored.

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.

Each transformation scenario listed in the middle section of the dialog box also includes the following contextual menu actions:

Edit

This button opens the **Edit Scenario** configuration dialog box *(on page 762)* that allows you to configure the options of the transformations scenario.

Duplicate

Use this button to create a duplicate transformation scenario *(on page 764)*.

Remove

Use this button to remove custom transformation scenarios.

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 Eclipse 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 Eclipse plugin adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export selected scenarios individually. Oxygen XML Developer Eclipse plugin creates a `scenarios` file that contains the exported scenarios. This is useful if you want to share scenarios with others or export them to another computer.

Bottom Section

The bottom section of the dialog box contains the following actions and information:

Association follows selection

Select 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 selected, the **Association** column is hidden.

New

This button allows you to create a new transformation scenario *(on page 691)*.

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 *(on page 763)*.

Note: If you try to edit a transformation scenario associated with a defined document type, Oxygen XML Developer Eclipse plugin displays a warning message to inform you that this is not possible and gives you the option to create a duplicate transformation scenario *(on page 764)* to edit instead.

Duplicate

Use this button to create a duplicate transformation scenario *(on page 764)*.

Remove

Use this button to remove transformation scenarios.

Note: Removing scenarios associated with a defined document type is not allowed.

Associated scenarios section

Displays the scenarios that are associated with the current document. Selecting a checkbox in the **Association** column *(on page 767)* in the list of scenarios will add that scenario to this section. To remove a scenario from being associated with the current document, simply click the remove icon ( ) to the right of the scenario name.

Save and close

Saves the current configuration and closes the dialog box.

Apply associated

Use this button to apply the associated scenarios and run the transformation on the current document.

Cancel

Cancels any changes made in the dialog box and reverts to the previously saved association.

Your selections in the **Configure Transformation Scenarios(s)** dialog box are persistent so the configured associations for the current document will be remembered after the dialog box is closed.

Related Information:

- Editing a Transformation Scenario *(on page 762)*
- Duplicating a Transformation Scenario *(on page 764)*
- Applying Associated Transformation Scenarios *(on page 764)*
- Creating New Transformation Scenarios *(on page 691)*
- Sharing Transformation Scenarios *(on page 770)*

Batch Transformations

A transformation action can be applied on a batch of selected files from the contextual menu of the **Navigator** view *(on page 196)* 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 Eclipse plugin editor variables *(on page 148)* 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 *(on page 148)* is assigned for the input file. For example, for a *DocBook PDF transformation*, make sure the **XML URL** input box is set to the *${currentFileURL}* editor variable *(on page 152)*. For a DITA PDF transformation, make sure the `args.input` parameter is set to the *${cf}* editor variable *(on page 152)*.
    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 *(on page 148)*, 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 148)*

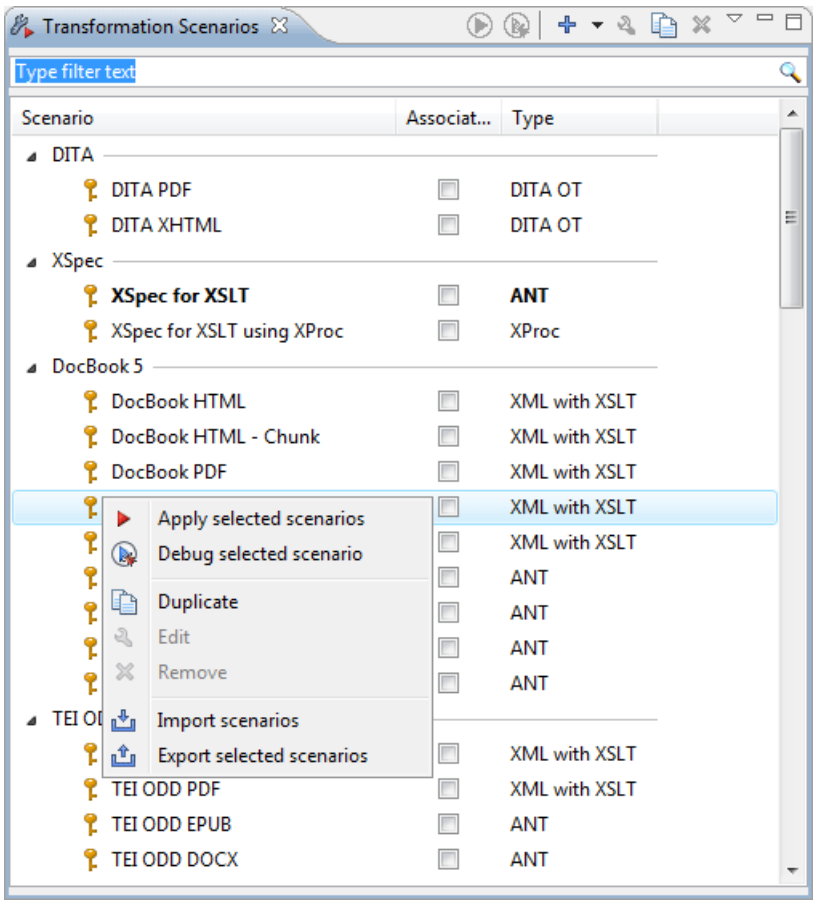
Sharing Transformation Scenarios

The transformation scenarios and their settings can be shared with other users by exporting them to a specialized scenarios file *(on page 147)* 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 285. Transformation Scenarios view



Oxygen XML Developer Eclipse plugin supports multiple scenarios association. To associate multiple scenarios with a document, select the checkboxes in front of each scenario. You can also associate multiple scenarios with a document from the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

By default, Oxygen XML Developer Eclipse plugin presents the items in the following order:

1. Scenarios that match the current *framework (on page 1325)*.
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 (on page 1327)* 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 *(on page 691)*. Oxygen XML Developer Eclipse 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 *(on page 763)*.

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 Eclipse 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 Eclipse 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 Eclipse 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 Eclipse 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 Eclipse 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 Eclipse plugin adds `imported` to the name of the imported scenario.

Export selected scenarios

Use this option to export selected scenarios individually. Oxygen XML Developer Eclipse plugin creates a `scenarios` file that contains the exported scenarios. 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.

Your selections in the **Transformation Scenarios** view are persistent so the configured associations for the current document will be remembered whenever the document is opened.

Related Information:

Editing a Transformation Scenario *(on page 762)*

Creating New Transformation Scenarios *(on page 691)*

WebHelp Output Customization

Oxygen XML WebHelp provides the ability to generate two different types of output, **WebHelp Responsive** and **WebHelp Classic**. Each type has its own set of options and features. The **WebHelp Responsive** variant is available for DITA documents while the **WebHelp Classic** variants are available for DocBook.

Table 31. WebHelp System Feature Matrix

Features	WebHelp System Variants		
	Responsive	Classic w/ Feedback	Classic
Desktop Systems	✔	✔	✔
Mobile Devices	✔		
Built-in Skins	✔	✔	✔
Built-in Templates	✔		
Search Capabilities	✔	✔	✔
Modern Layout	✔		
Adaptable to Any Screen Size	✔		
Comments Section	✔	✔	
DITA Documents	✔		
DocBook Documents		✔	✔
Tri-Pane Frames or Frameless Version		✔	✔

WebHelp Responsive Output for DITA

WebHelp Responsive features a very flexible layout, is designed to adapt to any screen size to provide an optimal viewing and interaction experience. It is based upon the *Bootstrap* responsive front-end framework and is available for DITA document types.

WebHelp Responsive output can be generated by using the **DITA Map WebHelp Responsive** *(on page 672)* transformation scenario.

Layout and Features

This section contains information about the layout and features of the WebHelp Responsive output.

Layout of the Responsive Page Types

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 layout includes a collection of skins that you can choose from, or you can customize your own.

Figure 286. WebHelp Responsive Output on a Normal Screen

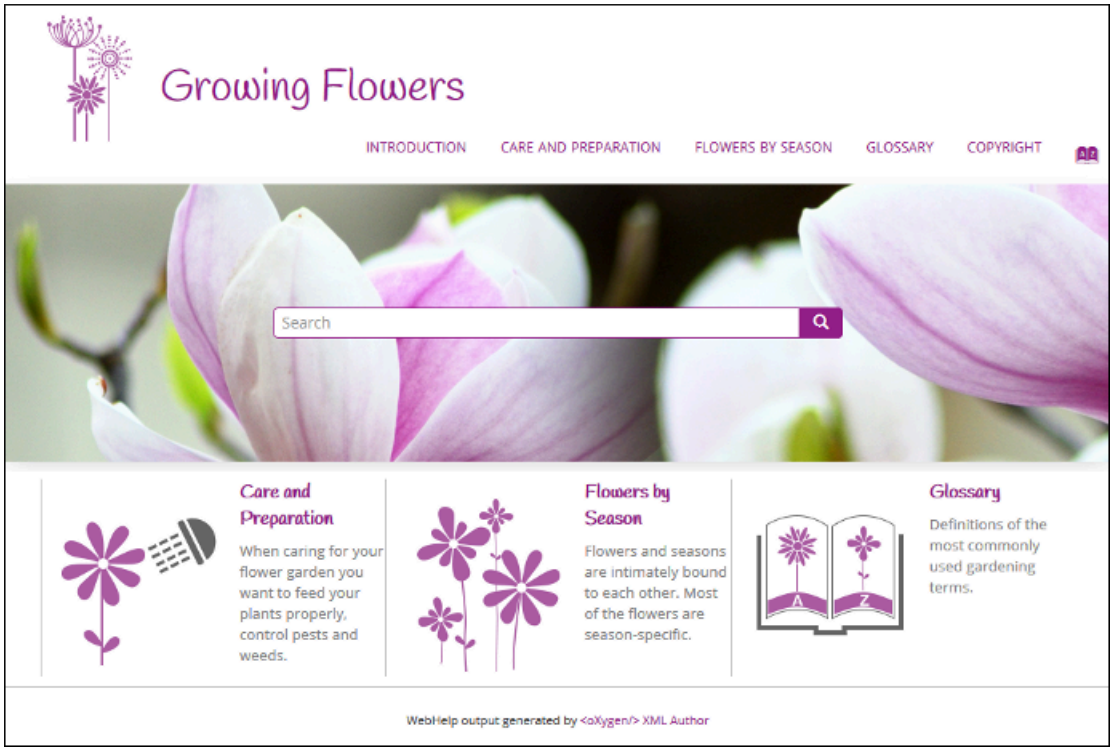


Figure 287. WebHelp Responsive Output on a Narrow Screen

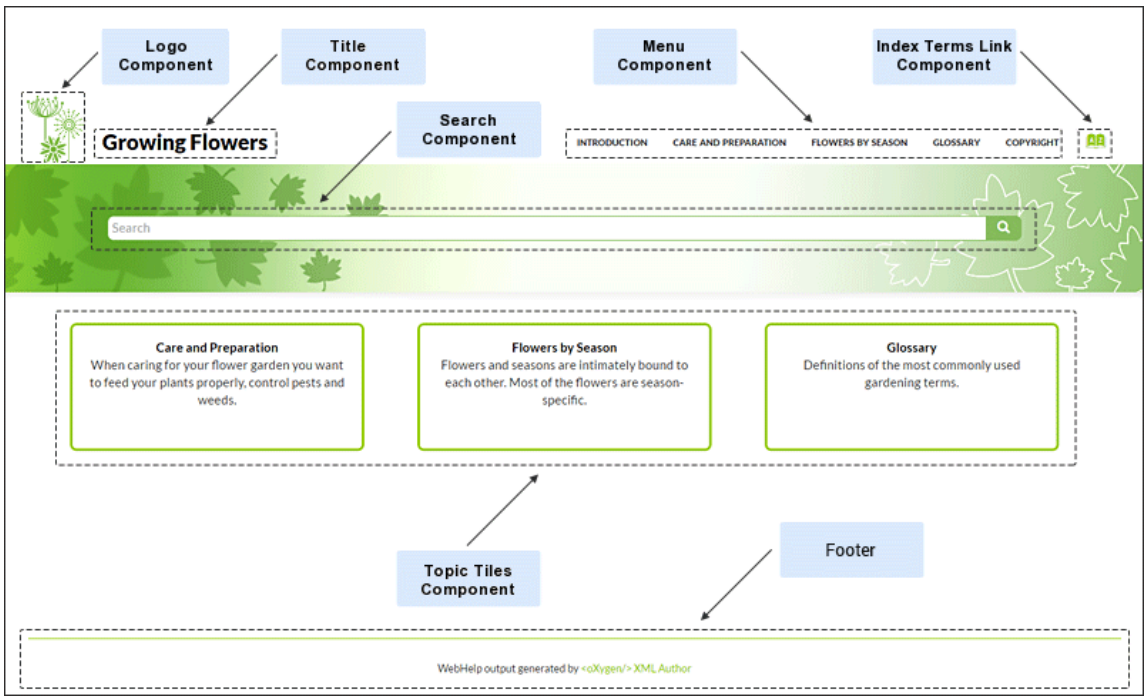
Main Page

The *Main Page* is the home page generated in the WebHelp Responsive output. 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 main page also consists of various other components, such as a logo, title, menu, search field, or index link.

Main Page - Tiles Layout

In the *tiles* presentation mode, a tile component is created for each chapter (first-level topic) in the publication. The tile presents a link to the topic and its short description.

Figure 288. Main Page - Tiles Layout

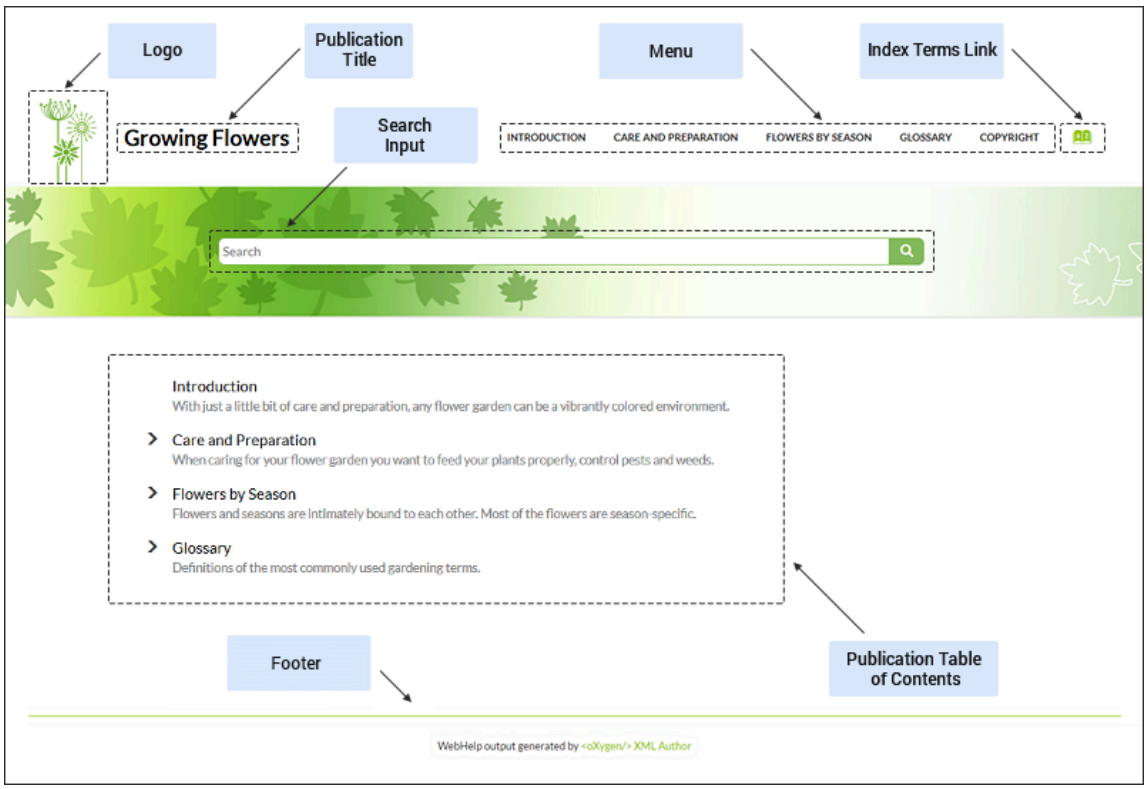


1. Logo Component *(on page 778)*
2. Title Component *(on page 778)*
3. Search Input Component *(on page 779)*
4. Menu Component *(on page 778)*
5. Index Terms Link Component *(on page 779)*
6. Topic Tiles Component *(on page 779)*
7. Footer Component *(on page 779)*

Main Page - Tree Layout

In the *tree* presentation mode, links to the first and second level topics in the publication are displayed using a tree-like component.

Figure 289. Main Page - Tree Layout



1. Logo Component *(on page 778)*
2. Title Component *(on page 778)*
3. Search Input Component *(on page 779)*
4. Menu Component *(on page 778)*
5. Index Terms Link Component *(on page 779)*
6. Table of Contents Component *(on page 779)*
7. Footer Component *(on page 779)*

Main Page Components

The layout components displayed in the main page are:

Publication Title

The title of the publication. It is usually taken from the DITA map title.

Logo

Displays a logo associated with the publication. Additionally, you can set a target URL that will be opened when you click on the logo image.

The logo image can be specified using the webhelp.logo.image transformation parameter *(on page 907)*. For the target URL, use the webhelp.logo.image.target.url parameter *(on page 907)*.

Menu

Helps you to navigate to your documentation. This component presents a set of links to all topics from your publication. For information about customizing the menu, see How to Customize the Menu *(on page 865)* topic.

Index Terms Link

Presents a link to the index terms page. You can control if this component is displayed by using the webhelp.show.indexterms.link parameter *(on page 909)*.

Search Input

An input text field where you can enter search queries.

Topic Tiles

A tile associated with a main topic. Each topic tile has three sections that correspond to the topic title, short description, and image.

Topic Tile Title

Presents the navigation title of the associated topic.

Topic Tile Short Description

Presents the short description of the topic. It may be collected either from the topic or from the DITA map topic meta.

Topic Tile Image

Presents an image associated with the topic. The image association *(on page 865)* is done in the DITA map.

Tree Table of Contents

An area that contains first and second-level topic titles from your publication.

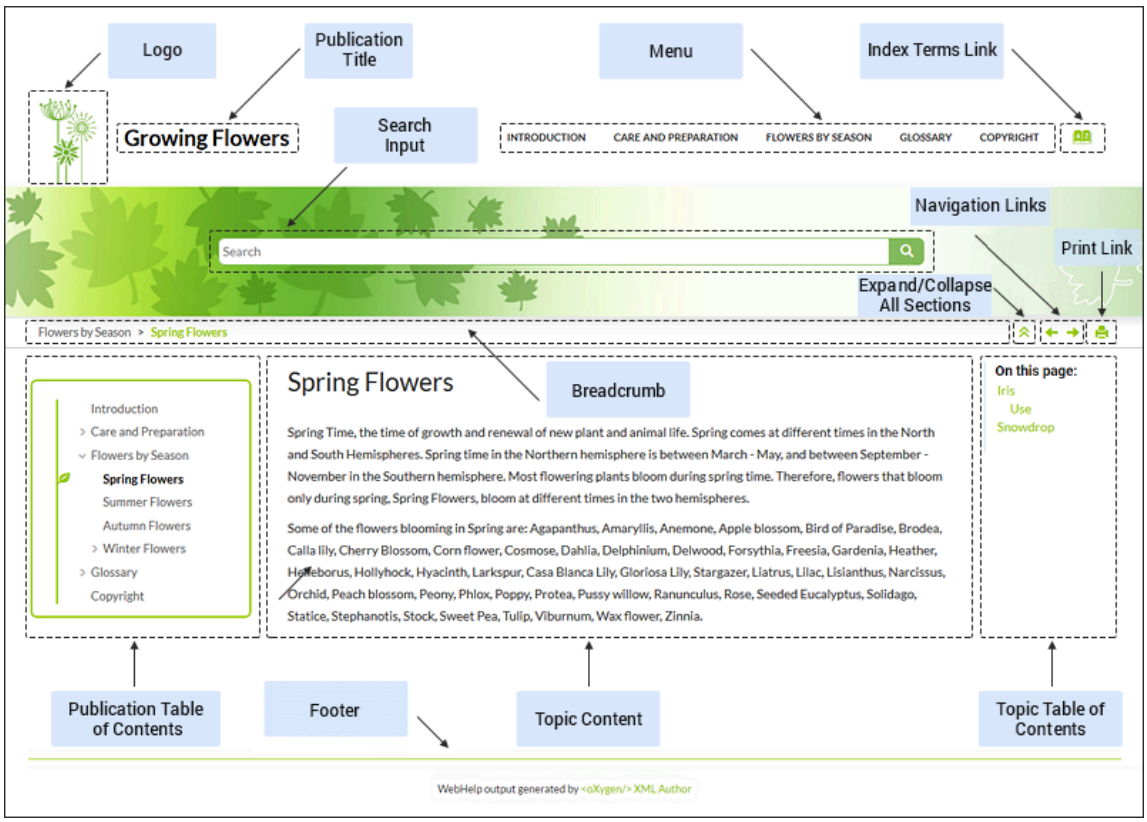
Page Footer

WebHelp Responsive output footer.

Topic Page

The *Topic Page* is the page generated for each DITA topic in the WebHelp Responsive output. 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 290. Topic Page



1. Logo Component *(on page 780)*
2. Title Component *(on page 780)*
3. Search Input Component *(on page 781)*
4. Menu Component *(on page 781)*
5. Index Terms Link Component *(on page 781)*
6. Expand/Collapse All Sections Component *(on page 781)*
7. Navigation Links Component *(on page 781)*
8. Print Link Component *(on page 781)*
9. Breadcrumb Component *(on page 781)*
10. Publication Table of Contents Component *(on page 781)*
11. Footer Component *(on page 782)*
12. Topic Content Component *(on page 781)*
13. Topic Table of Contents Component *(on page 781)*

Topic Page Components

The layout components displayed in this page are:

Publication Title

The title of the publication. It is usually taken from the DITA map title.

Logo

Displays a logo associated with the publication. Additionally, you can set a target URL that will be opened when you click on the logo image.

The logo image can be specified using the webhelp.logo.image transformation parameter *(on page 907)*. For the target URL, use the webhelp.logo.image.target.url parameter *(on page 907)*.

Menu

Helps you to navigate to your documentation. This component presents a set of links to all topics from your publication. For information about customizing the menu, see How to Customize the Menu *(on page 865)* topic.

Index Terms Link

Presents a link to the index terms page. You can control if this component is displayed by using the webhelp.show.indexterms.link parameter *(on page 909)*.

Search Input

An input text field where you can enter search queries.

Navigation Links

The navigation links (⬅ **Previous** / ➡ **Next** arrows) can be used to navigate to the previous or next topic. These navigation links are controlled by the `collection-type` attribute. For example, if you set `collection-type="sequence"` on a parent topic reference, navigation links will be generated in the output for that topic and all of its child topics. You can also use the `webhelp.default.collection.type.sequence` parameter and set its value to `yes` to generate navigation links for all topics, regardless of whether or not the `collection-type` attribute is present.

Tip: To hide the navigation links, you can edit the transformation scenario and set the value of the `webhelp.show.navigation.links` parameter to `no`.

Expand/Collapse Sections Button

Icon that expands or collapses sections listed in the side table of contents within a topic.

Print Link

A print icon that opens the print dialog box for your particular browser.

Breadcrumb

Presents the path of the current displayed DITA topic.

Topic Content

Presents the content of the associated DITA topic.

Publication Table of Contents

A Table of Content for the publication displayed in the left side.

Topic Table of Contents (*On this page* links)

A table of contents for the topic displayed on the right side with a heading named **On this page** and it contains links to each section within the current topic and the section corresponding to

the current scroll position is highlighted. This component is generated for any topic that contains at least two `<section>` elements and each `<section>` must have an `@id` attribute.

Page Footer

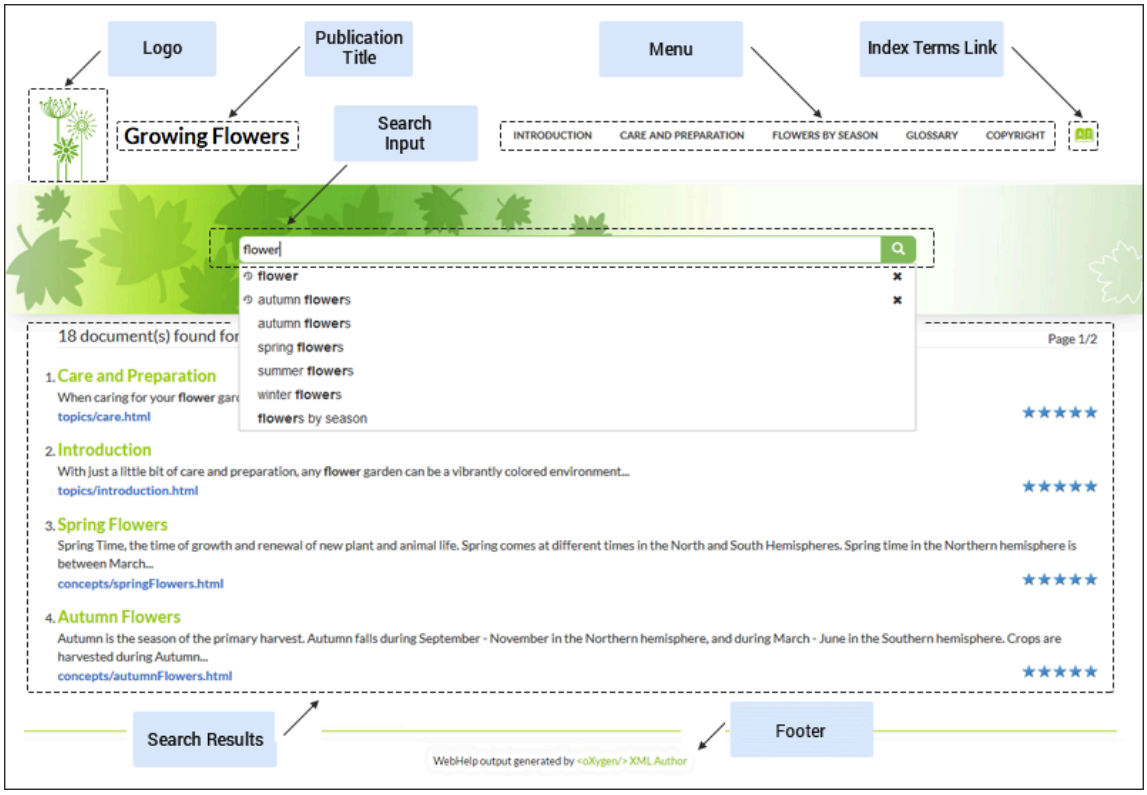
WebHelp Responsive output footer.

Search Page

The *Search Page* presents search results in the WebHelp Responsive output. The HTML page consists of a search results component along with various other additional components, such as a title, menu, or index link.

When you enter search terms in the **Search** field, the results are displayed in a results page. When you click on a result, the topic is opened in the main pane and the search results are highlighted. If you want to remove the colored highlights, click the **Toggle Highlights** button at the top-right side of the page. The **Search** field also includes an *autocomplete* feature.

Figure 291. Search Results Page



1. Logo Component *(on page 783)*
2. Title Component *(on page 783)*
3. Search Input Component *(on page 783)*
4. Menu Component *(on page 783)*
5. Index Terms Link Component *(on page 783)*
6. Search Results Component *(on page 783)*
7. Footer Component *(on page 783)*

Search Page Components

The layout components displayed in the search page are:

Publication Title

The title of the publication. It is usually taken from the DITA map title.

Logo

Displays a logo associated with the publication. Additionally, you can set a target URL that will be opened when you click on the logo image.

The logo image can be specified using the webhelp.logo.image transformation parameter *(on page 907)*. For the target URL, use the webhelp.logo.image.target.url parameter *(on page 907)*.

Menu

Helps you to navigate to your documentation. This component presents a set of links to all topics from your publication. For information about customizing the menu, see How to Customize the Menu *(on page 865)* topic.

Index Terms Link

Presents a link to the index terms page. You can control if this component is displayed by using the webhelp.show.indexterms.link parameter *(on page 909)*.

Search Input

An input text field where you can enter search queries.

Search Results

Each result includes the topic title that can be clicked to open that page. Under the title, a breadcrumb is displayed that shows the path of the topic and you can click any of the topics in the breadcrumb to open that particular page.

Page Footer

WebHelp Responsive output footer.

Autocomplete Suggestions in the Search Text Field

When you are typing in the search text field, proposals are presented to help you to compute the search query. The information proposed when you are typing is collected from:

- The search queries from the history of the previous searches.
- The titles collected from your documentation.
- Documentation index terms and keywords. For example, in a DITA topic, the keywords and index terms are specified in the topic prolog section like this:

```
<prolog>
  <metadata>
 <keywords><indexterm>databases</indexterm></keywords>
 <keyword>installing</keyword>
```

```
 <keyword>uninstalling</keyword>
    <keyword>prerequisites</keyword>
  </metadata>
</prolog>
```

Missing Terms

If you enter multiple search terms (other than *stop words*), for any result that the search engine found at least one term but not one or more of the other terms, the **Missing** terms will be listed below each result.

Related Information:

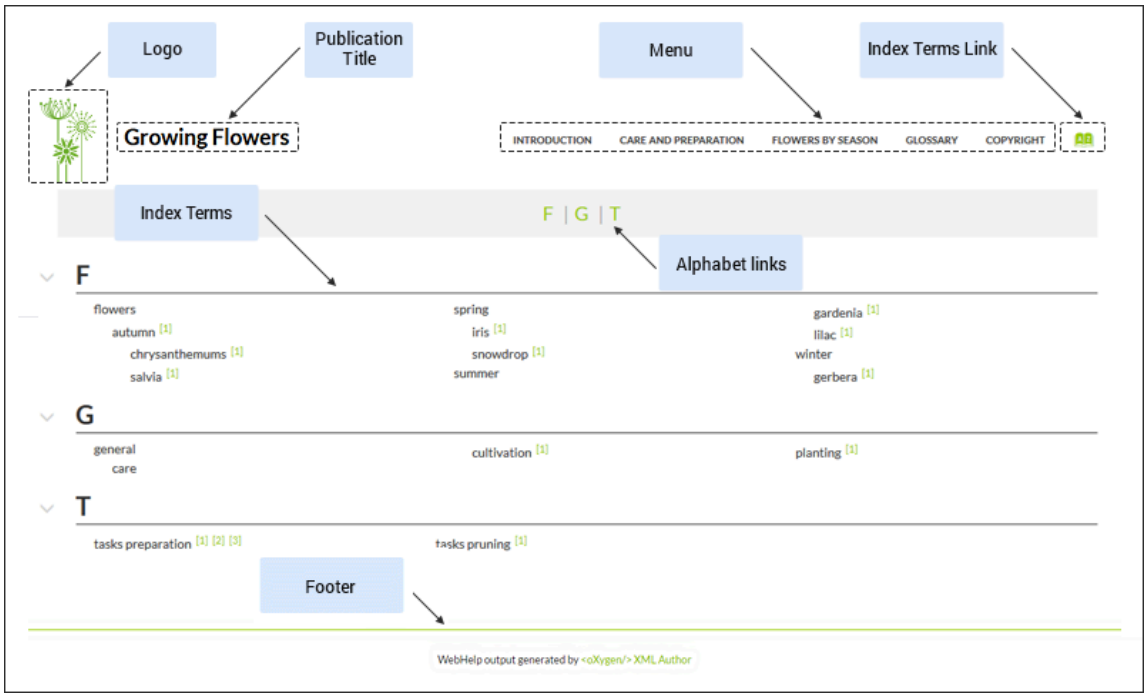
WebHelp Responsive Search Engine *(on page 785)*

Index Terms Page

The *Index Terms Page* page consists of an index terms section along with various other additional components, such as a title, menu, or search field.

An alphabet that contains the first letter of the documentation index terms is generated at the top of the index page. Each letter represents a link to a specific indices section. The indexes are presented in multiple columns to make it easier to read this page.

Figure 292. Index Terms Page



1. Logo Component *(on page 785)*
2. Title Component *(on page 785)*
3. Menu Component *(on page 785)*
4. Index Terms Link Component *(on page 785)*
5. Index Terms Component *(on page 785)*

6. Alphabet Links Component *(on page 785)*
7. Footer Component *(on page 785)*

Index Terms Page Components

The layout components displayed in this page are:

Publication Title

The title of the publication. It is usually taken from the DITA map title.

Logo

Displays a logo associated with the publication. Additionally, you can set a target URL that will be opened when you click on the logo image.

The logo image can be specified using the webhelp.logo.image transformation parameter *(on page 907)*. For the target URL, use the webhelp.logo.image.target.url parameter *(on page 907)*.

Menu

Helps you to navigate to your documentation. This component presents a set of links to all topics from your publication. For information about customizing the menu, see How to Customize the Menu *(on page 865)* topic.

Index Terms Link

Presents a link to the index terms page. You can control if this component is displayed by using the webhelp.show.indexterms.link parameter *(on page 909)*.

Index Terms Alphabet

An alphabet that contains the first letter of index terms. Each letter represents a link to a specific indices section.

Index Terms

The first letter of the index along with the list of index terms.

Page Footer

WebHelp Responsive output footer.

Search Engine

Search engine has two main components:

Search indexer

It is also known as a **spider**. This component is active when you publish your documentation to WebHelp and it is responsible for creating the *search index*. This component traverses all HTML pages (for DITA topics) to gather information.

Search interface

This component is an interface between the user and the *search index*. It helps the user to search through the *search index* and displays results in the search page.

Search Field and Results Page

When you enter search terms in the **Search** field, the results are displayed in a results page. When you click on a result, the topic is opened in the main pane and the search results are highlighted. If you want to remove the colored highlights, click the **Toggle Highlights** button at the top-right side of the page. The **Search** field also includes an *autocomplete* feature.

Each result includes the topic title that can be clicked to open that page. Under the title, a breadcrumb is displayed that shows the path of the topic and you can click any of the topics in the breadcrumb to open that particular page.

If you enter multiple search terms (other than *stop words*), for any result that the search engine found at least one term but not one or more of the other terms, the **Missing** terms will be listed below each result.

5-Star Rating Mechanism and Sorting

The **Search** feature is also enhanced with a rating mechanism that computes scores for every result that matches the search criteria. These scores are then translated into a 5-star rating scheme and the stars are displayed to the right of each result. The search results are sorted depending on the following:

- Search entries that satisfy the phrase search criterion are presented first.
- 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 term is included in a meta keyword.
  - The search term is in the title of the page.
  - The search term is in bold text in a paragraph.
  - The search term is in normal text in a paragraph.

Tag Element Scoring Values

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 How to Change Element Scoring in Search Results *(on page 876)*.

Search Rules

Rules that are applied during a search include:

- You can use quotes to perform an exact search for multiple word phrases (for example, *"grow flowers"* will only return results if both words are found consecutively and exactly as they are typed in the search field). This type of search is known as a *phrase search*.
- *Boolean Search* is 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.
- Words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters count as a single word.
- Your search terms should contain two or more characters (note that stop words will be ignored). This rule does not apply to CJK (Chinese, Japanese, Korean) languages.
- When searching for multi-word phrases in CJK (Chinese, Japanese, Korean) languages that often have multiple words appear in strings without a space separator, you need to add a space to separate the words. Otherwise, WebHelp will not find results. For example, Chinese uses a specialized character for space separators, but the current WebHelp implementation cannot detect such specialized characters, so to search for **开始之前** (it translates as "before you begin" or "before start"), you have to enter **开始 之前** (notice the space between the second and third symbols) in the search field.

Tip: The `<indexterm>` and `<keywords>` DITA elements are an effective way to increase the ranking of a page (for example, content inside a *keywords* element weighs more than an *H1* HTML element).

Excluded Terms

To improve performance, the **Search** feature excludes certain *stop words*. For example, the English version of the *stop words* includes: *a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the, their, then, there, these, they, this, to, was, will, with*.

Related Information:

WebHelp Responsive HTML5 Pages: Search Page *(on page 782)*

Oxygen Feedback Comments Component

You can add a comments component in your WebHelp Responsive output to provide a simple and efficient way for your community to interact and offer feedback. The comments component is contributed by **Oxygen Feedback**, a modern comment management system that can be integrated with your WebHelp Responsive output to provide a comments area at the bottom of each WebHelp page where readers can add new comments or reply to existing ones.

Oxygen Feedback includes a modern, user-friendly administration interface where you can moderate comments, manage users, view statistics, and configure settings. It is very easy to integrate and there are no requirements for installing additional software. You simply need to create an **Oxygen Feedback** site configuration in the administration interface, copy the HTML installation fragment that is generated at the end of the creation process, and paste the generated fragment in the **Feedback** tab in the WebHelp Responsive transformation scenario dialog box.

An add-on is also available that contributes a **Feedback Comments Manager** view in *Oxygen XML Editor/ Author* where the documentation team can see all the comments added in your WebHelp output. This means they can react to user feedback by making corrections and updating the source content without leaving the application.

Adding the Feedback System to WebHelp Responsive Documentation

Prerequisite

To install and manage **Oxygen Feedback**, you will need to obtain a license for the product. This requires that you choose a subscription plan during the installation procedure. To see the subscription plans prior to installing the product, go to: https://www.oxygenxml.com/oxygen_feedback/buy_feedback.html.

Installation Procedure

1. Log in to your Feedback account from the *administration login page* (https://feedback.oxygenxml.com/login). You can click on **Log in with Google** or **Log in with Facebook** to create an account using your Google or Facebook credentials, or click the **Sign Up** tab to create an account using your name and email address.
2. Click the **Add site** button to create a site (a Feedback configuration). If you have not already selected a subscription plan, you will be directed to a page where you can choose from several options.
3. In the **Settings** page, enter a **Name** and **Description** for the site configuration. There are some optional settings that can be adjusted according to your needs. For more details, see the Site Settings topic. Click **Continue**.
4. In the **Initial version** page, enter the **Base URL** for your website (you can add additional URLs by clicking the **Add** button). You can also specify an **Initial version** if you want it to be something other than *1.0*. If you do not plan to have multiple versions, leave the version as *1.0*. For more details, see the Initial Version topic. Click **Continue**.
5. In the **Installation** page, choose a site generation option:
    a. If you will generate the documentation using a transformation scenario in *Oxygen XML Editor/Author*, select the **Oxygen XML Editor** option. Then, click **Finish**, open *Oxygen XML Editor/Author*, and continue with these steps:
        i. Open the **Configure Transformation Scenario(s)** dialog box.
        ii. Select and duplicate the **DITA Map WebHelp Responsive** scenario.
        iii. Go to the **Feedback** tab.
        iv. Click the **Edit** button and paste the generated installation fragment.
    b. If you will generate the documentation using a command-line script, select the **Oxygen XML WebHelp** option and continue with these steps:
        i. Create an XML file (for example, `feedback-install.xml`) with the generated installation fragment. The required content for the file is displayed on-screen.
        ii. Use the `webhelp.fragment.feedback` parameter in your command-line script to specify the path to the file you just created. For example:

```
dita.bat -Dwebhelp.fragment.feedback=c:\path\to\feedback-install.xml
```

For more details about **Oxygen Feedback**, how to configure settings, moderate comments, view statistics, and much more, see the **Oxygen Feedback** user guide.

Context-Sensitive Help 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, 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 (on page 1328)*. 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.

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
```

Note: The `contextId` parameter is not case-sensitive.

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.

Note: You can use an *anchor (on page 1323)* 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
```

Another parameter indicates the search query:

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`.

Accessibility

Oxygen XML WebHelp Responsive output is compliant with the Section 508 accessibility standard, making the output accessible for people with visual impairment and other disabilities. Documentation and interface components are considered accessible when they have support in place that allows those with disabilities to use assistive technologies to understand the content.

Generally speaking, the WebHelp Responsive output has two major parts: topic content and WebHelp Responsive-related components (publication TOC, breadcrumb, menu). While the WebHelp Responsive components are designed to comply with the accessibility rules, it is important to adhere to some rules when you write DITA topics so that the content is also accessible.

Related Information:

DITA-OT Day 2017 Presentation: Accessibility in DITA-OT

Writing Guidelines for Accessible Documentation

To create accessible content, good authoring practices involve following guidelines, such as marking table headers, using semantic elements where available, and using alternative text for images.

Accessible Images

Images must have text alternatives that describe the information or function represented by them.

Short Text Equivalents for Images

When using the `<image>` element, specify a short alternative text with the `<alt>` element.

```
<image href="puffin.jpg">
  <alt>Puffin figure</alt>
</image>
```

Long Descriptions of Images

For complex images, when a short text equivalent does not suffice to adequately convey the function or role of an image, provide additional information in a file designated by the `<longdescref>` element.

```
<image href="puffin.jpg">
  <alt>Puffin figure</alt>
  <longdescref href="http://www.example.org/birds/puffin.html"
               scope="external"
               format="html"/>
</image>
```

Related Information:

Darwin Information Typing Architecture (DITA) Specification `<image>` element

Web Accessibility Tutorials: Alt Decision Tree

Accessible Image Maps

For image maps, text alternatives are needed on both the `<image>` element itself (to describe the informative context) and on each of the `<area>` elements (to convey the link destination or the action that will be initiated if the link is followed). The `<xref>` content within the `<area>` element contains the intended alternative text or hover text for that image map area.

```
<imagemap id="gear_pump_map">
  <image href="../images/Gear_pump_exploded.png" id="gear_pump_exploded">
    <alt>Gear Pump</alt>
  </image>
  <area>
    <shape>circle</shape>
      <coords>172, 265, 14</coords>
```

```
      <xref href="parts/bushings.dita#bushings_topic/bushings"
            format="dita">Bushings</xref>
  </area>
  <area>
      <shape>circle</shape>
      <coords>324, 210, 14</coords>
      <xref href="parts/ports.dita#ports_topic/sucction_port" format="dita"
        >Suction Port</xref>
   </area>
</imagemap>
```

Related Information:

Darwin Information Typing Architecture (DITA) Specification `<imagemap>` element

Accessible Tables

Accessible HTML tables need markup that indicates header cells and data cells and defines their relationship. Header cells must be marked with `<th>`, and data cells with `<td>`, to make tables accessible. For more complex tables, explicit associations may be needed using `@scope`, `@id`, and `@headers` attributes.

When you implement the table, it is best to use the `<table>` element (CALS table or OASIS Table Exchange Model). The `<table>` element includes all that you need to make a fully accessible table.

Related Information:

Darwin Information Typing Architecture (DITA) Specification `<table>` element

Table with Header Cells in the Top Row Only

For this type of table, you have to embed the table rows in the `<thead>` element.

Table 32. Example: Oxygen Events

Event	Date	Location
Evolution of TC 2018	May 31 - June 1, 2018	Sofia, Bulgaria
Markup UK	June 9 - 10, 2018	London, United Kingdom
Balisage 2018 - The Markup Conference	July 31 - August 3, 2018	Rockville, Maryland, USA

```
<table colsep="1" rowsep="1" frame="all">
  <title>
    <b>Oxygen Events</b>
  </title>
  <tgroup cols="3">
    <colspec colname="COLSPEC0" colwidth="1*"/>
```

```
    <colspec colname="COLSPEC1" colwidth="1.1*"/>
    <colspec colname="newCol3" colwidth="1*"/>
    <thead>
      <row>
        <entry colname="COLSPEC0" valign="top">Event</entry>
        <entry colname="COLSPEC1" valign="top">Date</entry>
        <entry>Location</entry>
      </row>
    </thead>
    <tbody>
      <row>
        <entry>Evolution of TC 2018</entry>
        <entry>May 31 - June 1, 2018</entry>
        <entry>Sofia, Bulgaria</entry>
      </row>
      <row>
        <entry>Markup UK</entry>
        <entry>June 9 - 10, 2018</entry>
        <entry>London, United Kingdom</entry>
      </row>
      <row>
        <entry>Balisage 2018 - The Markup Conference</entry>
        <entry>July 31 - August 3, 2018</entry>
        <entry>Rockville, Maryland, USA</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

Table with Header Cells in the First Column Only

For this type of table, you have to set the `rowheader="firstcol"` attribute on the `<table>` element to identify the header column.

Table 33. Example: Oxygen Events

Event	Evolution of TC 2018	Markup UK	Balisage 2018 - The Markup Conference
Date	May 31 - June 1, 2018	June 9 - 10, 2018	July 31 - August 3, 2018
Location	Sofia, Bulgaria	London, United Kingdom	Rockville, Maryland, USA

```
<table rowheader="firstcol" colsep="1" rowsep="1" frame="all">
  <title>
```

```
    <b>Oxygen Events</b>
  </title>
  <tgroup cols="4">
    <colspec colname="COLSPEC0" colwidth="1*"/>
    <colspec colname="COLSPEC1" colwidth="1.1*"/>
    <colspec colname="newCol3" colwidth="1*"/>
    <colspec colname="newCol4" colwidth="1*"/>
    <tbody>
      <row>
        <entry>Event</entry>
        <entry>Evolution of TC 2018</entry>
        <entry>Markup UK</entry>
        <entry>Balisage 2018 - The Markup Conference</entry>
      </row>
      <row>
        <entry>Date</entry>
        <entry>May 31 - June 1, 2018</entry>
        <entry>June 9 - 10, 2018</entry>
        <entry>July 31 - August 3, 2018</entry>
      </row>
      <row>
        <entry>Location</entry>
        <entry>Sofia, Bulgaria</entry>
        <entry>London, United Kingdom</entry>
        <entry>Rockville, Maryland, USA</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

Table with Header Cells in the Top Row and First Column

For this type of table, you can use `<thead>` to identify header rows and `@rowheader` to identify a header column.

Table 34. Example: Bus Timetable

	Mon-day	Tues-day	Wednes-day	Thurs-day	Friday
09:00 - 11:00	Closed	Open	Open	Closed	Closed
11:00 - 13:00	Open	Open	Closed	Closed	Closed
13:00 - 15:00	Open	Open	Open	Closed	Closed
15:00 - 17:00	Closed	Closed	Closed	Open	Open

```
<table id="table_dqk_n24_vdb" rowheader="firstcol" colsep="1" rowsep="1" frame="all">
   <title>Example: Bus Timetable</title>
   <tgroup cols="6">
     <colspec colnum="1" colname="col1"/>
     <colspec colnum="2" colname="col2"/>
     <colspec colnum="3" colname="col3"/>
     <colspec colnum="4" colname="col4"/>
     <colspec colnum="5" colname="col5"/>
     <colspec colnum="6" colname="col6"/>
     <thead>
       <row>
        <entry/>
        <entry>Monday</entry>
        <entry>Tuesday</entry>
        <entry>Wednesday</entry>
        <entry>Thursday</entry>
        <entry>Friday</entry>
       </row>
     </thead>
     <tbody>
      <row>
        <entry>09:00 - 11:00</entry>
        <entry>Closed</entry>
        <entry>Open</entry>
        <entry>Open</entry>
        <entry>Closed</entry>
        <entry>Closed</entry>
      </row>
      <row>
        <entry>11:00 - 13:00</entry>
        <entry>Open</entry>
        <entry>Open</entry>
        <entry>Closed</entry>
        <entry>Closed</entry>
        <entry>Closed</entry>
      </row>
      <row>
        <entry>13:00 - 15:00</entry>
        <entry>Open</entry>
        <entry>Open</entry>
        <entry>Open</entry>
        <entry>Closed</entry>
```

```
        <entry>Closed</entry>
      </row>
      <row>
        <entry>15:00 - 17:00</entry>
        <entry>Closed</entry>
        <entry>Closed</entry>
        <entry>Closed</entry>
        <entry>Open</entry>
        <entry>Open</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

WebHelp Responsive VPAT Accessibility Conformance Report

International Edition

VPAT® Version 2.3 – April 2019

Product Name/Version

Oxygen XML WebHelp Responsive

Product Description

Oxygen XML WebHelp Responsive enables you to publish DITA content on the web and present it in a user-friendly interface that is easy to navigate. You can design your WebHelp Responsive output to be available on desktop systems or various mobile devices. With **Oxygen XML WebHelp Responsive**, your published content is accessible, interactive, and convenient.

Date

May 2019

Contact Information

support@oxygenxml.com

Notes

Oxygen XML WebHelp Responsive has been designed and enhanced to adhere to the U.S. Government Section 508 accessibility standards and the Web Content Accessibility Guidelines (WCAG). For details, see WebHelp Responsive Accessibility *(on page 791)*.

Evaluation Methods Used:

The following applications were used for testing **Oxygen XML WebHelp Responsive**:

- Desktop browsers: Chrome, Firefox, Safari, Edge, and Internet Explorer
- Assistive technologies: NVDA, VoiceOver, JAWS, Microsoft Narrator

Applicable Standards/Guidelines

This report covers the degree of conformance for the following accessibility standards/guidelines:

Standard/Guideline	Included In Report
Web Content Accessibility Guidelines 2.0	Level A - Yes Level AA - Yes Level AAA - No
Web Content Accessibility Guidelines 2.1	Level A - Yes Level AA - Yes Level AAA - No
Revised Section 508 standards published January 18, 2017 and corrected January 22, 2018	Yes
EN 301 549 Accessibility requirements suitable for public procurement of ICT products and services in Europe - V2.1.2 (2018-08)	No

Terms

The terms used in the Conformance Level information are defined as follows:

- **Supports**: The functionality of the product has at least one method that meets the criterion without known defects or meets with equivalent facilitation.
- **Partially Supports**: Some functionality of the product does not meet the criterion.
- **Does Not Support**: The majority of product functionality does not meet the criterion.
- **Not Applicable**: The criterion is not relevant to the product.
- **Not Evaluated**: The product has not been evaluated against the criterion. This can be used only in WCAG 2.0 Level AAA.

WCAG 2.x Report

Tables 1 and 2 also document conformance with:

Revised Section 508: Chapter 5 – 501.1 Scope, 504.2 Content Creation or Editing, and Chapter 6 – 602.3 Electronic Support Documentation.

Note: When reporting on conformance with the WCAG 2.x Success Criteria, they are scoped for full pages, complete processes, and accessibility-supported ways of using technology as documented in theWCAG 2.0 Conformance Requirements.

Table 1: Success Criteria, Level A

Criteria	Conformance Level	Remarks and Explanations
1.1.1 Non-text Content (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Partially Supports	Text alternatives are provided for many instances of non-text content, with exceptions that include: • Permalinks for subtopics and sections. • Enlarge image action.
1.2.1 Audio-only and Video-only (Prerecorded) (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The authors of the input DITA document are responsible for providing a transcript of the media content in the document.
1.2.2 Captions (Prerecorded) (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The product does not provide prerecorded media that requires captions.
1.2.3 Audio Description or Media Alternative (Prerecorded) (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The authors of the input DITA document are responsible for providing an alternative for time-based media or audio description of the prerecorded video content in the document. See: G58: Placing a link to the alternative for time-based media immediately next to the non-text content
1.3.1 Info and Relationships (Level A) Also applies to: Revised Section 508	Partially Supports	Information, structure, and relationships conveyed through presentation can be programmatically determined or are available in text, with exceptions that include:

Criteria	Conformance Level	Remarks and Explanations
• 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)		• Some landmarks are not marked with the corresponding role or do not have an associated label. • Some link groups are not structured using lists or are not marked as navigation regions. The authors of the input DITA document are responsible for: • Using semantic elements to mark up structure. • Using semantic markup to mark emphasized or special text. • Using caption elements to associate data table captions with data tables.
1.3.2 Meaningful Sequence (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The product presents content in a meaningful sequence. Authors should use Unicode right-to-left mark (RLM) or left-to-right mark (LRM) to mix text direction inline.
1.3.3 Sensory Characteristics (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	Authors should ensure that items are referenced in the content in ways that do not depend on sensory perception.
1.4.1 Use of Color (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software)	Supports	(Cobalt template) Color is not used as the only visual means of conveying information, indicating an action, prompting a response, or distinguishing a visual element.

Criteria	Conformance Level	Remarks and Explanations
• 504.2 (Authoring Tool) • 602.3 (Support Docs)		
1.4.2 Audio Control (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	There is no sound that plays automatically.
2.1.1 Keyboard (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Partially Supports	Most of the content is operable through a keyboard interface, with exceptions that include: • The submenus (the user cannot tab to the submenus). • The top-level links in the main page accordion cannot be accessed.
2.1.2 No Keyboard Trap (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The product does not contain content that traps the keyboard focus.
2.1.4 Character Key Shortcuts (Level A 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	The product does not include character key shortcuts.
2.2.1 Timing Adjustable (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The product does not include time limits.

Criteria	Conformance Level	Remarks and Explanations
2.2.2 Pause, Stop, Hide (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The product does not include elements that move, blink, scroll, or auto-update.
2.3.1 Three Flashes or Below Threshold (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The product does not contain flashing content.
2.4.1 Bypass Blocks (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) – Does not apply to non-web software • 504.2 (Authoring Tool) • 602.3 (Support Docs) – Does not apply to non-web docs	Supports	Each page contains a link at the top that goes directly to the main content area. Each page contains *ARIA* landmarks that identify the available regions.
2.4.2 Page Titled (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	Each page contains a non-empty `<title>` element in the `<head>` section.
2.4.3 Focus Order (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool)	Supports	Focusable components receive focus in an order that preserves meaning and operability.

Criteria	Conformance Level	Remarks and Explanations
• 602.3 (Support Docs)		
2.4.4 Link Purpose (In Context) (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The purpose of each link can be determined from the link text alone or from the link text together with its programmatically-determined link context. The authors can create hypertext links using text that describes the purpose of the hypertext. There is no control that allows the user to choose between short or long link text (G189 / SCR30).
2.5.1 Pointer Gestures (Level A 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	The WebHelp Responsive output does not rely on path-based or multi-point gestures and does not provide controls that require complex gestures.
2.5.2 Pointer Cancellation (Level A 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	The product has operations that are activated on the pointer up event.
2.5.3 Label in Name (Level A 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	The names of the user interface components contain the text that is presented visually.
2.5.4 Motion Actuation (Level A 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	The product does not contain functionality that can be operated by device or user motion.
3.1.1 Language of Page (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The web pages indicate the language of the content when the content language has been specified by authors.

Criteria	Conformance Level	Remarks and Explanations
3.2.1 On Focus (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	No changes of context occur when any component receives focus.
3.2.2 On Input (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	Changing the setting of any user interface component does not automatically cause a change of context.
3.3.1 Error Identification (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Partially Supports	If a search operation is performed leaving the search input empty, an error message is automatically displayed to the user, but no *aria-invalid* information is provided.
3.3.2 Labels or Instructions (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Partially Supports	The search input does not have a visible label specified using a label element.
4.1.1 Parsing (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Partially Supports	Several HTML validation errors are reported by the W3C validator.

Criteria	Conformance Level	Remarks and Explanations
4.1.2 Name, Role, Value (Level A) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Partially Supports	The *Home* link from the breadcrumb does not have an associated *aria-label*.

Table 2: Success Criteria, Level AA

Criteria	Conformance Level	Remarks and Explanations
1.2.4 Captions (Live) (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	No live audio content is used.
1.2.5 Audio Description (Prerecorded) (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The authors of the input DITA document can ensure that the output document meets this criterion.
1.3.4 Orientation (Level AA 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	Content does not restrict its view and operation to a single display orientation.
1.3.5 Identify Input Purpose (Level AA 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	The content does not contain input fields that collect information about the user.
1.4.3 Contrast (Minimum) (Level AA) Also applies to: Revised Section 508	Partially Supports	The missing words element from the search results page does not have the contrast ratio 4.5:1.

Criteria	Conformance Level	Remarks and Explanations
• 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)		
1.4.4 Resize text (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Partially Supports	Text can be resized up to 200 percent without loss of content or functionality and without using assistive technology. Some text content has dimensions specified in pixels rather that em units.
1.4.5 Images of Text (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The output does not contain images of text. The authors of the input DITA content can ensure that this criterion is met.
1.4.10 Reflow (Level AA 2.1 only) Also applies to: Revised Section 508 – Does not apply	Partially Supports	The majority of the content can be presented without loss of information or functionality, and without requiring scrolling in two dimensions. Long URLs determine the page to display the horizontal scroll bar.
1.4.11 Non-text Contrast (Level AA 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	(Cobalt template) There is no contrast issue regarding user interface components or graphical objects.
1.4.12 Text Spacing (Level AA 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	There is no loss of content or functionality that occurs by setting line height (line spacing), spacing following paragraphs, letter spacing, and word spacing.
1.4.13 Content on Hover or Focus (Level AA 2.1 only) Also applies to: Revised Section 508 – Does not apply	Partially Supports	Tooltips and submenus are not dismissible. Also, the tooltips are not hoverable.

Criteria	Conformance Level	Remarks and Explanations
2.4.5 Multiple Ways (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) – Does not apply to non-web software • 504.2 (Authoring Tool) • 602.3 (Support Docs) – Does not apply to non-web docs	Supports	There is a search form provided that will go to a page that contains the search term and links to the corresponding page. Also, a table of contents is provided. The authors of the input DITA document are responsible for providing links to all pages from the home page or providing links to navigate to related pages from the current page.
2.4.6 Headings and Labels (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	Headings and labels describe the topic or purpose. DITA authors can ensure that this criterion is met.
2.4.7 Focus Visible (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Partially Supports	Placing focus on a focusable element using the mouse doesn't render a visible focus indicator. Also, the search button does not have a visible focus indicator.
3.1.2 Language of Parts (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	DITA authors can ensure that this criterion is met.
3.2.3 Consistent Navigation (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) – Does not apply to non-web software	Supports	Repeated components appear in the same relative in each page.

Criteria	Conformance Level	Remarks and Explanations
• 504.2 (Authoring Tool) • 602.3 (Support Docs) – Does not apply to non-web docs		
3.2.4 Consistent Identification (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) – Does not apply to non-web software • 504.2 (Authoring Tool) • 602.3 (Support Docs) – Does not apply to non-web docs	Partially Supports	The output uses labels, names, and text alternatives consistently for items that have the same functionality. Text alternatives are provided for many instances of non-text content, with exceptions that include: • Permalinks for subtopics and sections. • Enlarge images action. The *Home* link from the breadcrumb does not have an associated *aria-label*.
3.3.3 Error Suggestion (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Does Not Support	The Search input does not have the *aria-required* information set and does not contain a text description specifying that it is a required field.
3.3.4 Error Prevention (Legal, Financial, Data) (Level AA) Also applies to: Revised Section 508 • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs)	Supports	The Web pages do not cause legal commitments or financial transactions for the user to occur, that modify or delete user-controllable data in data storage systems, or that submit user test responses.
4.1.3 Status Messages(Level AA 2.1 only) Also applies to: Revised Section 508 – Does not apply	Supports	The pages do not contain status messages as defined by this criterion.

Table 3: Success Criteria, Level AAA

Criteria	Conformance Level	Remarks and Explanations
1.2.6 Sign Language (Prerecorded) (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
1.2.7 Extended Audio Description (Pre-recorded) (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
1.2.8 Media Alternative (Prerecorded) (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
1.2.9 Audio-only (Live) (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
1.3.6 Identify Purpose (Level AAA 2.1 only) Revised Section 508 – Does not apply	Not Evaluated	
1.4.6 Contrast Enhanced (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
1.4.7 Low or No Background Audio (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
1.4.8 Visual Presentation (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
1.4.9 Images of Text (No Exception) Control (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
2.1.3 Keyboard (No Exception) (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
2.2.3 No Timing (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
2.2.4 Interruptions (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	

Criteria	Conformance Level	Remarks and Explanations
2.2.5 Re-authenticating (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
2.2.6 Timeouts (Level AAA 2.1 only) Revised Section 508 – Does not apply	Not Evaluated	
2.3.2 Three Flashes (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
2.3.3 Animation from Interactions (Level AAA 2.1 only) Revised Section 508 – Does not apply	Not Evaluated	
2.4.8 Location (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
2.4.9 Link Purpose (Link Only) (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
2.4.10 Section Headings (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
2.5.5 Target Size (Level AAA 2.1 only) Revised Section 508 – Does not apply	Not Evaluated	
2.5.6 Concurrent Input Mechanisms (Level AAA 2.1 only) Revised Section 508 – Does not apply	Not Evaluated	
3.1.3 Unusual Words (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
3.1.4 Abbreviations (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
3.1.5 Reading Level (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
3.1.6 Pronunciation (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
3.2.5 Change on Request (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	

Criteria	Conformance Level	Remarks and Explanations
3.3.5 Help (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	
3.3.6 Error Prevention (All) (Level AAA) Revised Section 508 – Does not apply	Not Evaluated	

Revised Section 508 Report

N/A

Chapter 3: Functional Performance Criteria (FPC)

Criteria	Conformance Level	Remarks and Explanations
302.1 Without Vision	Partially Supports	Most of the content is accessible without vision with exceptions that include: • Some components do not have text alternatives or labels. • Some landmarks are not marked with the corresponding role or do not have an associated label. • Some link groups are not structured using lists or are not marked as navigation regions.
302.2 With Limited Vision	Partially Supports	Most of the content is accessible with limited vision with exceptions that include: • Some components do not have text alternatives or labels. • Some landmarks are not marked with the corresponding role or do not have an associated label. • Some link groups are not structured using lists or are

Criteria	Conformance Level	Remarks and Explanations
		not marked as navigation regions.
302.3 Without Perception of Color	Supports	(Cobalt template) Color is not used as the only visual means of conveying information, indicating an action, prompting a response, or distinguishing a visual element.
302.4 Without Hearing	Supports	The authors can create content that does not require hearing abilities for use.
302.5 With Limited Hearing	Supports	The authors can create content that does not require hearing abilities for use.
302.6 Without Speech	Supports	The output does not require speech for use.
302.7 With Limited Manipulation	Supports	The WebHelp Responsive output does not rely on path-based or multi-point gestures and does not provide controls that require complex gestures.
302.8 With Limited Reach and Strength	Supports	The WebHelp Responsive output does not rely on path-based or multi-point gestures and does not provide controls that require complex gestures.
302.9 With Limited Language, Cognitive, and Learning Abilities	Supports	The authors can create content that can be used by users with limited language, cognitive, and learning abilities.

Chapter 4: Hardware

Notes: Not Applicable - **Oxygen XML WebHelp Responsive** is not a hardware product.

Chapter 5: Software

Notes: **Oxygen XML WebHelp Responsive** is a web application, not a software product. However, the web application includes authoring functionality, hence Chapter 5: Software 504 Authoring Tools applies to this product.

501 General

Criteria	Conformance Level	Remarks and Explanations
501.1 Scope – Incorporation of WCAG 2.0 AA	See WCAG 2.x section *(on page 798)*	See information in WCAG section

502 Interoperability with Assistive Technology

Criteria	Conformance Level	Remarks and Explanations
502.2.1 User Control of Accessibility Features	Not Applicable	The product is not platform software.
502.2.2 No Disruption of Accessibility Features	Supports	The product does not disrupt platform features that are defined in the platform documentation as accessibility features.

502.3 Accessibility Services

Criteria	Conformance Level	Remarks and Explanations
502.3.1 Object Information	Partially Supports	The majority of object roles, state(s), properties, boundary, name, and description are programmatically determinable. The *Home* link from the breadcrumb does not have an associated *aria-label*.
502.3.2 Modification of Object Information	Supports	States and properties that can be set by the user can be set programmatically.
502.3.3 Row, Column, and Headers	Supports	The headers associated with the rows or columns of a table can be programmatically determined.
502.3.4 Values	Supports	The current values of an object can be programmatically determined.
502.3.5 Modification of Values	Supports	Values that can be set by the user are capable of being set programmatically.
502.3.6 Label Relationships	Partially Supports	Information, structure, and relationships conveyed through presentation can be programmatically determined or are available in text.

Criteria	Conformance Level	Remarks and Explanations
		See WCAG 1.3.1 *(on page 799)*.
502.3.7 Hierarchical Relationships	Supports	The content is hierarchically structured using language-specific elements and their relationships can be programmatically determined.
502.3.8 Text	Supports	The content of text objects, text attributes, and the boundary of text rendered to the screen shall be programmatically determinable.
502.3.9 Modification of Text	Supports	The editable text (search input) can be set programmatically.
502.3.10 List of Actions	Not Applicable	There are no custom actions available that can be executed on the content.
502.3.11 Actions on Objects	Not Applicable	There are no custom actions available that can be executed on the content.
502.3.12 Focus Cursor	Not Applicable	The product is a web application and is isolated from the underlying platform software (web browser).
502.3.13 Modification of Focus Cursor	Not Applicable	The product is a web application and is isolated from the underlying platform software (web browser).
502.3.14 Event Notification	Not Applicable	There are no automatic focus changes, caret movement, selection changes, or added components in the content.
502.4 Platform Accessibility Features	Not Applicable	This product is not platform software.

503 Applications

Criteria	Conformance Level	Remarks and Explanations
503.2 User Preferences	Not Applicable	This section does not apply to web applications.

Criteria	Conformance Level	Remarks and Explanations
503.3 Alternative User Interfaces	Not Applicable	The application does not provide an alternative user interface that functions as assistive technology.

503.4 User Controls for Captions and Audio Description

Criteria	Conformance Level	Remarks and Explanations
503.4.1 Caption Controls	Not Applicable	The product does not provide controls for volume adjustment.
503.4.2 Audio Description Controls	Not Applicable	The product does not provide controls for program selection.

504 Authoring Tools

Criteria	Conformance Level	Remarks and Explanations
504.2 Content Creation or Editing (if not authoring tool, enter “not applicable”)	Not Applicable See the WCAG 2.x section *(on page 798)*	The product is not an authoring tool. See information in WCAG section
504.2.1 Preservation of Information Provided for Accessibility in Format Conversion	Not Applicable	The product is not an authoring tool.
504.2.2 PDF Export	Not Applicable	The product is not an authoring tool.
504.3 Prompts	Not Applicable	The product is not an authoring tool.
504.4 Templates	Not Applicable	The product is not an authoring tool.

Chapter 6: Support Documentation and Services

601.1 Scope

602 Support Documentation

Criteria	Conformance Level	Remarks and Explanations
602.2 Accessibility and Compatibility Features	Partially Supports	The product documentation is distributed in the WebHelp Responsive format. See the Chapter 3 *(on page 811)* and Chapter 5 *(on page 812)* sections.

Criteria	Conformance Level	Remarks and Explanations
602.3 Electronic Support Documentation	See the WCAG 2.x section *(on page 798)*	See information in the WCAG section.
602.4 Alternate Formats for Non-Electronic Support Documentation	Not Applicable	Documentation is not provided in non-electronic formats.

603 Support Services

Criteria	Conformance Level	Remarks and Explanations
603.2 Information on Accessibility and Compatibility Features	Supports	The support services cover the accessibility features.
603.3 Accommodation of Communication Needs	Supports	Support services are available by phone or e-mail.

Legal Disclaimer

This report describes **Oxygen XML WebHelp's** ability to support the stated VPAT Standards/Guidelines, subject to Syncro Soft's interpretation of the same. This accessibility report is provided for informational purposes only, and the contents hereof are subject to change without notice. SYNCRO SOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT. For more information regarding the accessibility status, please contact us at *sales@oxygenxml.com*.

© 2019 Syncro Soft SRL. All rights reserved.

Publishing Templates

An *Oxygen Publishing Template* defines all aspects of the layout and styles for output obtained from the following transformation scenarios:

- **WebHelp Responsive**
- **DITA Map PDF - based on HTML5 & CSS**

It is a self-contained customization package stored as a ZIP archive or folder that can easily be shared with others. It provides the primary method for customizing the output.

Some possible customization methods include:

- Add additional template resources to customize the output (such as logos, *Favicons*, or CSS files).
- Extend the default processing by specifying one or more XSLT extension points.
- Specify one or more transformation parameters to customize the output.
- Customize various aspects of the output through simple CSS styling.
- For **WebHelp Responsive** output, change the layout of the main page or topic pages by customizing which components will be displayed and where they will be positioned in the page.

The following graphics are possible sample structures for *Oxygen Publishing Template* packages:

Figure 293. Oxygen Publishing Template Package (WebHelp Responsive)

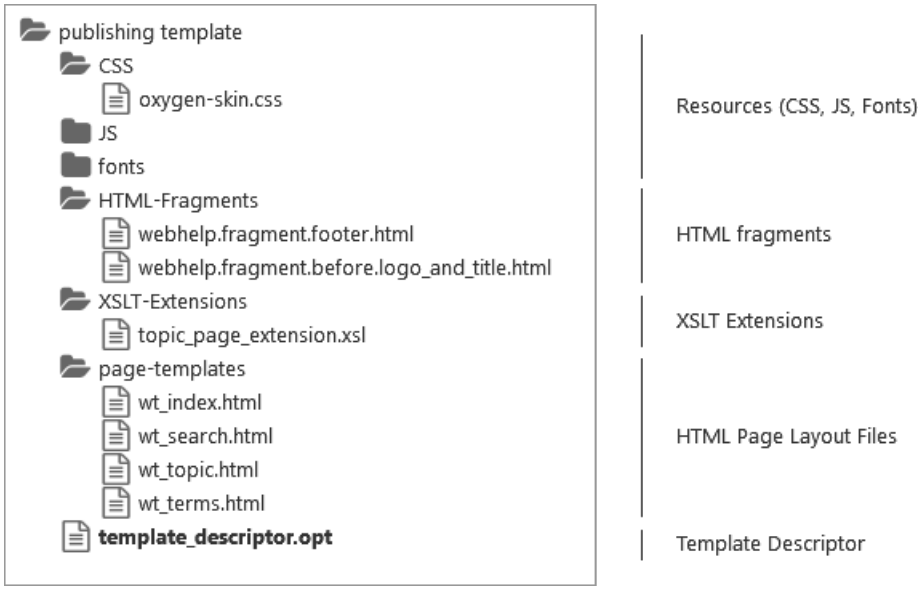


Figure 294. Oxygen Publishing Template Package (PDF)

For information about creating and customizing publishing templates, and how to adjust the WebHelp and PDF output through CSS styling and other customization methods, watch our Webinar: **Creating Custom Publishing Templates for WebHelp and PDF Output**. The Webinar slides and sample project are also available from that webpage.

Related Information:

How to Create a Publishing Template *(on page 848)*

How to Edit a Packed Publishing Template *(on page 850)*

How to Add a Publishing Template to the Publishing Templates Gallery *(on page 850)*

How to Share a Publishing Template *(on page 971)*

Publishing Templates Gallery

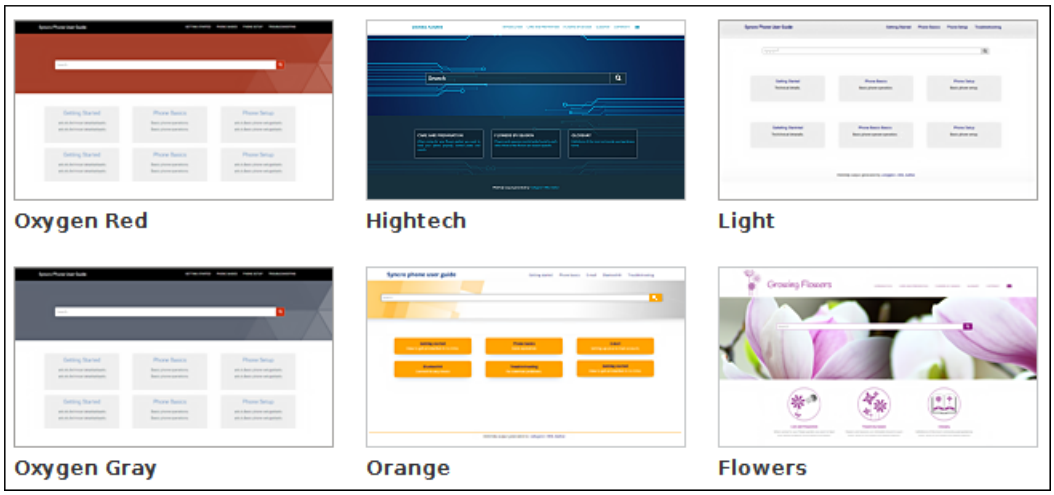
Oxygen XML Developer Eclipse plugin comes bundled with a variety of built-in templates. You can use one of them to publish your documentation or as a starting point for a new publishing template.

Built-in Templates

There are two categories of templates, *Tiles* and *Tree*. You can see the built-in templates in the **Templates** tab when editing a WebHelp Responsive transformation scenario in **Oxygen XML Editor/Author**. Each one also includes an **Online preview** icon in the bottom-right corner that opens a webpage in your default browser that provides a sample of how the main page will look when that particular template is used to generate the output.

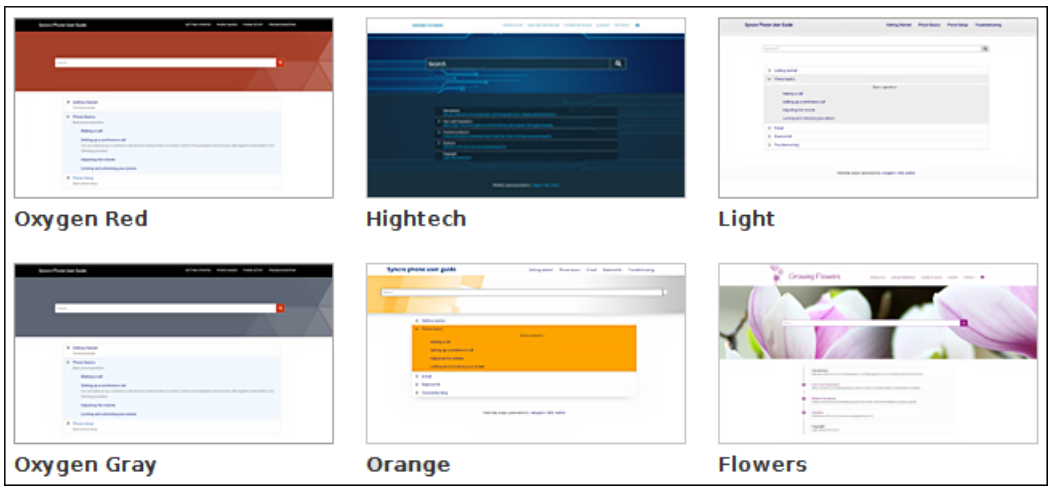
Tiles Templates

The main page in the WebHelp output presents a tile for each main topic (chapter) of the documentation.



Tree Templates

The main page in the WebHelp output presents a tree-like table of contents.



Built-in Templates Location

All built-in templates are stored in the following directory: *DITA-OT-DIR*`/plugins/com.oxygenxml.webhelp.responsive/templates`.

Custom Templates

You can use a built-in template as a starting point for creating your own custom template *(on page 968)*.
You can store all of your custom templates in a particular directory. Then, go to **Options > Preferences > DITA > Publishing** and add your directory to the list, and all the templates stored in that directory will be displayed in the preview pane in the transformation scenario's **Template** tab along with all the built-in templates.

Sharing Publishing Template

To share a publishing template with others, following these steps:

1. Copy your template in a new folder.
2. Go to **Options > Preferences > DITA > Publishing** and add that new folder to the list.
3. Switch the option as the bottom of that preferences page to **Project Options**.
4. Share your project file (`.xpr`).

Publishing Template Package Contents for WebHelp Responsive Customizations

An *Oxygen Publishing Template* package for WebHelp output must contain a template descriptor file and at least one CSS file, and may contain other resources (such as graphics, XHTML files, XSLT files, etc.). All the template resources can be stored in either a ZIP archive or in a folder. It is recommended to use a ZIP archive because it is easier to share with others.

Template Descriptor File

Each publishing template includes a descriptor file that defines the meta-data associated with the template. It is an XML file that defines all the resources included in a template (such as CSS files, images, JS files, and transformation parameters).

The template descriptor file must have the `.opt` file extension and must be located in the template's root folder.

A template descriptor might look like this:

```
<publishing-template>
    <name>Flowers</name>
    <webhelp>
        <tags>
            <tag>tree</tag>
            <tag>light</tag>
        </tags>
        <preview-image file="flowers-tree.png"/>

        <!-- Resources (CSS, favicon, logo and others) -->
        <resources>
```

```
            <!-- Main CSS file -->
            <css file="flowers.css"/>

             <!-- Resources to copy to the output folder -->
            <fileset>
                <include name="resources/**/*"/>
                <exclude name="resources/**/.svn"/>
                <exclude name="resources/**/.git"/>
            </fileset>
        </resources>

        <parameters>
            <parameter name="webhelp.show.main.page.tiles" value="no"/>
            <parameter name="webhelp.show.main.page.toc" value="yes"/>
            <parameter name="webhelp.top.menu.depth" value="3"/>
        </parameters>
    </webhelp>
</publishing-template>
```

Tip: It is recommended to edit the template descriptor in **Oxygen XML Editor/Author** because it provides content completion and validation support.

Template Name and Description

Each template descriptor file requires a `<name>` element. This information is displayed as the name of the template in the transformation scenario dialog box.

Optionally, you can include a `<description>` and it displayed when the user hovers over the template in the transformation scenario dialog box.

```
<publishing-template>
    <name>Lorem Ipsum</name>
    <description>Lorem ipsum dolor sit amet, consectetur adipiscing elit</description>
    ...
```

Template Author

Optionally, you can include author information in the descriptor file and it displayed when the user hovers over the template in the transformation scenario dialog box. This information might be useful if users run into an issue or have questions about a certain template.

If you include the `<author>` element, a `<name>` is required and optionally you can include `<email>`, `<organization>`, and `<organizationUrl>` information.

```
<publishing-template>
    ...
    <author>
```

```
      <name>John Doe</name>
      <email>jdoe@example.com</email>
      <organization>ACME</organization>
      <organizationUrl>http://www.example.com/jdoe</organizationUrl>
    </author>
    ...
```

Webhelp Element

The `<webhelp>` element contains various details that define the WebHelp Responsive output. It is a required element if you intend on using a WebHelp Responsive transformation scenario. The elements that are allowed in this `<webhelp>` section specify the template tags *(on page 821)*, template preview image *(on page 822)*, resources *(on page 822)* (such as CSS, JS, fonts, logos), transformation parameters *(on page 823)*, HTML fragment extensions *(on page 825)* (used to add fragments to placeholders), XSLT extensions *(on page 824)*, or HTML page layout files *(on page 830)*.

```
    <webhelp>
        <tags>
          ...
        </tags>
        <preview-image file="MyPreview.png"/>
        <resources>
          ...
        </resources>
        <html-page-layouts>
          ...
        </html-page-layouts>
        <parameters>
          ...
        </parameters>
    </webhelp>
```

Template Tags

The `<tags>` section provides meta information about the template (such as layout type or color theme). Each *tag* is displayed at the top of the **Templates** tab window in the transformation scenario dialog box and they help the user filter and find particular templates.

```
<publishing-template>
    ...
    <webhelp>
      <tags>
        <tag>tree</tag>
        <tag>dark</tag>
      </tags>
```

Template Preview Image

The `<preview-image>` element is used to specify an image that will be displayed in the transformation scenario dialog box. It provides a visual representation of the template to help the user select the right template. The image dimensions should be `200 x 115` pixels and the supported image formats are: `JPEG`, `PNG`, or `GIF`.

You can also include an `<online-preview-url>` element to specify the URL of a published sample of your template. This will display an **Online preview** icon in the bottom-right corner the image in the transformation scenario dialog box and if the user clicks that icon, it will open the specified URL in their default browser.

```
<publishing-template>
    ...
    <webhelp>
      ...
      <preview-image file="ashes/ashes-tree.png"/>
      <online-preview-url>https://www.example.com/samples/tiles/ashes</online-preview-url>
```

Template Resources

The `<resources>` section of the descriptor file specifies a set of resources (CSS, JS, fonts, logos, graphics, etc.) that are used to customize various components in the generated output. These resources will be copied to the output folder during the transformation process. At least one CSS file must be included, while the other types of resources are optional.

This section is defined using the **resources** element and the types of resources that can be specified include:

- **CSS files** - One or more CSS files that will define the styles of all generated HTML pages. They are referenced using the `<css>` element.
- **Favicon** - You can specify the path to an image for the *favicon* associated with your website. It is referenced using the `<favicon>` element.
- **Logo** - You can specify the path to a logo image that will be displayed in the left side of the output header. It is referenced using the `<logo>` element. Optionally, you can also specify:
  - `<target-url>` - will redirect the user to the specified URL if they click the logo in the output.
  - `<alt>` - provides an alternate text for the logo image.
- **JavaScript AMD module** - The path to a JavaScript module that uses the AMD (Asynchronous Module Definition) format. This module will be loaded in the output HTML pages using the *RequireJS* library. It can be referenced using the `<js-amd-module>` element. For more information, see How to Insert JavaScript AMD Modules *(on page 861)*.
- **Additional Resources (graphics, JS, fonts)** - For other resources (such as images referenced in CSS, JS, fonts, etc.) that need to be included in the output, you need to instruct the transformation to include them in the output folder. You can specify one or more sets of additional resources to be copied to the output folder by using the `<fileset>` element and you can use one or more `<include>` and `<exclude>` elements. This semantic is similar to the ANT FileSet.

```
<publishing-template>
    ...
    <webhelp>
      ...
      <resources>
          <css file="css/custom_styles.css"/>
          <css file="css/custom_fonts.css"/>
          <favicon file="images/favicon.png"/>
          <logo
           file="images/logo.png"
           target-url="http://www.example.com"
           alt="Alternate text for the logo image"/>
          <js-amd-module file="js/template-main.js"/>
          <fileset>
              <include name="common/**/*"/>
              <include name="JS/**/*"/>
              <exclude name="**/*.svn"/>
              <exclude name="**/*.git"/>
          </fileset>
      </resources>
```

Note: All relative paths specified in the descriptor file are relative to the template root folder.

The resources specified in the template descriptor are copied to the following output folder: *[WebHelp_OUTPUT_DIR]*/oxygen-webhelp/template. The following graphic illustrates the mapping between the template resources and the location where they will be copied to the output folder:

Figure 295. Template Resources Mapping

Transformation Parameters

You can also set one or more WebHelp transformation parameters in the descriptor file.

```
<publishing-template>
    ...
    <webhelp>
      ...
      <parameters>
        <parameter
          name="webhelp.show.main.page.toc"
          value="yes"/>
        <parameter
          name="webhelp.top.menu.depth"
          value="3"/>
        <parameter
          name="webhelp.fragment.welcome"
          value="html-fragment/webhelp.fragment.welcome.html"
          type="filePath"/>
      </parameters>
    </webhelp>
```

The following information can be specified in the `<parameter>` element:

Parameter name

The name of the parameter. It may be one of the WebHelp Responsive transformation parameters *(on page 906)* or a DITA-OT HTML-based output parameter.

Note: It is not recommended to specify an input/output parameter in the descriptor file (such as the input Map, DITAVAL file, or temporary directory).

Attention: JVM arguments like *-Xmx* cannot be specified as a transformation parameter.

Parameter Value

The value of the parameter. It should be a relative path to the template root folder for file paths parameters.

Parameter Type

The type of the parameter: `string` or `filepath`. The `string` value is default.

After creating a publishing template *(on page 968)* and adding it to the templates gallery *(on page 850)*, when you select the template in the transformation scenario dialog box, the **Parameters** tab will automatically be updated to include the parameters defined in the descriptor file. These parameters are displayed in italics.

XSLT Extension Points

The publishing templates support one or more XSLT extension points. They are helpful when you want to change the structure of the HTML pages that are primarily generated from XSLT processing. They can be specified using the `<xslt>` element in the descriptor file using the following structure:

```
<publishing-template>
    ...
    <webhelp>
        ...
        <xslt>
            <extension
              id="com.oxygenxml.webhelp.xsl.dita2webhelp"
              file="xsl/customDita2webhelp.xsl"/>
            <extension
              id="com.oxygenxml.webhelp.xsl.createMainPage"
              file="xsl/customMainPage.xsl"/>
        </xslt>
```

For more information, and a full list of the available extension points, see: XSLT-Import and XSLT-Parameter Extension Points *(on page 912)*.

Note: You can read the value of a WebHelp transformation parameter from your XSLT extension stylesheets by using the `getParameter(param.name)` function from the `http://www.oxygenxml.com/functions` namespace.

HTML Fragment Extension Points

The HTML pages contain component placeholders that can be used to insert custom HTML fragments (either a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment). These fragments and their placeholder location are defined in the descriptor file using a `<fragment>` element inside the `<html-fragments>` section.

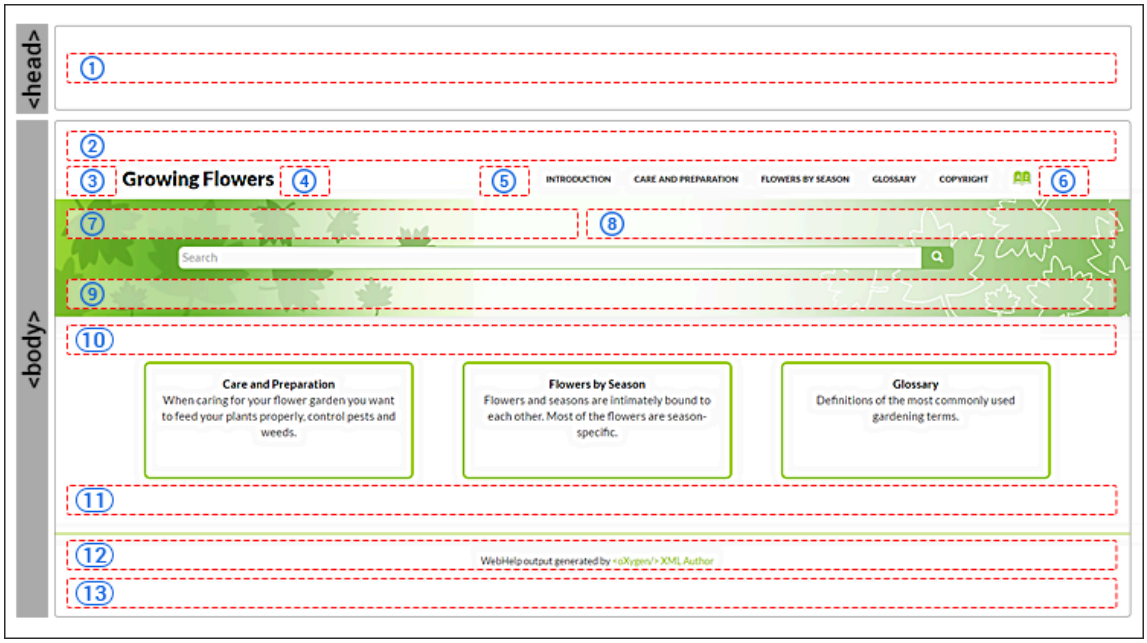
You can specify one or more HTML fragment extension points in the descriptor file using the following structure:

```
<publishing-template>
    ...
    <webhelp>
      ...
      <html-fragments>
        <fragment
          file="html-fragments/webhelp_fragment_welcome.html"
          placeholder="webhelp.fragment.welcome"/>
        <fragment
          file="html-fragments/webhelp_fragment_footer.html"
          placeholder="webhelp.fragment.footer"/>
      </html-fragments>
```

Some of these placeholders are left empty in the default output configurations, but you can use them to insert custom content.

Each placeholder has an associated parameter value in the transformation. These predefined placeholders are illustrated and described below.

Figure 296. Predefined Placeholders Diagram



1. Header *(on page 826)*
2. Before Body *(on page 826)*
3. Before Logo and Title *(on page 827)*
4. After Logo and Title *(on page 827)*
5. Before Top Menu *(on page 827)*
6. After Top Menu *(on page 827)*
7. Before Main Page Search Input *(on page 827)*
8. Welcome Message *(on page 827)*
9. After Main Page Search Input *(on page 827)*
10. Before Tiles or TOC *(on page 827)*
11. After Tiles or TOC *(on page 827)*
12. Footer *(on page 828)*
13. After Body *(on page 828)*

Each of these placeholders can hold either a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment:

1- webhelp.fragment.head

In the generated output it inserts a given XHTML fragment in the `<head>` element. The value of the parameter can be either a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment.

Important: This parameter should only be used if you are using a valid, purchased license of Oxygen XML Developer Eclipse plugin (do not use it with a trial license).

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 a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment.

Related Information:

How to Insert HTML Content *(on page 857)*

WebHelp Responsive Macros

You can use the `whc:macro` layout component to specify a macro value (a variable that will be expanded when the output files are generated).

A macro has the following syntax:

```
${macro-name}
```

or

```
${macro-name(macro-parameter)}
```

A macro name can accept any alphanumeric characters, as well as the following characters: `-` (minus), `_` (underscore), `.` (dot), `:` (colon). The value of a parameter may contain any character except the `}` (close curly bracket) character.

Implementations

The following *macros* are supported:

i18n

For localizing a string.

```
${i18n(string.id)}
```

param

Returns the value of a transformation parameter.

```
${param(webhelp.show.main.page.tiles)}
```

env

Returns the value of an environment variable.

```
${env(JAVA_HOME)}
```

system-property

Returns the value of a system property.

```
${system-property(os.name)}
```

timestamp

Can be used to format the current date and time. Accepts a string (as a parameter) that determines how the date and time will be formatted (format string or *picture string* as it is known in the XSLT specification). The format string must comply with the rules of the XSLT `format-dateTime` function specification.

```
${timestamp([h1]:[m01] [P] [M01]/[D01]/[Y0001])}
```

path

Returns the path associated with the specified path ID. The following paths IDs are supported:

- **oxygen-webhelp-output-dir** - The path to the output directory. The path is relative to the current HTML file.
- **oxygen-webhelp-assets-dir** - The path to the `oxygen-webhelp` subdirectory from the output directory. The path is relative to the current HTML file.
- **oxygen-webhelp-template-dir** - The path to the template directory. The path is relative to the current HTML file.

```
${path(oxygen-webhelp-template-dir)}
```

Note: New paths IDs can be added by overriding the `wh-macro-custom-path` template from `com.oxygenxml.webhelp.responsive\xsl\template\macroExpander.xsl`:

```
<!-- Extension template for expanding a custom path macro. -->
<xsl:template name="wh-macro-custom-path">
    <xsl:param name="pathId"/>
    <xsl:value-of select="$pathId"/>
</xsl:template>
```

map-xpath

Can be used to execute an XPath expression over the DITA map file from the temporary directory.

Tip: Available in all template layout HTML pages.

```
${map-xpath(/map/title)}
```

topic-xpath

Can be used to execute an XPath expression over the current topic.

Tip: Available only in the topic HTML page template (`wt_topic.html`).

```
${topic-xpath(string-join(//shortdesc//text(), ' '))}
```

oxygen-webhelp-build-number

Returns the current WebHelp distribution ID (build number).

```
${oxygen-webhelp-build-number}
```

Extensibility

To add new *macros*, you can add an XSLT extension to overwrite the `wh-macro-extension` template from the `com.oxygenxml.webhelp.responsive\xsl\template\macroExpander.xsl` file.

```
<!-- Extension template for expanding custom macro constructs -->
<xsl:template name="wh-macro-extension">
    <xsl:param name="name"/>
    <xsl:param name="params"/>
    <xsl:param name="contextNode"/>
    <xsl:param name="matchedString"/>

    <xsl:choose>
        <xsl:when test="$contextNode instance of attribute()">
            <xsl:value-of select="$matchedString"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:message>Cannot expand macro:
                [<xsl:value-of select="$matchedString"/>]</xsl:message>
            <xsl:copy-of select="$contextNode"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

The `wh-macro-extension` template has the following parameters:

- **name** - The name of the current *macro*.
- **params** - List of parameters of the current *macro* as a `string` sequence. The current *macros* parsing mechanism only allows *macros* with a maximum of one parameter. Consequently, this list will contain at most one element.
- **contextNode** - The current element or attribute where the *macro* was declared.
- **matchedString** - The entire value of the matched *macro* as specified in the HTML template page.

HTML Page Layout Files

The HTML page layout files define the default layout of the generated pages in the output for the built-in template. There are four types of pages (*main*, *search*, *topic*, *index*) and each type of page is a simple HTML file. Each page type has various components that appear by default and each component has a corresponding element and when that element is included in the HTML file, the corresponding components will appear in the output.

⚠ **Warning:** It is no longer recommended for you to customize these files because if you upgrade to a newer version of **Oxygen**, those files may no longer produce the desired results and if new components have been added, you won't have access to them.

If you do choose to customize these HTML files, each type of page is defined inside an `<html-page-layout-files>` element in the descriptor file.

```
<publishing-template>
      ...
      <webhelp>
      ...
      <!-- HTML page layout files -->
      <html-page-layout-files>
        <page-layout-file page="main" file="page-templates/wt_index.html"/>
        <page-layout-file page="search" file="page-templates/wt_search.html"/>
        <page-layout-file page="topic" file="page-templates/wt_topic.html"/>
        <page-layout-file page="index-terms" file="page-templates/wt_terms.html"/>
      </html-page-layout-files>
```

If you do use the **html-page-layout-files** element, you must specify all four types of pages (*main*, *search*, *topic*, *index*). When not specified, the files from the *DITA-OT-DIR*`/plugins/com.oxygenxml.webhelp.responsive/oxygen-webhelp/page-templates` folder will be used to define the layout of each type of page.

HTML Page Components

Each type of page contains various components that control the layout of that page. The rendering of each component depends on the context where it is placed and its content depends on the transformed *DITA map (on page 1324)*.

Some of the components can be used in all four types of pages, while some are only available for certain pages. For instance, the *Publication Title* component can be used in all pages, but the *Navigation Breadcrumb* component can only be used in the **Topic Page**.

To include a component in the output of a particular type of page, you have to reference a specific element in that particular HTML file. All the elements associated with a component should belong to the `http://www.oxygenxml.com/webhelp/components` namespace.

Every component can contain custom content or reference another component. To specify where the component content will be located in the output, you can use the `<whc:component_content>` element as a

descendant of the component element. It can specify the location as before, after, or it can wrap the component content. The following snippet contains an example of each:

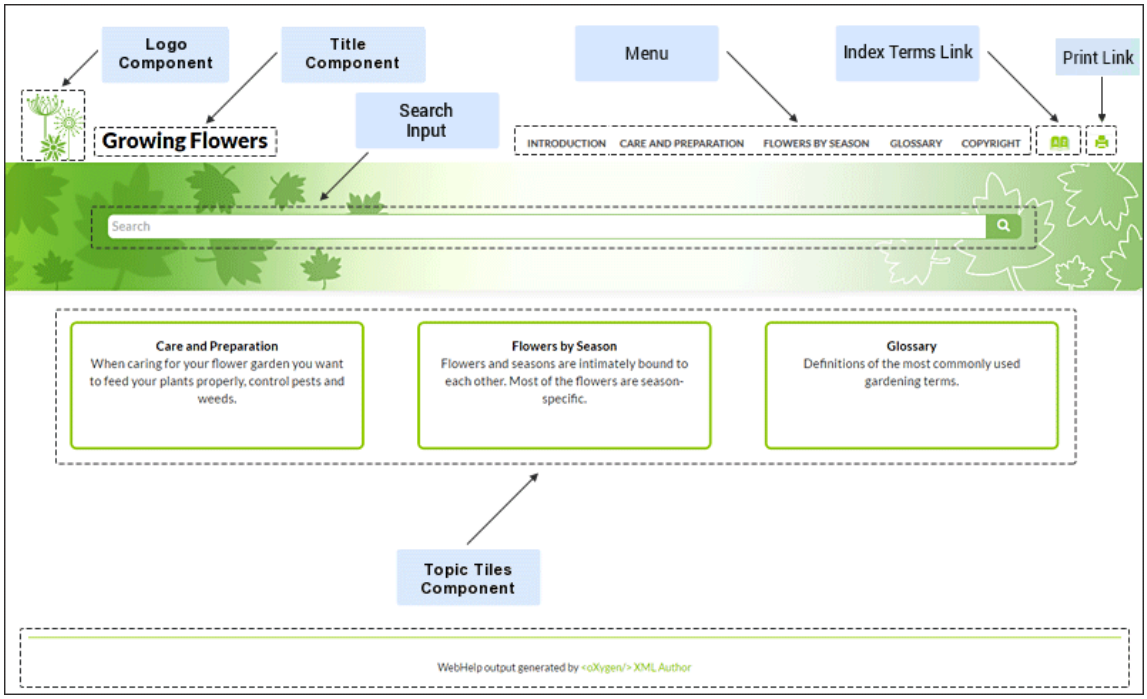
```
<whc:webhelp_search_input class="navbar-form wh_main_page_search"
       role="form" >
       <div class="custom-content-before">Enter search terms here:</div>
 <div class="custom-wrapper">
         <whc:component_content/>
 </div>
       <div class="custom-content-after">Results will be displayed in a new window.</div>
</whc:webhelp_search_input>
```

Main Page

The *Main Page* is the home page generated in the WebHelp Responsive output. The name of the HTML file that defines this page is `wt_index.html` and it is located in the following directory: *DITA-OT-DIR*`/plugins/com.oxygenxml.webhelp.responsive/oxygen-webhelp/page-templates`.

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.

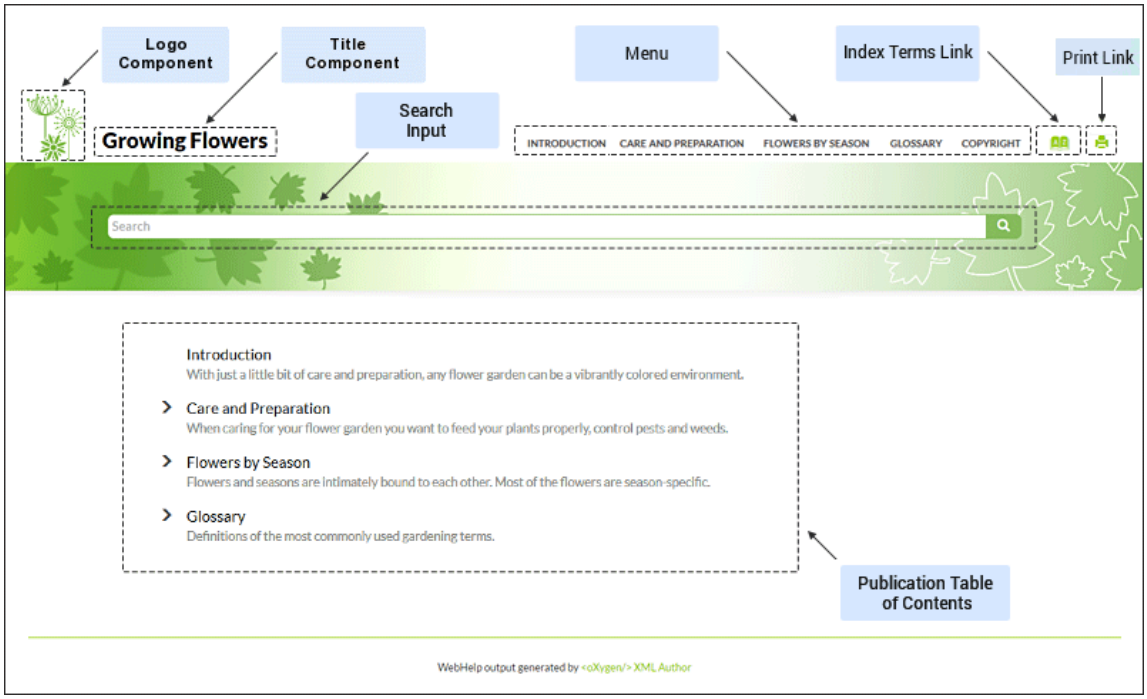
Figure 297. Examples of Main Page Components for a Tiles Style of Layout



1. Publication Logo *(on page 833)*
2. Publication Title *(on page 833)*
3. Search Input *(on page 834)*
4. Main Menu *(on page 834)*
5. Index Terms Link *(on page 835)*

6. Topic Tiles *(on page 834)*
7. Print Link *(on page 834)*

Figure 298. Examples of Main Page Components for a Tree Style of Layout



1. Publication Logo *(on page 833)*
2. Publication Title *(on page 833)*
3. Search Input *(on page 834)*
4. Main Menu *(on page 834)*
5. Index Terms Link *(on page 835)*
6. Table of Contents *(on page 835)*
7. Print Link *(on page 834)*

The following components can be referenced in the *Main Page* (`wt_index.html`) file:

Publication Title (`webhelp_publication_title`)

This component generates the publication title in the output. To generate this component, the `<whc:webhelp_publication_title>` element must be specified in the HTML file 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 `<whc:webhelp_logo>` element must be specified in the HTML file 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.

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 `<whc:webhelp_search_input>` element must be specified in the HTML file 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: `wh_search_input`.

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 `<whc:webhelp_print_link>` element must be specified in the HTML file 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`.

Main Menu (`webhelp_top_menu`)

This component generates a menu with all the documentation topics. To generate this component, the `<whc:webhelp_top_menu>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_top_menu
    xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_top_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 How to Customize the Menu *(on page 865)*.

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 correspond to the topic title, short description, and image. To generate this component, the `<whc:webhelp_tiles>` element must be specified in the HTML file 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="wh_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 How to Configure the Tiles on the WebHelp Responsive Main Page *(on page 870)*.

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 `<whc:webhelp_main_page_toc>` element must be specified in the HTML file 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`.

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 contain any index terms, then the link will not be generated. To generate this component, the `<whc:webhelp_indexterms_link>` element must be specified in the HTML file 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 Skins 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 `<whc:webhelp_skin_resources>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_skin_resources/>
```

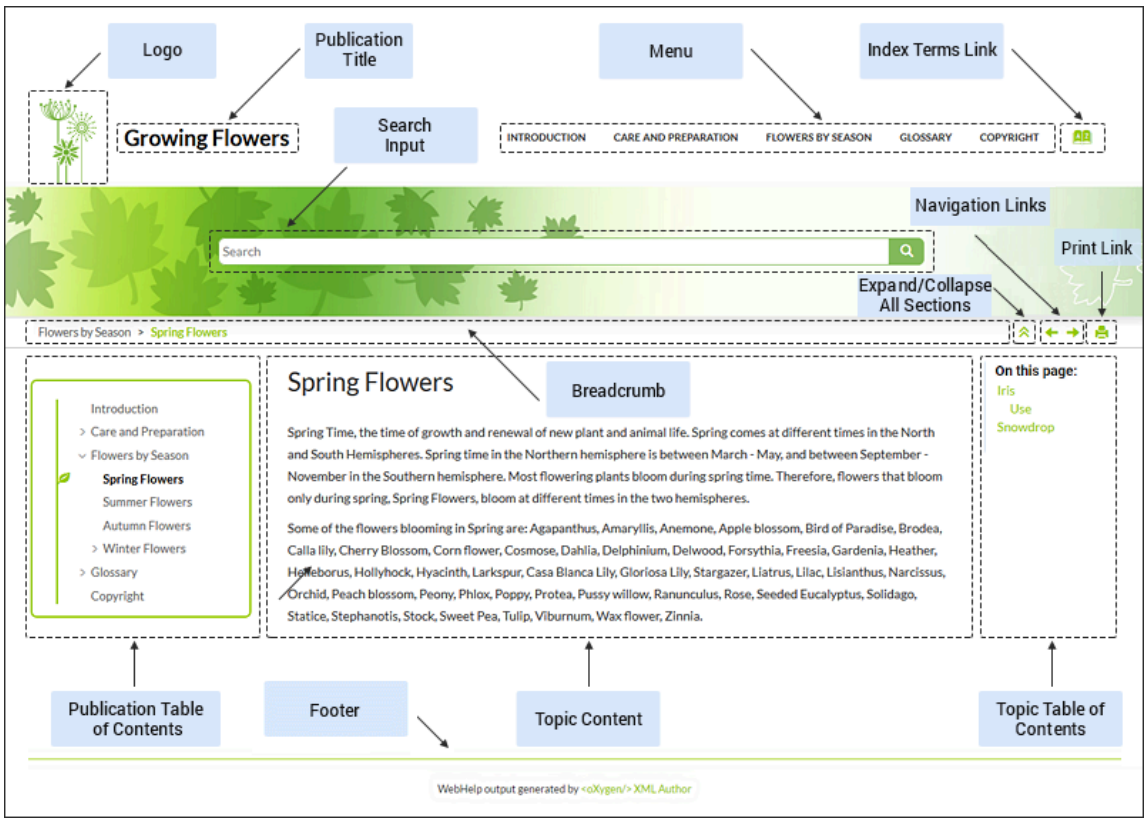
In the output, you will find a link to the skin resources.

Topic Page

The *Topic Page* is the page generated for each DITA topic in the WebHelp Responsive output. The name of the HTML file that defines this page is `wt_topic.html` and it is located in the following directory: *DITA-OT-DIR*`/plugins/com.oxygenxml.webhelp.responsive/oxygen-webhelp/page-templates`.

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 299. Examples of Topic Page Components



1. Publication Logo *(on page 837)*
2. Publication Title *(on page 837)*
3. Search Input *(on page 837)*
4. Main Menu *(on page 839)*
5. Index Terms Link *(on page 839)*
6. Expand/Collapse All Sections *(on page 839)*
7. Navigation Links *(on page 838)*
8. Print Link *(on page 838)*
9. Breadcrumb *(on page 837)*

10. Publication Table of Contents *(on page 838)*
11. Topic Content *(on page 838)*
12. Topic Table of Contents *(on page 838)*

The following components can be referenced in the *Topic Page* (`wt_topic.html`) file:

Publication Title (`webhelp_publication_title`)

This component generates the publication title in the output. To generate this component, the `<whc:webhelp_publication_title>` element must be specified in the HTML file 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 `<whc:webhelp_logo>` element must be specified in the HTML file 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.

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 `<whc:webhelp_search_input>` element must be specified in the HTML file 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: `wh_search_input`.

Topic Breadcrumb (`webhelp_breadcrumb`)

This component generates a breadcrumb that displays the path of the current topic. To generate this component, the `<whc:webhelp_breadcrumb>` element must be specified in the HTML file 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.

Navigation Links (`webhelp_navigation_links`)

This component generates navigation links to the next and previous topics. To generate this component, the `<whc:webhelp_navigation_links>` element must be specified in the HTML file 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.

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 `<whc:webhelp_print_link>` element must be specified in the HTML file 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`.

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 `<whc:webhelp_topic_content>` element must be specified in the HTML file 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`.

Publication TOC (`webhelp_publication_toc`)

This component generates a mini table of contents for the current topic (on the left side). It will contain links to the children of the current topic, its siblings, and all of its ancestors. To generate this component, the `<whc:webhelp_publication_toc>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_publication_toc
    xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_publication_toc`. This element will contain links to the topics that are close to the current topic.

Topic TOC (`webhelp_topic_toc`)

This component generates a topic table of contents for the current topic (on the right side) with a heading named **On this page**. It contains links to each section within the current topic and

the section corresponding to the current scroll position is highlighted. The topic must contain at least two `<section>` elements and each `<section>` must have an `@id` attribute. To generate this component, the `<whc:webhelp_topic_toc>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_topic_toc
    xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_topic_toc`. This element will contain links to the topics that are close to the current topic.

Expand/Collapse Sections (`webhelp_expand_collapse_sections`)

This component is used to generate an icon that expands or collapses sections listed in the side table of contents within a topic. To generate this component, the `<whc:webhelp_expand_collapse_sections>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_expand_collapse_sections
    xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `webhelp_expand_collapse_sections`.

Topic Feedback (`webhelp_feedback`)

This component generates a placeholder for where the comments section will be presented. To generate this component, the `<whc:webhelp_feedback>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_feedback
    xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

Main Menu (`webhelp_top_menu`)

This component generates a menu with all the documentation topics. To generate this component, the `<whc:webhelp_top_menu>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_top_menu
    xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_top_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 How to Customize the Menu *(on page 865)*.

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 contain any index terms, then the link will not be generated.

To generate this component, the `<whc:webhelp_indexterms_link>` element must be specified in the HTML file 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.

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 `<whc:webhelp_child_links>` element must be specified in the HTML file 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 `<whc:webhelp_related_links>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_related_links
     xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

Link to Skins 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 `<whc:webhelp_skin_resources>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_skin_resources/>
```

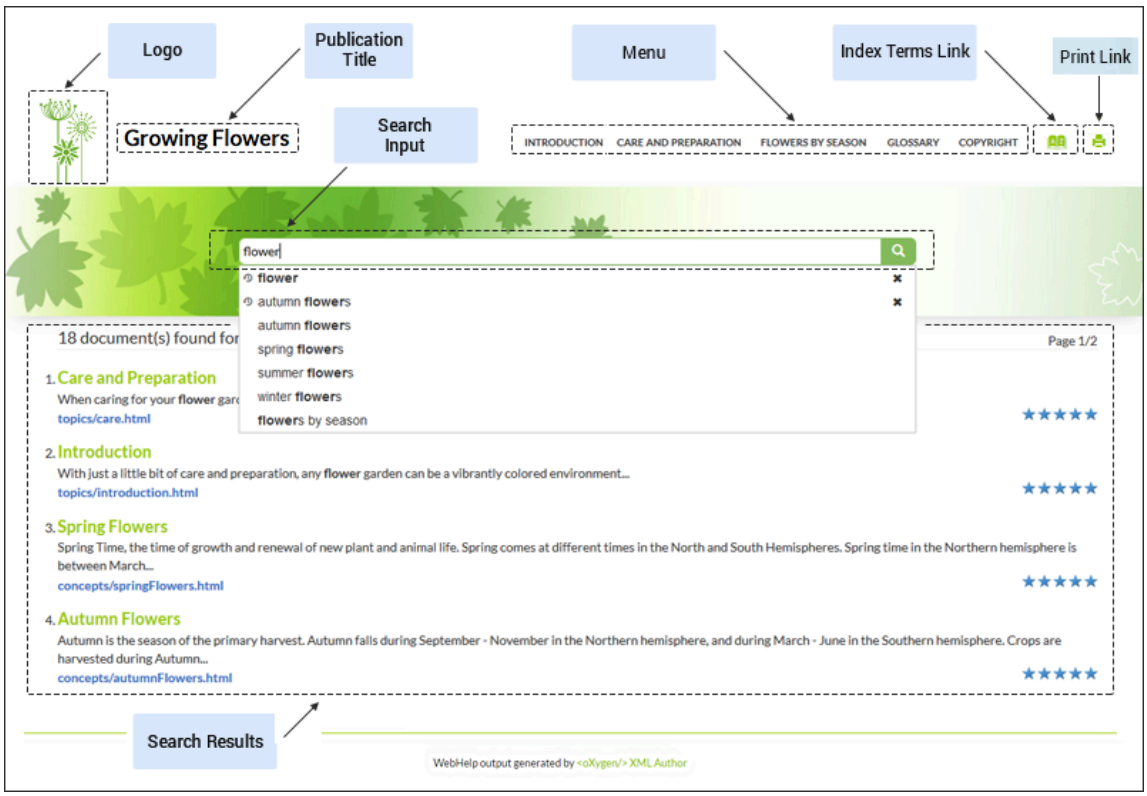
In the output, you will find a link to the skin resources.

Search Results Page

The *Search Results Page* is the page generated that presents search results in the WebHelp Responsive output. The name of the HTML file that defines this page is `wt_search.html` and it is located in the following directory: *`DITA-OT-DIR`*`/plugins/com.oxygenxml.webhelp.responsive/oxygen-webhelp/page-templates`.

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 300. Examples of Search Results Page Components



1. Publication Logo *(on page 841)*
2. Publication Title *(on page 841)*
3. Search Input *(on page 842)*
4. Main Menu *(on page 842)*
5. Index Terms Link *(on page 843)*
6. Search Results *(on page 842)*
7. Print Link *(on page 842)*

The following components can be referenced in the *Search Results Page* (`wt_search.html`) file:

Publication Title (`webhelp_publication_title`)

This component generates the publication title in the output. To generate this component, the `<whc:webhelp_publication_title>` element must be specified in the HTML file 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 `<whc:webhelp_logo>` element must be specified in the HTML file 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.

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 `<whc:webhelp_search_input>` element must be specified in the HTML file 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: `wh_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 `<whc:webhelp_search_results>` element must be specified in the HTML file 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`.

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 `<whc:webhelp_print_link>` element must be specified in the HTML file 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`.

Main Menu (`webhelp_top_menu`)

This component generates a menu with all the documentation topics. To generate this component, the `<whc:webhelp_top_menu>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_top_menu
    xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_top_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 How to Customize the Menu *(on page 865)*.

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 contain any index terms, then the link will not be generated. To generate this component, the `<whc:webhelp_indexterms_link>` element must be specified in the HTML file 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 Skins 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 `<whc:webhelp_skin_resources>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_skin_resources/>
```

In the output, you will find a link to the skin resources.

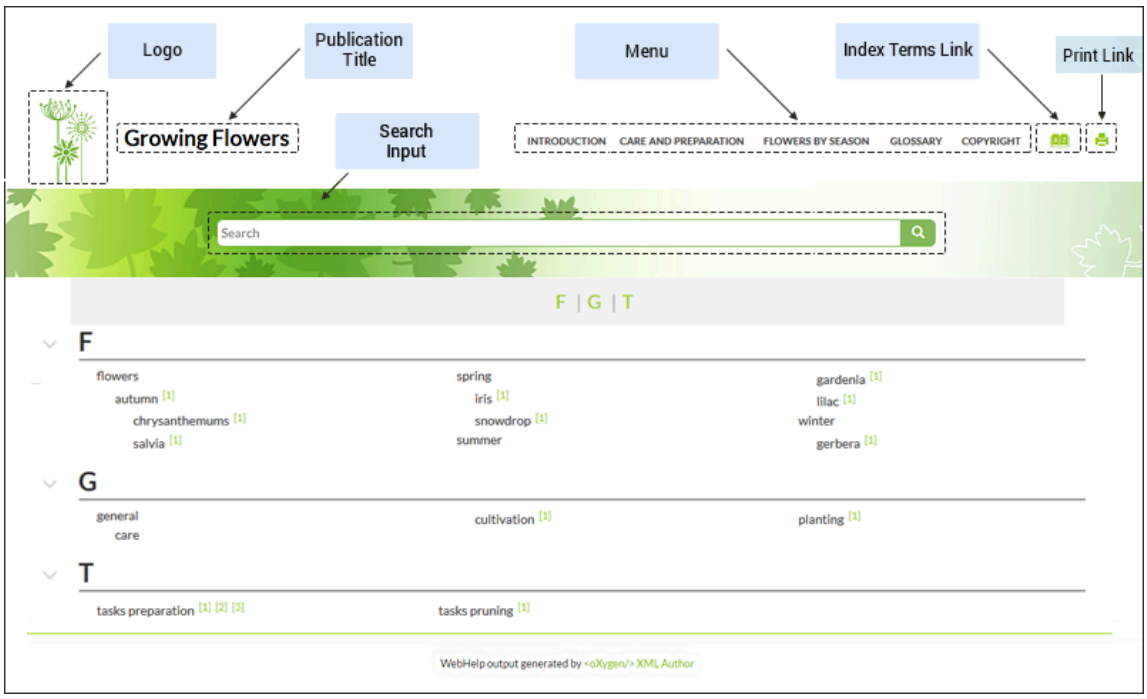
Index Terms Page

The *Index Terms Page* is the page generated that presents index terms in the WebHelp Responsive output. The name of the HTML file that defines this page is `wt_terms.html` and it is located in the following directory: *`DITA-OT-DIR`*`/plugins/com.oxygenxml.webhelp.responsive/oxygen-webhelp/page-templates`.

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.

An alphabet that contains the first letter of the documentation index terms is generated at the top of the index page. Each letter represents a link to a specific indices section.

Figure 301. Example of Index Terms Page Components



1. Publication Logo *(on page 844)*
2. Publication Title *(on page 844)*
3. Search Input *(on page 845)*
4. Main Menu *(on page 845)*
5. Index Terms Link (webhelp_indexterms_link) *(on page 845)*
6. Print Link *(on page 845)*

The following components can be referenced in the *Index Terms Page* (`wt_terms.html`) file:

Publication Title (`webhelp_publication_title`)

This component generates the publication title in the output. To generate this component, the `<whc:webhelp_publication_title>` element must be specified in the HTML file 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 `<whc:webhelp_logo>` element must be specified in the HTML file 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.

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 `<whc:webhelp_search_input>` element must be specified in the HTML file 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: `wh_search_input`.

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 `<whc:webhelp_print_link>` element must be specified in the HTML file 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`.

Main Menu (`webhelp_top_menu`)

This component generates a menu with all the documentation topics. To generate this component, the `<whc:webhelp_top_menu>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_top_menu
    xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>
```

In the output, you will find an element with the class: `wh_top_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 How to Customize the Menu *(on page 865)*.

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 contain any index terms, then the link will not be generated. To generate this component, the `<whc:webhelp_indexterms_link>` element must be specified in the HTML file 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 Skins 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 `<whc:webhelp_skin_resources>` element must be specified in the HTML file as in the following example:

```
<whc:webhelp_skin_resources/>
```

In the output, you will find a link to the skin resources.

Combining WebHelp Responsive and PDF Customizations in a Template Package

An *Oxygen Publishing Template* package can contain both a WebHelp Responsive and PDF customization in the same template package and you can use that same template in both types of transformations. The template descriptor file can define the customization for both types by including both a `<webhelp>` and `<pdf>` element and some of the resources can be reused. Resources referenced in elements in the `<webhelp>` element will only be used for WebHelp transformations, and resources referenced in the elements in the `<pdf>` element will only be used in PDF transformations.

```
<publishing-template>
    <name>Flowers</name>
    <description>Flowers themed light-colored template</description>
    <webhelp>
        <tags>
            <tag>purple</tag>
            <tag>light</tag>
        </tags>
        <preview-image file="flowers-preview.png"/>
        <resources>
            <css file="flowers-wh.css"/>
            <css file="flowers-page-styling.css"/>
        </resources>
        <parameters>
            <parameter name="webhelp.show.main.page.tiles" value="no"/>
            <parameter name="webhelp.show.main.page.toc" value="yes"/>
        </parameters>
    </webhelp>
    <pdf>
        <tags>
            <tag>purple</tag>
            <tag>light</tag>
        </tags>
```

```
        <preview-image file="flowers-preview.png"/>
        <resources>
            <css file="flowers-pdf.css"/>
            <css file="flowers-page-styling.css"/>
        </resources>
        <parameters>
            <parameter name="show.changes.and.comments" value="yes"/>"/>
        </parameters>
    <pdf>
</publishing-template>
```

Related Information:

Publishing Template Package Contents for PDF Customizations *(on page 963)*

Generating WebHelp Responsive Output

The publishing process can be initiated from a transformation scenario within **Oxygen XML Editor/Author**, from a command line outside **Oxygen XML Editor/Author**, or from an integration server.

Running from Oxygen XML Editor/Author

To publish a *DITA map (on page 1324)* as **WebHelp Responsive** output, follow these steps:

1. Select the **Configure Transformation Scenario(s)** action from the 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 built-in skins that you can use for the layout of your WebHelp system output.
   - **Parameters Tab** - This tab includes numerous transformation parameters that can be set to customize your WebHelp system output.
   - **Feedback Tab** - This tab is for those who want to add the **Oxygen Feedback** comments component at the bottom of each WebHelp page so that you can interact with your readers.
   - **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.

Result: When the **DITA Map WebHelp Responsive** transformation is complete, the output is automatically opened in your default browser.

Automating the WebHelp Responsive Output for DITA

DITA-based WebHelp output can be generated from an automated publishing process using a command line outside of **Oxygen XML Editor/Author** or an automatic publishing system, such as *Jenkins* or *Travis*. **However, to do this, you must purchase an additional Oxygen XML WebHelp license**.

Related Information:

Generating WebHelp Responsive Output for DITA

Customizing WebHelp Responsive Output

Oxygen XML Developer Eclipse plugin provides support for customizing the **WebHelp Responsive** output to suit your specific needs. The **WebHelp Responsive** output is based upon the *Bootstrap* responsive front-end framework and is available for DITA document types.

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 layout pages, 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.

Working with Publishing Templates

An *Oxygen Publishing Template (on page 1327)* defines all aspects of the layout and styles of the **WebHelp Responsive** output. It is a self-contained customization package stored as a ZIP archive or folder that can easily be shared with others. It provides the primary method for customizing the output. The recommended method for customizing the *WebHelp Responsive* output is to use a custom publishing template.

This section contains topics about how to create, edit, publish, and share publishing templates.

Related Information:

Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

How to Create a Publishing Template

To create a customization, you can start from scratch or from an existing template, and then adapt it according to your needs.

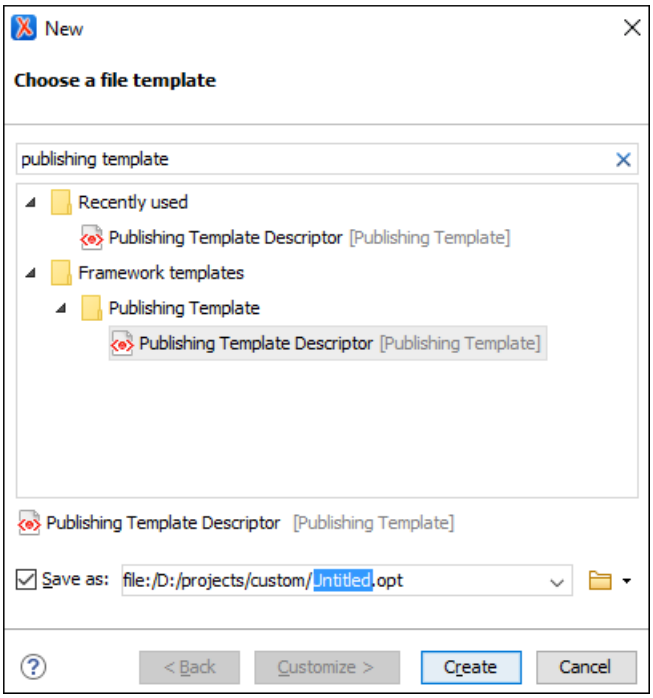
Creating a Template Starting from Scratch

To create a new *Oxygen Publishing Template*, follow these steps:

1. Create a folder that will contain all the template files.

2. In **Oxygen XML Editor/Author**, open the new document wizard (use **File > New** or the **New** toolbar button), then choose the **Publishing Template Descriptor** template.

Figure 302. Choosing the Publishing Template Descriptor Document Template



3. Save the `.opt` file into your customization directory.
4. Open the `.opt` file in the editor and customize it to suit your needs.

Creating a Template Starting from an Existing Template

If you are using a **WebHelp Responsive** or **DITA Map PDF - based on HTML5 & CSS** transformation, the easiest way to create a new *Oxygen Publishing Template (on page 1327)* is to select an existing template in the transformation scenario dialog box and use the **Save template as** button to save that template into a new template package that can be used as a starting point.

To create a new *Oxygen Publishing Template*, follow these steps:

1. Open the transformation scenario dialog box and select the publishing template you want to export and use as a starting point.
2. **Optional:** You can set one or more transformation parameters from the **Parameters** tab and the edited parameters will be exported along with the selected template. You will see which parameters will be exported in the dialog box that is displayed after the next step.
3. Click the **Save template as** button.

   Step Result: This opens a template package configuration dialog box that contains some options and displays the parameters that will be exported to your template package.
4. Specify a name for the new template.
5. **Optional:** Specify a template description.
6. **Optional:** The same publishing template package can contain both a WebHelp Responsive and PDF customization and you can use the same template in both types of transformations (**DITA**

Map WebHelp Responsive or **DITA Map to PDF - based on HTML5 & CSS**). You can use the **Include WebHelp customization** and **Include PDF customization** options to specify whether your custom template will include both types of customizations.

7. **Optional:** For **WebHelp Responsive** customizations, you can select the **Include HTML Page Layout Files** option if you want to copy the default *HTML Page Layout Files (on page 830)* in your template package. They are helpful if you want to change the structure of the generated HTML pages.
8. In the **Save as** field, specify the name and path of the ZIP file where the template will be saved.

   Step Result: A new ZIP archive will be created on disk in the specified location with the specified name.
9. Open the `.opt` file in the editor and customize it to suit your needs.

For more information about creating and customizing publishing templates, watch our video demonstration:

https://www.youtube.com/embed/zNmXfKWXwO8

Related Information:

- Publishing Template Package Contents for PDF Customizations *(on page 963)*
- Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

How to Edit a Packed Publishing Template

To edit an existing *Oxygen Publishing Template (on page 1327)* package, follow these steps:

1. Unzip the ZIP archive associated with the *Oxygen Publishing Template* in a separate folder.
2. Link the folder associated with the template in the **Navigator** view.
3. Using the **Navigator** view, you can modify the resources (CSS, JS, fonts) within the *Oxygen Publishing Template* folder to fit your needs.
4. Open the publishing template descriptor file (`.opt` extension) in the editor and modify it to suit your needs.
5. **Optional:** Once you finish your customization, you can archive the folder as a ZIP file.

Related Information:

- Publishing Template Package Contents for PDF Customizations *(on page 963)*
- Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

How to Add a Publishing Template to the Publishing Templates Gallery

To add the publishing template to your templates gallery, follow these steps:

1. Open the transformation scenario dialog box by editing a WebHelp Responsive transformation.
2. In the **Templates** tab, click the **Configure Publishing Templates Gallery** link to.
   This will open the preferences page.
3. Click the **Add** button and specify the location of your template directory.

Your template directory is now added to the **Additional Publishing Templates Galleries** list.

4. Click **OK** to return to the transformation scenario dialog box.
   All the templates contained in your template directory will be displayed in the preview pane along with all the built-in templates.

How to Use a Publishing Template from a Command Line

Before you run the transformation, you need to know if the publishing template has a single or multiple template descriptor files *(on page 819)*. If you don't know, open the ZIP archive or folder and check for files with the `.opt` extension.

Using a Publishing Template with a Single Descriptor

A template with a single descriptor is used for a single customization.

To run from a command line, you need to use the webhelp.publishing.template parameter *(on page 906)*. This parameter specifies the path to the ZIP archive (or root folder) that contains your custom WebHelp Responsive template.

Command-Line Example:

- **Windows**:

```
dita.bat
   --format=webhelp-responsive
   --input=c:\path\to\mySample.ditamap
   --output=c:\path\to\output
   -Dwebhelp.publishing.template=custom-template
```

- **Linux/Mac OS X**:

```
dita
   --format=webhelp-responsive
   --input=/path/to/mySample.ditamap
   --output=/path/to/output
   -Dwebhelp.publishing.template=custom-template
```

Tip: You can also start the `dita` process by passing it a DITA OT Project File. Inside the project file you can specify as parameters for the `webhelp-responsive` transformation type the WebHelp-related parameters.

Using a Publishing Template with Multiple Descriptors

A template with multiple descriptors contains multiple customizations.

Because the publishing template is self-contained, it is used to reuse resources that are common to multiple publications.

To run from a command line, you need to use the webhelp.publishing.template *(on page 906)* and webhelp.publishing.template.descriptor *(on page 906)* parameters.

The webhelp.publishing.template *(on page 906)* parameter specifies the path to the ZIP archive (or root folder) while the webhelp.publishing.template.descriptor *(on page 906)* parameter specifies the name of the descriptor you want to use.

Command-Line Example:

- **Windows**:

```
dita.bat
   --format=webhelp-responsive
   --input=c:\path\to\mySample.ditamap
   --output=c:\path\to\output
   -Dwebhelp.publishing.template=custom-template
   -Dwebhelp.publishing.template.descriptor=flowers.opt
```

- **Linux/Mac OS X**:

```
dita
   --format=webhelp-responsive
   --input=/path/to/mySample.ditamap
   --output=/path/to/output
   -Dwebhelp.publishing.template=custom-template
   -Dwebhelp.publishing.template.descriptor=flowers.opt
```

Tip: You can also start the `dita` process by passing it a DITA OT Project File. Inside the project file you can specify as parameters for the `webhelp-responsive` transformation type the WebHelp-related parameters.

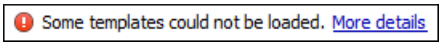
How to Share a Publishing Template

To share a publishing template with others, following these steps:

1. Copy your template in a new folder in your project.
2. Go to **Options > Preferences > DITA > Publishing** and add that new folder to the list.
3. Switch the option as the bottom of that preferences page to **Project Options**.
4. Share your project file (`.xpr`).

Troubleshooting: Errors Encountered when Loading Templates

When the **Templates** tab of a **WebHelp Responsive** transformation scenario dialog box is opened, all templates (built-in and custom) are loaded and validated. Specifically, certain elements in the template descriptor file *(on page 819)* are checked for validity. If errors are encountered that prevents the template from loading, the following message will be displayed toward the bottom of the dialog box:



If you click the **More details** link, a window will open with more information about the encountered error. For example, it might offer a hint that the element is missing from the expected descriptor file structure *(on page 819)*.

Also, if a template could be loaded, but certain elements could not be found in the descriptor file *(on page 819)*, a warning icon (⚠) will be displayed on the template's image (in the **Templates** tab of the transformation dialog box). For example, this happens if a valid preview-image element *(on page 822)* cannot be found.

Converting Old Templates to Newer Versions

WebHelp templates that were created in older versions of Oxygen XML Developer Eclipse plugin can be converted to the Publishing Template format that was introduced in Oxygen XML Developer Eclipse plugin version 20.0. This section contains several procedures for converting old templates depending on the version they were created in.

Convert Version 21 Publishing Templates to Version 22

If you have a custom Publishing Template that was created in Oxygen XML Developer Eclipse plugin version 21.0 or 21.1, the following conversion procedure is required for the template to be compatible with Oxygen XML Developer Eclipse plugin version 22:

1. In the **Navigator** view, add the root directory for your custom Publishing Template (you can use a linked folder *(on page 192)* and the easiest way to do this is to drag and drop the folder).

   Note: If your template is stored as a ZIP archive, you first need to unzip it.
2. Expand your template directory, right-click the `page-templates` subfolder, and select **Refactoring > XML Refactoring**.
3. In the **XML Refactoring** dialog box, scroll to the **Publishing Template** section and select **Migrate HTML Page Layout Files to v22**, then click **Next**.
4. The **Scope** should be left as **Selected project resources**.
5. You can use the **Preview** button to open a comparison panel where you can review all the changes that will be made by the refactoring operation before applying the changes.
6. Click **Finish** to perform the conversion.

Result: The converted Publishing Template can now be used in version 22.

Related Information:

- Convert Version 20 Publishing Templates to Version 21 *(on page 853)*
- Convert Version 19 (and Older) Publishing Templates to Version 20 *(on page 854)*

Convert Version 20 Publishing Templates to Version 21

If you have a custom Publishing Template that was created in Oxygen XML Developer Eclipse plugin version 20.0 or 20.1, the following conversion procedure is required for the template to be compatible with Oxygen XML Developer Eclipse plugin version 21.0 or 21.1:

1. In the **Navigator** view, add the root directory for your custom Publishing Template (you can use a linked folder *(on page 192)* and the easiest way to do this is to drag and drop the folder).

   Note: If your template is stored as a ZIP archive, you first need to unzip it.
2. Expand your template directory, right-click the `page-templates` subfolder, and select **Refactoring > XML Refactoring**.
3. In the **XML Refactoring** dialog box, scroll to the **Publishing Template** section and select **Migrate HTML Page Layout Files to v21**, then click **Next**.
4. The **Scope** should be left as **Selected project resources**.
5. You can use the **Preview** button to open a comparison panel where you can review all the changes that will be made by the refactoring operation before applying the changes.
6. Click **Finish** to perform the conversion.

Result: The converted Publishing Template can now be used in version 21.0 or 21.1.

Related Information:

Convert Version 21 Publishing Templates to Version 22 *(on page 853)*

Convert Version 19 (and Older) Publishing Templates to Version 20 *(on page 854)*

Convert Version 19 (and Older) Publishing Templates to Version 20

With the introduction of the Publishing Template concept in Oxygen XML Developer Eclipse plugin version 20.0, the old WebHelp output template formats (version 19.1 and older) are no longer supported. However, they can be easily converted to the new format (version 20.0 or 20.1) by following this procedure:

1. Set Oxygen XML Developer Eclipse plugin to point to the DITA-OT distribution that contains your old template:
   a. Go to **Options > Preferences > DITA**.
   b. Select **Custom** in the **DITA Open Toolkit** section and specify the DITA-OT directory that contains your old template.
2. Edit a **WebHelp Responsive** transformation scenario.
3. Select your old custom template in the **Templates** tab.
4. Click on the **Save Template as** button, complete the required fields, and save the template.
5. Reset the option set in step 1 to its previous value.
6. Edit the **WebHelp Responsive** transformation scenario again.
7. This time, use the **Choose custom template** button to select your converted template.
8. Save the scenario and use it to generate the WebHelp Responsive output.

Result: The converted template is now in a Publishing Template format for version 20.

Related Information:

Convert Version 20 Publishing Templates to Version 21 *(on page 853)*

Convert Version 19 (and Older) Publishing Templates to Version 21

If you have a custom template that was created in Oxygen XML Developer Eclipse plugin version 19.1 or older and you want to convert it to be compatible with Oxygen XML Developer Eclipse plugin version 21.0 or 21.1, you need to apply two conversion procedures:

1. Convert the old template to a version 20 Publishing Template by using this procedure: Convert Version 19 (and Older) Publishing Templates to Version 20 *(on page 854)*.
2. Convert that version 20 Publishing Template to version 21 by using this procedure: Convert Version 20 Publishing Templates to Version 21 *(on page 853)*.

Result: The converted Publishing Template can now be used in version 21.0 or 21.1.

Changing the Layout and Styles

This section contains topics that explain how to customize the output using CSS, inserting HTML fragments, changing the layout of the main page, and more.

How to Use CSS Styling to Customize the Output

The most common way to customize WebHelp Responsive 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 (using an *Oxygen Publishing Template (on page 1327)* or a transformation parameter). This custom file will be the final CSS to be applied so its content will override the styles in the other pre-existing CSS files.

Using CSS Inspector to Identify Content for Custom CSS File

You can use your browser's CSS inspector to identify the pertinent code in the current CSS files and you can even make changes directly in the CSS inspector to test the results so that you know exactly what content to use in your custom CSS file.

In most popular browsers (such as Chrome, Firefox, and Edge), you can access the CSS inspector by using **F12** or by selecting **Inspect Element** (or simply **Inspect**) from the contextual menu.

Tip: When using Safari on Mac OS X, you must first enable the Develop menu by going to the Advanced settings and selecting **Show Develop menu in menu bar**. Then you can select **Show Web Inspector** from the Develop menu or click **Command + Option + I**.

Create the Custom CSS

As a practical example, the following procedure would change the background color of the footer bar in the WebHelp output:

1. Use the browser's CSS inspector to identify the current CSS code that styles the footer bar. In this particular case, the pertinent code that would be identified is:

```
.wh_footer {
  font-size: 15px;
  line-height: 1.7em;
  background-color: #000;
}
```

2. If you want to test the color you want to apply as the background of this particular element, use the browser's CSS inspector to change the value of the `background-color` attribute. After you find a suitable color, copy that new code.
3. Create a custom CSS file and paste or enter the copied code. For example:

```
.wh_footer {
  background-color: #255890;
}
```

4. Save the custom CSS file at a location of your convenience.
5. Reference the CSS file in a *WebHelp Responsive* transformation using an *Oxygen Publishing Template (on page 856)* or the `args.css` parameter *(on page 857)*.

Referencing the CSS Using a Publishing Template

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Using the **Navigator** view, copy your custom CSS in a folder inside the publishing template root folder (for example, in the `custom_footer_template/resources` folder).
3. Open the template descriptor file *(on page 819)* associated with your publishing template and add your custom CSS in the *resources* section.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <resources>
            ...
            <css file="resources/MyCustom.css"/>
```

4. Open the *DITA Map WebHelp Responsive* transformation scenario.
5. Click the **Choose Custom Publishing Template** link and select your template.
6. Click **OK** to save the changes to the transformation scenario.
7. Run the transformation scenario.

Result: Your custom CSS will be applied as a final layer on top of any existing CSS rules and the output will reflect the changes you made.

Referencing the CSS Using the args.css Parameter

1. Edit the *DITA Map WebHelp Responsive* transformation scenario and open the **Parameters** tab.
2. Set the `args.css` parameter to the path of your custom CSS file.
3. Set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.
4. Click **OK** to save the changes to the transformation scenario.
5. Run the transformation scenario.

Result: Your custom CSS will be applied as a final layer on top of any existing CSS rules and the output will reflect the changes you made.

How to Insert HTML Content

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 (either from an *Oxygen Publishing Template (on page 1327)* or using one of the HTML fragment parameters *(on page 907)*). 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.

The XML File

There are several things to consider regarding this XML file:

- **Well-Formedness** - If the file is not a *Well-formed XML* document *(on page 277)* (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.
- **Referencing Resources in the XML File** - You can include references to local resources (such as JavaScript or CSS files) by using the built-in `${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 *(on page 816)*, you can use the `${oxygen-webhelp-template-dir}` macro.

To copy the referenced resources to the output directory, follow the procedure in: How to Copy Additional Resources to Output Directory *(on page 872)*.

- **Inline JavaScript or CSS Content**:

  JavaScript:

```
<script type="text/javascript">
    /* Include JavaScript code here. */

    function myFunction() {
      return true;
    }
</script>
```

  CSS:

```
<style>
    /* Include CSS style rules here. */
    *{
      color:red
    }
</style>
```

  Note: If you have special characters (for example, &, <) that break the well-formedness of the XML fragment, it is important to place this content inside an XML comment:

```
<script type="text/javascript">
  <!--
    /* Include JavaScript code here. */

    function myFunction() {
      return true;
    }
  -->
</script>
```

Using WebHelp Macros Inside an HTML Fragment File

The XML file can use WebHelp macros, which are variables that will be expanded when the content of the HTML fragment file will be copied in the final output.

There are two possibilities for using macros:

- **Directly in attribute values** - For example, if you want to reference a JavaScript file from the Publishing Template directory, you can use the following construct:

```
<script type="text/javascript" src="${path(oxygen-webhelp-template-dir)}/"></script>
```

- **In text content** - Using the `<whc:macro>` template component:

```
<script type="text/javascript" xmlns:whc="http://www.oxygenxml.com/webhelp/components">
  var outDirPath = '<whc:macro value="${path(oxygen-webhelp-output-dir)}"/>';
  console.log("The output directory path is:", outDirPath);
</script>
```

The following *macros* are supported:

i18n

For localizing a string.

```
${i18n(string.id)}
```

param

Returns the value of a transformation parameter.

```
${param(webhelp.show.main.page.tiles)}
```

env

Returns the value of an environment variable.

```
${env(JAVA_HOME)}
```

system-property

Returns the value of a system property.

```
${system-property(os.name)}
```

timestamp

Can be used to format the current date and time. Accepts a string (as a parameter) that determines how the date and time will be formatted (format string or *picture string* as it is known in the XSLT specification). The format string must comply with the rules of the XSLT `format-dateTime` function specification.

```
${timestamp([h1]:[m01] [P] [M01]/[D01]/[Y0001])}
```

path

Returns the path associated with the specified path ID. The following paths IDs are supported:

- **oxygen-webhelp-output-dir** - The path to the output directory. The path is relative to the current HTML file.
- **oxygen-webhelp-assets-dir** - The path to the `oxygen-webhelp` subdirectory from the output directory. The path is relative to the current HTML file.
- **oxygen-webhelp-template-dir** - The path to the template directory. The path is relative to the current HTML file.

```
${path(oxygen-webhelp-template-dir)}
```

Note: New paths IDs can be added by overriding the `wh-macro-custom-path` template from `com.oxygenxml.webhelp.responsive\xsl\template\macroExpander.xsl`:

```
<!-- Extension template for expanding a custom path macro. -->
<xsl:template name="wh-macro-custom-path">
    <xsl:param name="pathId"/>
    <xsl:value-of select="$pathId"/>
</xsl:template>
```

map-xpath

Can be used to execute an XPath expression over the DITA map file from the temporary directory.

Tip: Available in all template layout HTML pages.

```
${map-xpath(/map/title)}
```

topic-xpath

Can be used to execute an XPath expression over the current topic.

Tip: Available only in the topic HTML page template (`wt_topic.html`).

```
${topic-xpath(string-join(//shortdesc//text(), ' '))}
```

oxygen-webhelp-build-number

Returns the current WebHelp distribution ID (build number).

```
${oxygen-webhelp-build-number}
```

Referencing the HTML fragment using a Publishing Template

1. If you have not already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Insert the HTML content in a well-formed XML file (for example, `custom-html.html`).
3. Using the **Navigator** view, copy your custom HTML fragment in a folder inside publishing the template root folder (for example, in the `custom_footer_template/html-fragments` folder).
4. Open the template descriptor file *(on page 819)* associated with your publishing template and add a reference to the custom HTML fragment in the *html-fragments* section.

```
<publishing-template>
    ...
    <webhelp>
      ...
      <html-fragments>
        <fragment
          file="html-fragments/custom-html.html"
          placeholder="webhelp.fragment.head"/>
```

Note: If you want to insert the content in another location within the output document, you can reference the XML file from any other HTML Fragment extension points *(on page 825)*.

5. Open the *DITA Map WebHelp Responsive* transformation scenario.
6. Click the **Choose Custom Publishing Template** link and select your template.
7. Click **OK** to save the changes to the transformation scenario.
8. Run the transformation scenario.

Results: Your additional content will be included at the end of the `<head>` element of your output document.

Referencing the HTML Fragment using a Transformation Parameter

1. Insert the HTML content in a well-formed XML file.
2. Edit the *DITA Map WebHelp Responsive* transformation scenario and open the **Parameters** tab.
3. Edit the value of the `webhelp.fragment.head` parameter and set it to the absolute path of your XML file.

   Note: If you want to insert the content in another location within the output document, you can reference the XML file from any other HTML Fragment extension points *(on page 825)*.
4. Click **OK** to save the changes to the transformation scenario.
5. Run the transformation scenario.

Results: Your additional content will be included at the end of the `<head>` element of your output document.

Related Information:

HTML Fragment Extension Points *(on page 825)*

Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

How to Insert JavaScript AMD Modules

In the WebHelp Responsive output, you may want to include a JavaScript module that uses the Asynchronous Module Definition (AMD) format. Unlike the traditional JavaScript resources that are loaded using `<script>` tags, these modules are loaded using the *RequireJS* library. For the traditional JavaScript libraries, the standard procedure to contribute your script to the output would be to use an HTML fragment file as described here *(on page 857)*.

Although following the procedure that uses HTML fragments *(on page 857)* would result in having your JS file included in the output, loading the JS code in the browser will result in an error. Since your JS module uses the AMD API, it cannot be loaded using a `<script>` element. For example, because many jQuery plugins use the AMD format, it will be difficult for you to use those libraries in your custom WebHelp output.

Normally, a JavaScript AMD module can be loaded in one of the following ways:

- **As a top-level script**, using the `@data-main` attribute of the `<script>` element used to load the *RequireJS* library.

  ```
  <script data-main="js/template-main.js" src="js/require.js"></script>
  ```

  However, since WebHelp already loads its internal JS modules using *RequireJS*, a top-level script is already specified and you cannot specify another top-level script in the same page. Therefore, this

approach cannot be used to load your custom JS module in **Oxygen XML WebHelp**, leaving you with only the following option.

- **As a dependency module**, using a `require()` call from the top-level (main) script or from one of its dependency modules. This means that you would have to alter one of the WebHelp core JS libraries and inject a `require()` call to load your custom module:

```
require(['js/template-main.js']);
```

Note: Altering the WebHelp core libraries is not recommended because when you will upgrade the WebHelp plugin to a newer version, those modifications will be lost.

Contributing JavaScript AMD Modules Using a Publishing Template

Oxygen XML WebHelp provides the ability to contribute a custom JavaScript Asynchronous Module Definition (AMD) resource in the output by referencing it in the Publishing Template descriptor file *(on page 819)*. This module is automatically copied to the output directory and it is automatically loaded in the output HTML pages using a `require()` call. Thus, you can include your scripts in the output without altering WebHelp's core JavaScript libraries.

This module may contain all of your custom functionality or can be used to load other AMD JavaScript modules. The additional sub-modules can be stored either locally in your custom Publishing Template or on a remote web server.

Important: To enable loading of a JS module you should set the `webhelp.enable.template.js.module.loading` parameter to `yes` (the default value is `no`) in the Publishing Template descriptor file or in the transformation scenario.

The JavaScript Modules

The **JS Modules** sample template contains a main JavaScript module (`template-main.js`) that loads other modules stored in the template package or in a remote location on the Internet.

```
define(['require'], function (require) {
    require(['./red', './italic', './tables']);
});
```

Besides the main JavaScript example, the template contains 3 other sub-modules:

- **`red.js`** - Changes the font color of the publication title.

```
define(["jquery"], function ($) {
    $(document).ready(function () {
        // Make the title red
        $('.wh_publication_title a').attr('style', 'color:red');
    });
});
```

- **`italic.js`** - Changes the font style of your publication title.

```
define(["jquery"], function ($) {
    $(document).ready(function () {
```

```
        // Make the title italic
        $('.wh_publication_title a').wrapInner('<i></i>');
    });
});
```

- **tables.js** - Loads the DataTables jQuery plugin from a CDN.

```
define(["jquery",
    "https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"], function ($) {
    $(document).ready(function () {
        $('.table').DataTable();
    });
});
```

The JavaScript modules are stored in the Publishing Template package as follows:

```
[template-dir]
    [js]
        template-main.js
        italic.js
        red.js
        tables.js
```

Notes:

- The main module should be referenced in the Publishing Template descriptor file *(on page 819)* by specifying its path relative to the base directory of the template.

```
<js-amd-module file="js/template-main.js"/>
```

- The main JS module is copied automatically to the output directory, but the sub-modules are not. To instruct the Publishing Template engine to copy those modules to the output directory you should include a `<fileset>` section in the Publishing Template descriptor file *(on page 819)*.

```
<fileset>
    <include name="js/*.js"/>
</fileset>
```

- The main module can reference other modules bundled in the publishing template package as follows:
  - **Relative to template directory** - Prefix their paths with the ID of the template directory: **`template-base-dir`** (for example: `template-base-dir/js/italic`).
  - **Relative to the name of the current JS module** - Use `./` to prefix the paths of the referenced modules.
  - The `.js` extension **should be omitted** for local modules, because the *RequireJS* library will add it automatically.

Related Information:

How to Insert HTML Content *(on page 857)*

How to Change Numbering Styles for Ordered Lists

Ordered lists (`<ol>`) are usually numbered in XHTML output using numerals. If you want to change the numbering to alphabetical, follow these steps:

1. Define a custom `@outputclass` value and set it as an attribute of the ordered list, as in the following example:

```
<ol outputclass="number-alpha">
    <li>A</li>
    <li>B</li>
    <li>C</li>
</ol>
```

2. Add the following code snippet in a custom CSS file:

```
ol.number-alpha{
    list-style-type:lower-alpha;
}
```

3. Reference the CSS file in a *WebHelp Responsive* transformation using an *Oxygen Publishing Template (on page 864)* or the `args.css` parameter *(on page 865)*.

Referencing the Custom CSS from a Publishing Template

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Using the **Navigator** view, copy your custom CSS in a folder inside the publishing template root folder (for example, in the `custom_footer_template/resources` folder).
3. Open the template descriptor file *(on page 819)* associated with your publishing template and add your custom CSS in the *resources* section.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <resources>
            ...
            <css file="resources/MyCustom.css"/>
```

4. Open the *DITA Map WebHelp Responsive* transformation scenario.
5. Click the **Choose Custom Publishing Template** link and select your template.
6. Click **OK** to save the changes to the transformation scenario.
7. Run the transformation scenario.

Result: Your custom CSS will be applied as a final layer on top of any existing CSS rules and the output will reflect the changes you made.

Referencing the CSS Using the *args.css* Parameter

1. Edit the *DITA Map WebHelp Responsive* transformation scenario and open the **Parameters** tab.
2. Set the `args.css` parameter to the path of your custom CSS file.
3. Set the `args.copycss` parameter to `yes` to automatically copy your custom CSS in the output folder when the transformation scenario is processed.
4. Click **OK** to save the changes to the transformation scenario.
5. Run the transformation scenario.

Result: Your custom CSS will be applied as a final layer on top of any existing CSS rules and the output will reflect the changes you made.

How to Show or Hide Navigation Links in Topic Pages

The topic pages *(on page 779)* in WebHelp Responsive output can contain navigation links (⬅ **Previous** / ➡ **Next** arrows) that can be used to navigate to the previous or next topic.

How to Control Which Topic Pages Include Navigation Links

The navigation links are controlled by the `@collection-type` attribute. For example, if you set `collection-type="sequence"` on a parent topic reference in your DITA map, navigation links will be generated in the output for all of its child topics (from children to parent, and from child to previous sibling and next sibling).

```
<map id="example_map" title="Example Map">
 <topicref href="../topics/ParentTopic.dita" collection-type="sequence">
  <topicref href="../topics/Childtopic.dita"/>
 </topicref>
```

How to Generate Navigation Links for All Topics (Ignoring the Collection Type Attribute)

You can use the `webhelp.default.collection.type.sequence` parameter in the transformation and set its value to `yes` to generate navigation links for all topics, regardless of whether or not the `collection-type` attribute is present.

How to Hide All Navigation Links

To hide all navigation links, use the `webhelp.show.navigation.links` parameter in the transformation and set its value to `no`.

How to Change the Main Page Layout

This section contains topics that explain how to customize the layout of the main page in the WebHelp Responsive output.

How to Customize 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 (on page 1324)* 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 Eclipse 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. Reference the CSS file in a *WebHelp Responsive* transformation using an *Oxygen Publishing Template (on page 856)* or the `args.css` parameter *(on page 857)*.

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.

Using a Publishing Template

To hide the menu using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Open the template descriptor file *(on page 819)* associated with your publishing template and add the `webhelp.show.top.menu` parameter in the *parameters* section with its value set to `no`.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <parameters>
            <parameter name="webhelp.show.top.menu" value="no"/>
        </parameters>
    </webhelp>
```

3. Open the *DITA Map WebHelp Responsive* transformation scenario.
4. Click the **Choose Custom Publishing Template** link and select your template.
5. Click **OK** to save the changes to the transformation scenario.
6. Run the transformation scenario.

Using a Transformation Scenario in Oxygen XML Editor/Author

To hide the menu using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Edit the *DITA Map WebHelp Responsive* transformation scenario and choose a *template*.
2. Open the **Parameters** tab and set the `webhelp.show.top.menu` parameter to `no`.
3. Click **OK** to save the changes to the transformation scenario.
4. Run the transformation scenario.

How to Add a Welcome Message in the WebHelp Responsive Main Page

The main page of the WebHelp Responsive output contains a set of empty placeholders *(on page 825)* that can be used to display customized text fragments. These placeholders are available to you through WebHelp Responsive transformation scenario parameters. For example, the placeholder identified through the `webhelp.fragment.welcome` parameter displays text content above the search box in the main page.

Using a Publishing Template

To add a customized welcome message in the main page of the WebHelp Responsive output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Open the template descriptor file *(on page 819)* associated with your publishing template and add the `webhelp.fragment.welcome` parameter in the *parameters* section with its value set to one of the following:

- A small well-formed XHTML fragment (such as: `<i>Welcome to the User Guide</i>`).
- A path to a file that contains well-formed XHTML content.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <parameters>
            <parameter name="webhelp.fragment.welcome" value="c:\myMessage.xhtml"/>
        </parameters>
    </webhelp>
```

3. Open the *DITA Map WebHelp Responsive* transformation scenario.
4. Click the **Choose Custom Publishing Template** link and select your template.
5. Click **OK** to save the changes to the transformation scenario.
6. Run the transformation scenario.

Result: In the WebHelp output, your custom message will be displayed above the search box in the main page.

Using a Transformation Scenario in Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To add a customized welcome message in the main page of the WebHelp Responsive output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Edit the *DITA Map WebHelp Responsive* transformation scenario and choose a *template*.
2. Open the **Parameters** tab and set the `webhelp.fragment.welcome` parameter with its value set to one of the following:
   - A small well-formed XHTML fragment (such as: `<i>Welcome to the User Guide</i>`).
   - A path to a file that contains well-formed XHTML content.
3. Click **OK** to save the changes to the transformation scenario.
4. Run the transformation scenario.

Result: In the WebHelp output, your custom message will be displayed above the search box in the main page.

How to Create a Custom Footer

The main page of the WebHelp Responsive output contains a set of empty placeholders *(on page 825)* that can be used to display customized text fragments. These placeholders are available to you through

WebHelp Responsive transformation scenario parameters. For example, the placeholder identified through the `webhelp.fragment.footer` parameter displays the custom content at the bottom of the page.

Using a Publishing Template

To create a custom footer in the WebHelp Responsive output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Open the template descriptor file *(on page 819)* associated with your publishing template and add the `webhelp.fragment.footer` parameter in the *parameters* section with its value set to one of the following:
   - A small well-formed XHTML fragment.
   - A path to a file that contains well-formed XHTML content.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <parameters>
            <parameter name="webhelp.fragment.footer" value="c:\myFooter.xhtml"/>
        </parameters>
    </webhelp>
```

   Important: This parameter should only be used if you are using a valid, purchased license of Oxygen XML Developer Eclipse plugin (do not use it with a trial license).
3. Open the *DITA Map WebHelp Responsive* transformation scenario.
4. Click the **Choose Custom Publishing Template** link and select your template.
5. Click **OK** to save the changes to the transformation scenario.
6. Run the transformation scenario.

Result: In the WebHelp output, your custom footer will be displayed at the bottom of the page.

Using a Transformation Scenario in Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To create a custom footer in the WebHelp Responsive output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Edit the *DITA Map WebHelp Responsive* transformation scenario and choose a *template*.

2. Open the **Parameters** tab and set the `webhelp.fragment.footer` parameter with its value set to one of the following:
    - A small well-formed XHTML fragment.
    - A path to a file that contains well-formed XHTML content.
3. Click **OK** to save the changes to the transformation scenario.
4. Run the transformation scenario.

Result: In the WebHelp output, your custom footer will be displayed at the bottom of the page.

How to Configure the Tiles on the WebHelp Responsive 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 (on page 1324)*. 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 the 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 Eclipse 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. Reference the CSS file in a *WebHelp Responsive* transformation using an *Oxygen Publishing Template (on page 856)* or the `args.css` parameter *(on page 857)*.

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 Eclipse 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 to 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. Reference the CSS file in a *WebHelp Responsive* transformation using an *Oxygen Publishing Template (on page 856)* or the `args.css` parameter *(on page 857)*.

Adding Graphics and Media Resources

This section contains topics that explain how to add media resources to the published output or the output directory.

How to Copy Additional Resources to Output Directory

You can copy additional resources (such as JavaScript, CSS, or other resources) to the output directory either by using an *Oxygen Publishing Template (on page 1327)* or the `webhelp.custom.resources` parameter.

Copying Additional Resources to the Output Directory using a Publishing Template

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Add a new `<fileset>` element in the resources section of the template descriptor file *(on page 822)*.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <resources>
            <fileset>
                <include name="custom-resources/**/*"/>
                <exclude name="**/*.git"/>
            </fileset>
```

3. Open the *DITA Map WebHelp Responsive* transformation scenario.
4. Click the **Choose Custom Publishing Template** link and select your template.
5. Click **OK** to save the changes to the transformation scenario.
6. Run the transformation scenario.

Results: All files from the custom resources directory will be copied to the *`WebHelp Output Directory`*`/oxygen-webhelp/template` folder.

Copying Additional Resources to the Output Directory using a Transformation Parameter

1. Place all your resources in the same directory.
2. Edit the *DITA Map WebHelp Responsive* transformation scenario and open the **Parameters** tab.
3. Edit the value of the `webhelp.custom.resources` parameter and set it to the absolute path of the directory in step 1.
4. Click **OK** to save the changes to the transformation scenario.
5. Run the transformation scenario.

Results: All files from the new directory will be copied to the root of the WebHelp output directory.

How to Add a Logo Image in the Title Area

You can customize **WebHelp Responsive** output to include a logo in the title area. It will be displayed before the publication title. You can also specify a URL that can be used to send users to a specific website when they click the logo image.

This customization can be done using an *Oxygen Publishing Template* or using a transformation scenario from within **Oxygen XML Editor/Author**.

Using a Publishing Template

To add a logo in the title area of your WebHelp output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Open the template descriptor file *(on page 819)* associated with your publishing template and add the `webhelp.logo.image` parameter in the *parameters* section with its value set to the path of your logo.
3. If you also want to add a link to your website when you click the logo image, set its URL in the `webhelp.logo.image.target.url` parameter.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <parameters>
            <parameter name="webhelp.logo.image" value="c:\myLogo.jpg"/>
            <parameter name="webhelp.logo.image.target.url" value="http://sample.com"/>
        </parameters>
    </webhelp>
```

4. Open the *DITA Map WebHelp Responsive* transformation scenario.
5. Click the **Choose Custom Publishing Template** link and select your template.
6. Click **OK** to save the changes to the transformation scenario.
7. Run the transformation scenario.

Using a Transformation Scenario in Oxygen XML Editor/Author

To add a logo in the title area of your WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Edit the *DITA Map WebHelp Responsive* transformation scenario and choose a *template*.
2. Open the **Parameters** tab and set the `webhelp.logo.image` parameter to the path of your logo.
3. If you also want to add a link to your website when you click the logo image, set its URL in the `webhelp.logo.image.target.url` parameter.
4. Click **OK** to save the changes to the transformation scenario.
5. Run the transformation scenario.

How to Add a Favicon in WebHelp Systems

You can add a custom *favicon* to your WebHelp output by simply using a parameter in the transformation scenario to point to your *favicon* image.

This customization can be done using an *Oxygen Publishing Template* or using a transformation scenario from within **Oxygen XML Editor/Author**.

Using a Publishing Template

To add a *favicon* to your WebHelp output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Open the template descriptor file *(on page 819)* associated with your publishing template and add the `webhelp.favicon` parameter in the *parameters* section with its value set to the path of your image.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <parameters>
            <parameter name="webhelp.favicon" value="c:\myFavicon.jpg"/>
        </parameters>
    </webhelp>
```

3. Open the *DITA Map WebHelp Responsive* transformation scenario.
4. Click the **Choose Custom Publishing Template** link and select your template.
5. Click **OK** to save the changes to the transformation scenario.
6. Run the transformation scenario.

Result: Browsers that provide *favicon* support will display the *favicon* (typically in the browser's address bar, in the list of bookmarks, and in the history).

Using a Transformation Scenario in Oxygen XML Editor/Author

To add a *favicon* to your WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Edit the *DITA Map WebHelp Responsive* transformation scenario and choose a *template*.
2. Open the **Parameters** tab and set the `webhelp.favicon` parameter to the path of your image.
3. Click **OK** to save the changes to the transformation scenario.
4. Run the transformation scenario.

How to Add 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 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 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 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 video using a parameter like this:

   ```
   <object outputclass="audio">
     <param name="src" value="audio/MyClip.mp3"/>
   </object>
   ```
2. Apply a **DITA to WebHelp** transformation 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. If you want the video to be allowed to play in full screen mode once the document is converted to XHTML output, also add an `allowfullscreen` parameter and set its value to **true**:

```
<object outputclass="iframe" data="https://www.youtube.com/embed/m_vv2s5Trn4"/>
  <param name="allowfullscreen" value="true"/>
</object>
```

3. Apply a **DITA to WebHelp** transformation 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>
```

How to Add MathML Equations in WebHelp Output

Currently, only **Firefox** can render **MathML** equations embedded in the **HTML** code. MathJax is a solution to properly view MathML equations embedded in **HTML** content in a variety of browsers.

If you have DocBook content that has embedded **MathML** equations and you want to properly view the equations in published HTML output types (such as WebHelp), you need to add a reference to the MathJax script in the **head** element of all HTML files that have the equation embedded.

For example:

```
<script type="text/javascript"


 src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.1/MathJax.js?config=TeX-AMS-MML_HTMLorMML"
>
</script>
```

Result: The equation should now be properly rendered in the WebHelp output for other browsers.

Searching the Output

This section contains topics that explain how to use some of the search features in WebHelp Responsive output.

How to Change Element Scoring 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. The WebHelp directory 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. The properties file includes instructions and examples to help you with your customization. The file is located in: *DITA-OT-DIR*`\plugins\com.oxygenxml.webhelp.responsive\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 transformation.

How to Exclude Certain DITA Topics from Search Results

There are several ways to exclude certain DITA resources from your WebHelp system's search results. This is useful if you have topics in your *DITA map (on page 1324)* structure that you do not want to be included in search results for your WebHelp system. The first method involves setting a parameter in the WebHelp transformation scenario and the second involves setting an attribute for each DITA topic reference that you want to exclude.

Transformation Parameter Method

To exclude DITA topics from WebHelp search results using a transformation parameter, follow these steps:

1. Create a simple text file that will contain your excluded file patterns. Each 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.

Example: Suppose that in your project, you want to exclude all files located in the `resources` directory and all files located in the `topics` directory that have a `.bak` file extension. You could create a simple text file (for example, named `exclude.properties`), and add the following lines:

```
resources/*
topics/*.bak
```

2. Set the `webhelp.search.custom.excludes.file` parameter to specify the path to the file that contains the excluded file patterns (for example, `exclude.properties` in step 1). The parameter can be specified in the *parameters* section of the template descriptor file *(on page 823)* associated with your publishing template or in the **Parameters** tab of the transformation scenario dialog box in **Oxygen XML Editor/Author**.
3. Run the transformation.

Search Attribute Method

The WebHelp **Search** engine does not index DITA topics that have the `@search` attribute set to `no`.

To exclude DITA topics from WebHelp search results using this attribute, 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. Run your WebHelp system transformation.

How to Optimize Search Results

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** (optional) - The date when the page was last modified. The date format is `YYYY-MM-DD`.
- **changefreq** (optional) - 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** (optional) - 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.

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 Eclipse 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.

How to Index Japanese Content

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 Editor/Author** installation kit.

Restriction: The Kuromoji analyzer does not work if your WebHelp output is accessed locally. In this scenario, a warning message will be displayed informing you that the Kuromoji analyzer is disabled. It is possible to hide this warning message by using a transformation parameter named `webhelp.enable.search.kuromoji`. By default, its value is **yes**, which means the Kuromoji analyzer is enabled by default. To hide the warning message, set the value of the `webhelp.enable.search.kuromoji` parameter to **no** using either of the methods listed below. When that parameter is set to **no**, the Kuromoji analyzer is disabled even if you deploy your WebHelp output on a web server.

Using a Publishing Template

To add a logo in the title area of your WebHelp output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Open the template descriptor file *(on page 819)* associated with your publishing template and add the `default.language` parameter in the *parameters* section with its value set to `ja-jp`.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <parameters>
            <parameter name="default.language" value="ja-jp"/>
        </parameters>
    </webhelp>
```

3. Open the *DITA Map WebHelp Responsive* transformation scenario.
4. Click the **Choose Custom Publishing Template** link and select your template.
5. Click **OK** to save the changes to the transformation scenario.
6. Run the transformation scenario.

Using a Transformation Scenario in Oxygen XML Editor/Author

To activate the Japanese indexing in your WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Edit a **DITA to WebHelp** transformation scenario and in the **Parameters** tab, set the value of the `default.language` parameter to `ja-jp`.

   Note: Alternatively, you could set the `@xml:lang` attribute on the root of the *DITA map (on page 1324)* and the referenced topics to `ja-jp`. Another alternative for DITA output is to use the `webhelp.search.japanese.dictionary` parameter to specify a path to a Japanese dictionary that will be used by the *Kuromoji* morphological engine (note that the encoding for the dictionary must be **UTF8**).
2. Run the WebHelp transformation scenario to generate the output.

Localization

This section contains topics that explain how to use the localization support in WebHelp Responsive output.

How to Localize the Interface of WebHelp Responsive Output

Static labels used in the WebHelp output are stored in translation files that have the *strings-lang1-lang2.xml* name format, where *lang1* and *lang2* are ISO language codes. For example, the US English labels are kept in the *strings-en-us.xml* file.

These translation files are collected from two locations:

- ***DITA-OT-DIR*`/plugins/org.dita.base/xsl/common` folder** - DITA-OT's default translations (generated text for `<note>`, `<fig>`, and `<table>` elements).
- ***DITA-OT-DIR*`/plugins/com.oxygenxml.webhelp.responsive/oxygen-webhelp/resources/localization` folder** - These translations are contributed by the WebHelp plugin and extend the default ones provided by DITA-OT. The labels defined in this folder take precedence over the DITA-OT defaults.

There are two major reasons you may want to use modify the translation files: to modify the existing strings or to translate to a new language.

Related Information:

How to Index Japanese Content *(on page 880)*

Customizing Generated Text

Modifying the Existing Strings

To modify the generated text for WebHelp transformations, you need to create a DITA-OT extension plugin that uses the *dita.xsl.strings* extension point. The following procedure is for changing English labels, but you can adapt it for any language:

1. Create a `com.oxygenxml.webhelp.localization` plugin directory inside the *`DITA-OT-DIR`*`/plugins/` location.
2. Create a `plugin.xml` file inside that `com.oxygenxml.webhelp.localization` directory with the following content:

```
<plugin id="com.oxygenxml.webhelp.localization">
    <require plugin="com.oxygenxml.webhelp.classic"/>
    <require plugin="com.oxygenxml.webhelp.responsive"/>

    <feature extension="dita.xsl.strings" file="webhelp-extension-strings.xml"/>
</plugin>
```

3. Create a `webhelp-extension-strings.xml` file with the following content:

```
<langlist>
    <lang xml:lang="en"    filename="strings-en-us.xml"/>
    <lang xml:lang="en-us"    filename="strings-en-us.xml"/>
</langlist>
```

4. Copy the strings you want to change from the translation files *(on page 881)* to the `strings-en-us.xml` file. Make sure you leave the name attribute unchanged because this is the key used to look up the string. A sample content might be:

```
<strings xml:lang="en-US">
  <str name="Figure">Fig</str>
  <str name="Draft comment">ADDRESS THIS DRAFT COMMENT</str>
</strings>
```

5. Use the **Run DITA-OT Integrator** transformation scenario *(on page 684)* found in the **DITA Map** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

Adding a New Language

To add a new language for WebHelp transformations, you need to create a DITA-OT extension plugin that uses the *dita.xsl.strings* extension point. The following sample procedure is for adding translation files for the Polish language, but you can adapt it for any language:

1. Create a `com.oxygenxml.webhelp.localization` plugin directory inside the *`DITA-OT-DIR`*`/plugins/` location.
2. Create a `plugin.xml` file inside that `com.oxygenxml.webhelp.localization` directory with the following content:

```
<plugin id="com.oxygenxml.webhelp.localization">
    <require plugin="com.oxygenxml.webhelp.classic"/>
    <require plugin="com.oxygenxml.webhelp.responsive"/>

    <feature extension="dita.xsl.strings" file="webhelp-extension-strings.xml"/>
</plugin>
```

3. Create a `webhelp-extension-strings.xml` file with the following content:

```
<langlist>
    <lang xml:lang="pl"    filename="strings-pl-pl.xml"/>
    <lang xml:lang="pl-PL"    filename="strings-pl-pl.xml"/>
</langlist>
```

4. Copy the WebHelp strings file (*DITA-OT-DIR*`/plugins/com.oxygenxml.webhelp.responsive/oxygen-webhelp/resources/localization/strings-en-us.xml`) to your plugin directory, and rename it as `strings-pl-pl.xml`.
5. In the `strings-pl-pl.xml` file, change the *@xml:lang* attribute on the root element that conforms with the new language.

```
<strings xml:lang="pl-PL">
  ...
</strings>
```

6. Translate the content of each `<str>` element (make sure to leave the *name* attribute unchanged).

```
<strings xml:lang="pl-PL">
...
 <str name="webhelp.content" js="true" php="false">Polish translation for "Content".</str>
 <str name="webhelp.search" js="true" php="false">Polish translation for "Search"</str>
...
</strings>
```

7. Copy the common DITA-OT strings defined in the *DITA-OT-DIR*`/xsl/common/strings-en-us.xml` file. It defines a set generated text available for HTML-based transformations (such as `<note>`, `<fig>`, and `<table>` elements). Translate the content of each `<str>` element.

```
<strings xml:lang="pl-PL">
...
 <str name="webhelp.content" js="true" php="false">Polish translation for "Content".</str>
 <str name="webhelp.search" js="true" php="false">Polish translation for "Search"</str>
...
 <str name="Figure">Polish translation for "Figure"</str>
 <str name="Table">Polish translation for "Table"</str>
...
</strings>
```

8. Use the **Run DITA-OT Integrator** transformation scenario *(on page 684)* found in the **DITA Map** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.

How to Activate Support for Right-to-Left (RTL) Languages

To activate support for RTL (right-to-left) languages in WebHelp output, edit the *DITA map (on page 1324)* 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

Integrating Social Media and Google Tools in the WebHelp Output

This section contains topics that explain how to integrate some of the most popular social media sites in WebHelp output.

How to Add a Facebook Like Button in WebHelp Responsive Output

It is possible to integrate Facebook™ into your **WebHelp Responsive** output and you can specify where you want the widget to appear in your WebHelp page.

Using a Publishing Template

To add a Facebook™ *Like* widget to your WebHelp output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. Go to the Facebook Developers website.
2. Fill in the displayed form, then click the **Get Code** button.
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. Open the template descriptor file *(on page 819)* associated with your publishing template.

5. Use one of the parameters that begin with *webhelp.fragment (on page 825)* in the *html-fragments* section of the descriptor file. Set the value of that parameter to reference the `facebook-widget.xml` file that you created earlier.

```
<publishing-template>
    ...
    <webhelp>
        ...
      <html-fragments>
        <fragment
          file="HTML-fragments/facebook-widget.xml"
          placeholder="webhelp.fragment.after.toc_or_tiles"/>
      </html-fragments>
    </webhelp>
```

6. Open the *DITA Map WebHelp Responsive* transformation scenario.
7. Click the **Choose Custom Publishing Template** link and select your template.
8. Click **OK** to save the changes to the transformation scenario.
9. Run the transformation scenario.

Using a Transformation Scenario in Oxygen XML Editor/Author

To add a Facebook™ *Like* widget to your WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Go to the Facebook Developers website.
2. Fill in the displayed form, then click the **Get Code** button.
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. Edit the *DITA Map WebHelp Responsive* transformation scenario and choose a *template*.
5. Switch to the **Parameters** tab. Depending on where you want to display the button, edit one of the parameters that begin with *webhelp.fragment (on page 825)*. Set that parameter to reference the `facebook-widget.xml` file that you created earlier.
6. Click **Ok** and run the transformation scenario.

How to Add Tweet Button in WebHelp Responsive Output

It is possible to integrate *Twitter* into your **WebHelp Responsive** output and you can specify where you want the widget to appear in your WebHelp page.

Using a Publishing Template

To add a Twitter™ *Tweet* widget to your WebHelp Responsive output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. Go to the Tweet button generator page.
2. Fill in the displayed form. The **Preview and code** area displays the code that you will need.
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. Open the template descriptor file *(on page 819)* associated with your publishing template.
5. Use one of the parameters that begin with *webhelp.fragment (on page 825)* in the *html-fragments* section of the descriptor file. Set the value of that parameter to reference the `tweet-button.xml` file that you created earlier.

```
<publishing-template>
    ...
    <webhelp>
        ...
      <html-fragments>
        <fragment
          file="HTML-fragments/tweet-button.xml"
          placeholder="webhelp.fragment.after.toc_or_tiles"/>
      </html-fragments>
    </webhelp>
```

6. Open the *DITA Map WebHelp Responsive* transformation scenario.
7. Click the **Choose Custom Publishing Template** link and select your template.
8. Click **OK** to save the changes to the transformation scenario.
9. Run the transformation scenario.

Using a Transformation Scenario in Oxygen XML Editor/Author

To add a Twitter™ *Tweet* widget to your WebHelp Responsive output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Go to the Tweet button generator page.
2. Fill in the displayed form. The **Preview and code** area displays the code that you will need.
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. Edit the *DITA Map WebHelp Responsive* transformation scenario and choose a *template*.
5. Switch to the **Parameters** tab. Depending on where you want to display the button, edit one of the parameters that begin with *webhelp.fragment (on page 825)*. Set that parameter to reference the `tweet-button.xml` file that you created earlier.
6. Click **Ok** and run the transformation scenario.

How to Integrate Google Analytics in WebHelp Responsive Output

You can use *Google Analytics* to track and report site data for your **WebHelp Responsive** output.

Using a Publishing Template

To integrate *Google Analytics* into your WebHelp Responsive output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. Create a new Google Analytics account (if you do not already have one) and log on.
2. Choose the Analytics solution that best fits the needs of your website.
3. Follow the on-screen instructions to obtain a **Tracking Code** that contains your *Tracking ID*. A **Tracking Code** looks like this:

```
<script>
 (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
 (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
 m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
 })(window,document,'script','//www.google-analytics.com/analytics.js','ga');
 ga('create', 'UA-XXXXXXXX-X', 'auto');
 ga('send', 'pageview');
</script>
```

4. Save the Tracking Code (obtained in the previous step) in a new XML file called `googleAnalytics.xml`. Note that the file should only contain the tracking code.

5. Open the template descriptor file *(on page 819)* associated with your publishing template.
6. Use the webhelp.fragment.after.body parameter *(on page 907)* in the *html-fragments* section of the descriptor file. Set the value of that parameter to reference the `googleAnalytics.xml` file that you created earlier. The content of this file will be copied at the end of all generated output pages, right before the ending `<body>` element. This ensures that the page is loaded before the Google Analytics servers are contacted, thus reducing page loading time.

```
<publishing-template>
    ...
    <webhelp>
        ...
      <html-fragments>
        <fragment
          file="HTML-fragments/googleAnalytics.xml"
          placeholder="webhelp.fragment.after.body"/>
      </html-fragments>
    </webhelp>
```

7. Open the *DITA Map WebHelp Responsive* transformation scenario.
8. Click the **Choose Custom Publishing Template** link and select your template.
9. Click **OK** to save the changes to the transformation scenario.
10. Run the transformation scenario.

Using a Transformation Scenario in Oxygen XML Editor/Author

To integrate *Google Analytics* into your WebHelp Responsive output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Create a new Google Analytics account (if you do not already have one) and log on.
2. Choose the Analytics solution that best fits the needs of your website.
3. Follow the on-screen instructions to obtain a **Tracking Code** that contains your *Tracking ID*. A **Tracking Code** looks like this:

```
<script>
 (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
 (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
 m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
 })(window,document,'script','//www.google-analytics.com/analytics.js','ga');
 ga('create', 'UA-XXXXXXXX-X', 'auto');
 ga('send', 'pageview');
</script>
```

4. Save the Tracking Code (obtained in the previous step) in a new XML file called `googleAnalytics.xml`. Note that the file should only contain the tracking code.
5. Edit the *DITA Map WebHelp Responsive* transformation scenario and choose a *template*.
6. Switch to the **Parameters** tab. Edit the webhelp.fragment.after.body parameter *(on page 907)* and set it to reference the `googleAnalytics.xml` file that you created earlier. The content of this file will be

copied at the end of all generated output pages, right before the ending `<body>` element. This ensures that the page is loaded before the Google Analytics servers are contacted, thus reducing page loading time.

7. Click **Ok** and run the transformation scenario.

How to Integrate Google Search in WebHelp Responsive Output

It is possible to integrate the *Google Search Engine* into your **WebHelp Responsive** output and you can specify where you want the results to appear in your WebHelp page.

Using a Publishing Template

To integrate the *Google Search Engine* into your WebHelp Responsive output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. Go to the Google Custom Search Engine page using your Google account.
2. Select the **Create a custom search engine** button.
3. Follow the on-screen instructions to create a search engine component for your site.

   Important: For the **Layout**, you must select **Results only** for the *Google Search Engine* to work with **Oxygen XML WebHelp Responsive**.
4. At the end of this process you should obtain a code snippet that 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>
```

5. Save the script into a well-formed HTML file called `googlecse.html`.
6. Open the template descriptor file *(on page 819)* associated with your publishing template and add the `webhelp.google.search.script` parameter in the *parameters* section with its value set to reference the `googlecse.html` file that you created earlier.

```
<publishing-template>
  ...
  <webhelp>
```

```
    ...
    <parameters>
        <parameter
            name="webhelp.google.search.script"
            value="resources/googlecse.html"
            type="filePath"/>
    </parameters>
</webhelp>
```

7. 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>`.
8. Open the *DITA Map WebHelp Responsive* transformation scenario.
9. Click the **Choose Custom Publishing Template** link and select your template.
10. Click **OK** to save the changes to the transformation scenario.
11. Run the transformation scenario.

Using a Transformation Scenario in Oxygen XML Editor/Author

To integrate the *Google Search Engine* into your WebHelp Responsive output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Go to the Google Custom Search Engine page using your Google account.
2. Select the **Create a custom search engine** button.
3. Follow the on-screen instructions to create a search engine for your site.

    Important: For the **Layout**, you must select **Results only** for the *Google Search Engine* to work with **Oxygen XML WebHelp Responsive**.
4. At the end of this process you should obtain a code snippet that 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>
```

5. Save the script into a well-formed HTML file called `googlecse.html`.
6. Edit the *DITA Map WebHelp Responsive* transformation scenario and choose a *template*.
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** and run the transformation scenario.

XSLT Extensions for WebHelp Responsive

Since WebHelp Responsive output is primarily obtained by running XSLT transformations over the DITA input files, one customization method would be to override the default XSLT templates that are used by the WebHelp Responsive transformations.

There are two methods available to override the XSLT stylesheets implied by the WebHelp Responsive transformation.

- Use *XSLT-import extension points* from an *Oxygen Publishing Template (on page 1327)*.

  Note: Use this method if you want to affect only the transformations that use this *publishing template*.
- Use *XSLT-import extension points* from a DITA-OT extension plugin.

  Note: This method will affect all the outputs generated with the WebHelp system.

Related Information:

WebHelp Responsive XSLT-Import and XSLT-Parameter Extension Points *(on page 912)*

How to Use XSLT Extension Points from a Publishing Template

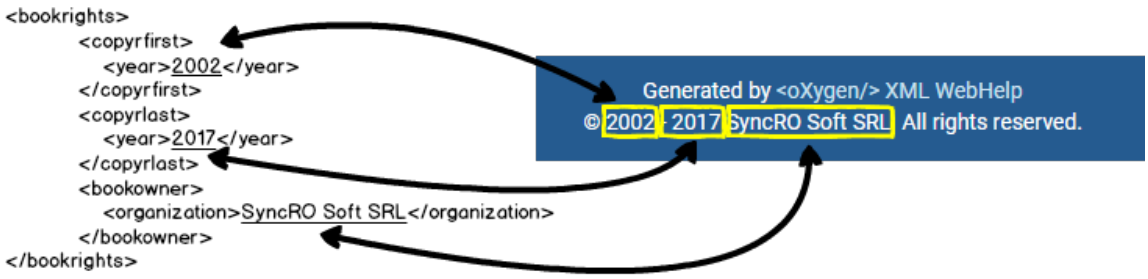
This example demonstrates how to use WebHelp XSLT-import Extension Points from an *Oxygen Publishing Template (on page 962)*.

Use Case 1: Add Copyright Information Extracted from a DITA Bookmap

Suppose you want to customize the WebHelp Responsive main page by adding information about the legal rights associated with the book in the footer (for example, copyright dates and owner). This information is specified in the bookmap:

```
<bookrights>
  <copyrfirst>
    <year>2002</year>
  </copyrfirst>
  <copyrlast>
    <year>2017</year>
  </copyrlast>
  <bookowner>
    <organization>SyncRO Soft SRL</organization>
  </bookowner>
</bookrights>
```

Figure 303. Example: Copyright information added in the WebHelp footer



The XSLT stylesheet that generates the main page is located in: *DITA-OT-DIR*`\plugins\com.oxygenxml.webhelp.responsive\xsl\mainFiles\createMainPage.xsl`. This XSLT stylesheet declares the `copy_template` mode that processes the main page template *(on page 832)* to expand its components. The main page template declares a component for the footer section that looks like this:

```
<div class=" footer-container text-center ">
  <whc:include_html href="${webhelp.fragment.footer}"/>
</div>
```

In the following example, the extension stylesheet will add a template that matches this component. It applies the default processing and adds the copyright information at the end.

```
<xsl:template match="*:div[contains(@class, 'footer-container')]" mode="copy_template">
  <!-- Apply the default processing -->
```

```
  <xsl:next-match/>

  <!-- Add a div containing the copyright information -->
  <div class="copyright_info">
      <xsl:choose>
          <!-- Adds the start-end years if they are defined -->
          <xsl:when test="exists($toc/*:topicmeta/*:bookrights/*:copyrfirst) and
                              exists($toc/*:topicmeta/*:bookrights/*:copyrlast)">
              <span class="copyright_years">
                 &#xa9;<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrfirst"/>
                       -<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrlast"/>
              </span>
          </xsl:when>

          <!-- Adds only the first year if last is not defined. -->
          <xsl:when test="exists($toc/*:topicmeta/*:bookrights/*:copyrfirst)">
              <span class="copyright_years">
                 &#xa9;<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrfirst"/>
              </span>
          </xsl:when>
      </xsl:choose>


      <xsl:if test="exists($toc/*:topicmeta/*:bookrights/*:bookowner/*:organization)">
          <span class="organization">
              <xsl:text> </xsl:text><xsl:value-of
                   select="$toc/*:topicmeta/*:bookrights/*:bookowner/*:organization"/>
              <xsl:text>. All rights reserved.</xsl:text>
          </span>
      </xsl:if>
  </div>
</xsl:template>
```

To add this functionality using a *Oxygen Publishing Template*, follow these steps:

1. If you haven't already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Link the folder associated with the publishing template to your current project in the **Navigator** view.
   You should have the `custom_footer_template` folder linked in your project.
3. Using the **Navigator** view, create an `xslt` folder inside the project root folder.
   You should have the `custom_footer_template/xsl` folder in your project.

4. Create your customization stylesheet (for example, **custom_mainpage.xsl**) in the `custom_footer_template/xsl` folder. Edit it to override the template that produces the footer section:

```
<xsl:template match="*:div[contains(@class, 'footer-container')]" mode="copy_template">
  <!-- Apply the default processing -->
  <xsl:next-match/>
  <!-- Add a div containing the copyright information -->
  <div class="copyright_info">
      <xsl:choose>
          <!-- Adds the start-end years if they are defined -->
          <xsl:when test="exists($toc/*:topicmeta/*:bookrights/*:copyrfirst) and
                       exists($toc/*:topicmeta/*:bookrights/*:copyrlast)">
              <span class="copyright_years">
                 &#xa9;<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrfirst"/>
                       -<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrlast"/>
              </span>
          </xsl:when>

          <!-- Adds only the first year if last is not defined. -->
          <xsl:when test="exists($toc/*:topicmeta/*:bookrights/*:copyrfirst)">
              <span class="copyright_years">
                 &#xa9;<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrfirst"/>
              </span>
          </xsl:when>
      </xsl:choose>


      <xsl:if test="exists($toc/*:topicmeta/*:bookrights/*:bookowner/*:organization)">
          <span class="organization">
              <xsl:text> </xsl:text><xsl:value-of
                   select="$toc/*:topicmeta/*:bookrights/*:bookowner/*:organization"/>
              <xsl:text>. All rights reserved.</xsl:text>
          </span>
      </xsl:if>
  </div>
</xsl:template>
```

5. Open the template descriptor file *(on page 819)* associated with your publishing template and set the XSLT stylesheet created in the previous step with the *com.oxygenxml.webhelp.xsl.createMainPage* XSLT extension point.

```
<publishing-template>
    ...
```

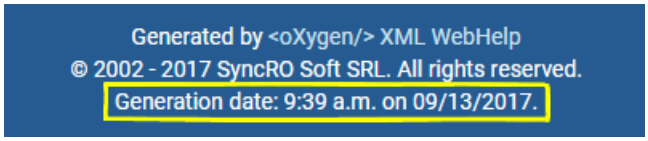
```
<webhelp>
    ...
    <xslt>
        <extension
        file="xslt/customMainPage.xsl"
        id="com.oxygenxml.webhelp.xsl.createMainPage"/>
```

6. Open the *DITA Map WebHelp Responsive* transformation scenario.
7. Click the **Choose Custom Publishing Template** link and select your template.
8. Click **OK** to save the changes to the transformation scenario.
9. Run the transformation scenario.

Use Case 2: Add generation time in the output footer

Another possible customization for the main page is to add the generation time in its footer. A transformation parameter is used to control whether or not this customization is active.

Figure 304. Generation time added in the WebHelp footer



To add this functionality, follow these steps:

1. In the customization stylesheet that you just created (for example, **custom_mainpage.xsl**), modify the template by adding the following XSLT code at the end.

```
<xsl:if test="oxyf:getParameter('webhelp.footer.add.generation.time') = 'yes'">
  <div class="generation_time">
    Generation date: <xsl:value-of
    select="format-dateTime(
                current-dateTime(),
                '[h1]:[m01] [P] on [M01]/[D01]/[Y0001].')"/>
  </div>
</xsl:if>
```

   Note: You can read the value of a WebHelp transformation parameter from your XSLT extension stylesheets by using the `getParameter(param.name)` function from the `http://www.oxygenxml.com/functions` namespace.
2. Open the template descriptor file *(on page 819)* associated with your publishing template and set the `webhelp.footer.add.generation.time` parameter to the default value.

```
<publishing-template>
    ...
    <webhelp>
```

```
...
<parameters>
  <parameter
    name="webhelp.footer.add.generation.time"
    value="yes"/>
```

3. Open the *DITA Map WebHelp Responsive* transformation scenario.
4. In the **Parameters** tab, you can change the value of the `webhelp.footer.add.generation.time` parameter.
5. Click **OK** to save the changes to the transformation scenario.
6. Run the transformation scenario.

How to Use XSLT Extension Points from a DITA-OT Plugin

In this example, the main page footer is modified by adding copyright information extracted from the DITA bookmap or by adding the output generation time. The first use-case uses an XSLT-Import extension point while the second uses an XSLT-Parameter extension point.

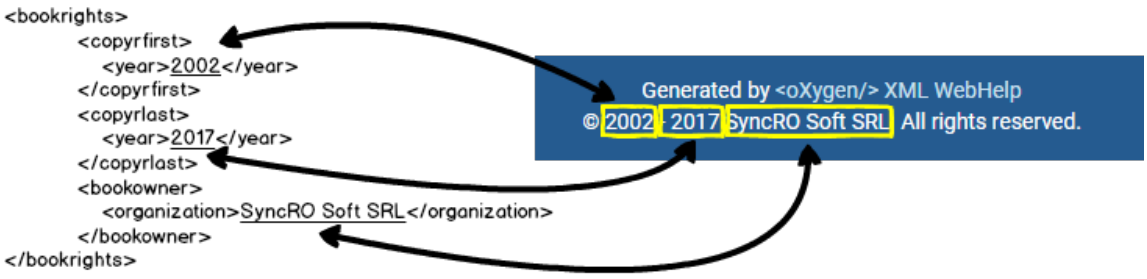
Note: This customization is available as a GitHub project at: https://github.com/oxygenxml/com.oxygenxml.webhelp.responsive.custom.footer.

Use Case 1: WebHelp XSLT-Import extension point to add copyright information extracted from a DITA Bookmap

Suppose you want to customize the WebHelp Responsive main page by adding information about the legal rights associated with the book in the footer (for example, copyright dates and owner). This information is specified in the bookmap:

```
<bookrights>
  <copyrfirst>
    <year>2002</year>
  </copyrfirst>
  <copyrlast>
    <year>2017</year>
  </copyrlast>
  <bookowner>
    <organization>SyncRO Soft SRL</organization>
  </bookowner>
</bookrights>
```

Figure 305. Example: Copyright information added in the WebHelp footer



The XSLT stylesheet that generates the main page is located in: *DITA-OT-DIR*`\plugins\com.oxygenxml.webhelp.responsive\xsl\mainFiles\createMainPage.xsl`. This XSLT stylesheet declares the `copy_template` mode that processes the main page template to expand its components. The main page template *(on page 832)* declares a component for the footer section that looks like this:

```
<div class=" footer-container text-center ">
  <whc:include_html href="${webhelp.fragment.footer}"/>
</div>
```

In the following example, the extension stylesheet will add a template that matches this component. It applies the default processing and adds the copyright information at the end.

```
<xsl:template match="*:div[contains(@class, 'footer-container')]" mode="copy_template">
  <!-- Apply the default processing -->
  <xsl:next-match/>

  <!-- Add a div containing the copyright information -->
  <div class="copyright_info">
      <xsl:choose>
          <!-- Adds the start-end years if they are defined -->
          <xsl:when test="exists($toc/*:topicmeta/*:bookrights/*:copyrfirst) and
                          exists($toc/*:topicmeta/*:bookrights/*:copyrlast)">
              <span class="copyright_years">
                 &#xa9;<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrfirst"/>
                       -<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrlast"/>
              </span>
          </xsl:when>

          <!-- Adds only the first year if last is not defined. -->
          <xsl:when test="exists($toc/*:topicmeta/*:bookrights/*:copyrfirst)">
              <span class="copyright_years">
                 &#xa9;<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrfirst"/>
              </span>
          </xsl:when>
      </xsl:choose>
```

```
    <xsl:if test="exists($toc/*:topicmeta/*:bookrights/*:bookowner/*:organization)">
        <span class="organization">
            <xsl:text> </xsl:text><xsl:value-of
                select="$toc/*:topicmeta/*:bookrights/*:bookowner/*:organization"/>
            <xsl:text>. All rights reserved.</xsl:text>
        </span>
    </xsl:if>
  </div>
</xsl:template>
```

You can implement this functionality with a WebHelp extension plugin that uses the **com.oxygenxml.webhelp.xsl.createMainPage** extension point *(on page 913)*. This extension point allows you to specify a customization stylesheet that will override the template described above.

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.responsive.custom.footer`).
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.responsive.custom.footer">
     <feature extension="com.oxygenxml.webhelp.xsl.createMainPage"
                file="custom_mainpage.xsl"/>
   </plugin>
   ```
3. Create your customization stylesheet (for example, **custom_mainpage.xsl**), and edit it to override the template that produces the footer section:

   ```
   <xsl:template match="*:div[contains(@class, 'footer-container')]" mode="copy_template">
     <!-- Apply the default processing -->
     <xsl:next-match/>
     <!-- Add a div containing the copyright information -->
     <div class="copyright_info">
         <xsl:choose>
             <!-- Adds the start-end years if they are defined -->
             <xsl:when test="exists($toc/*:topicmeta/*:bookrights/*:copyrfirst) and
                             exists($toc/*:topicmeta/*:bookrights/*:copyrlast)">
                 <span class="copyright_years">
                    &#xa9;<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrfirst"/>
                          -<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrlast"/>
                 </span>
             </xsl:when>
   ```

```
        <!-- Adds only the first year if last is not defined. -->
        <xsl:when test="exists($toc/*:topicmeta/*:bookrights/*:copyrfirst)">
            <span class="copyright_years">
               &#xa9;<xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyrfirst"/>
            </span>
        </xsl:when>
    </xsl:choose>

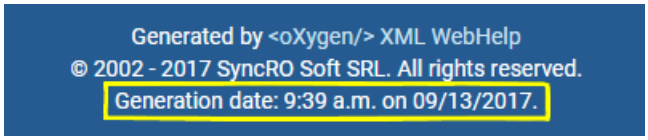

    <xsl:if test="exists($toc/*:topicmeta/*:bookrights/*:bookowner/*:organization)">
        <span class="organization">
            <xsl:text> </xsl:text><xsl:value-of
                 select="$toc/*:topicmeta/*:bookrights/*:bookowner/*:organization"/>
            <xsl:text>. All rights reserved.</xsl:text>
        </span>
    </xsl:if>
  </div>
</xsl:template>
```

4. Use the **Run DITA-OT Integrator** transformation scenario *(on page 684)* found in the **DITA Map** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.
5. Run a **DITA Map WebHelp Responsive** transformation scenario to obtain the customized side TOC.

Use-Case 2: WebHelp XSLT-Parameter extension point to control if generation time is displayed in the output

Another possible customization for the main page is to add the generation time in its footer. You can use an XSLT-Parameter extension point to control whether or note this customization is active. In this case, you can use the **com.oxygenxml.webhelp.xsl.createMainPage.param** extension point *(on page 914)*.

Figure 306. Generation time added in the WebHelp footer



To add this functionality, follow these steps:

1. Create a DITA-OT plugin structure by following the first 3 steps in the procedure above *(on page 897)*.
2. In the customization stylesheet that you just created (for example, **custom_mainpage.xsl**), declare `webhelp.footer.add.generation.time` as a global parameter and modify the template by adding the following XSLT code at the end.

```
<xsl:if test="$webhelp.footer.add.generation.time = 'yes'">
    <div class="generation_time">
        Generation date: <xsl:value-of select="format-dateTime(
```

```
                current-dateTime(), '[h1]:[m01] [P] on [M01]/[D01]/[Y0001].')"/>
    </div>
</xsl:if>
```

3. Edit the **plugin.xml** file to specify the **com.oxygenxml.webhelp.xsl.createMainPage.param** extension point and a custom parameter file by adding the following line:

```
<feature extension="com.oxygenxml.webhelp.xsl.createMainPage.param" file="params.xml"/>
```

4. Create a custom parameter file (for example, **params.xml**). It should look like this:

```
<dummy>
    <param name="webhelp.footer.add.generation.time"
        expression="${webhelp.footer.add.generation.time}"
        if="webhelp.footer.add.generation.time"/>
</dummy>
```

5. Use the **Run DITA-OT Integrator** transformation scenario *(on page 684)* found in the **DITA Map** section in the **Configure Transformation Scenario(s)** dialog box *(on page 765)*.
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 (`webhelp.footer.add.generation.time`).
7. Run the transformation scenario.

Related Information:

[DITA-OT] XSLT-Import Extension Points

[DITA-OT] XSLT-Parameter Extension Points

Miscellaneous Customization Topics

This section contains miscellaneous topics about how to customize the WebHelp Responsive output.

How to Add an Edit Link to Launch Oxygen XML Web Author

You can embed *Edit* links in the DITA WebHelp Responsive output that will automatically launch a particular document in Oxygen XML Web Author. A reviewer can then click the link to open the particular file in Oxygen XML Web Author where they can make or propose changes.

Using a Publishing Template

To embed an *Edit* link in the DITA Map WebHelp Responsive output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. If you have not already created a Publishing Template, see Working with Publishing Templates *(on page 848)*.
2. Open the template descriptor file *(on page 819)* associated with your publishing template and add the following parameters with their values set to the URLs:

- **editlink.ditamap.edit.url** - The URL of the DITA map used to publish your content. The easiest way to obtain the URL is to open the map in Web Author and copy the URL from the browser's address bar.
- **editlink.additional.query.parameters** - Optional query parameters to be appended to each generated edit link. Each parameter must start with *&* (for example: *&tags-mode=no-tags*).
- **editlink.remote.ditamap.url** (deprecated) - The URL of the DITA map suitable for opening in Oxygen XML Web Author.
- **editlink.web.author.url** (deprecated) - The URL of the Oxygen XML Web Author installation.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <parameters>
            <parameter name="editlink.remote.ditamap.url"
                       value="webdav-https://dav.box.com/dav/my.ditamap"/>
            <parameter name="editlink.web.author.url"
                       value="https://www.oxygenxml.com/webauthor.html"/>
        </parameters>
    </webhelp>
```

3. Open the *DITA Map WebHelp Responsive* transformation scenario.
4. Click the **Choose Custom Publishing Template** link and select your template.
5. Click **OK** to save the changes to the transformation scenario.
6. Run the transformation scenario.

Result: In the WebHelp output, all topics will have an **Edit** link to the right side of the title and clicking the link will launch that particular document in Oxygen XML Web Author.

For example:

- **Windows:**

```
dita.bat -i c:\mySample.ditamap -f webhelp-responsive -Deditlink.remote.ditamap.url=webdav-https://dav.box.com/dav/my.ditamap -Deditlink.web.author.url=https://www.oxygenxml.com/webauthor.html
```

- **Mac OS X/ Linux:**

```
dita -i /mySample.ditamap -f webhelp-responsive -Deditlink.remote.ditamap.url=webdav-https://dav.box.com/dav/my.ditamap -Deditlink.web.author.url=https://www.oxygenxml.com/webauthor.html
```

Using a Transformation Scenario in Oxygen XML Editor/Author

To embed an *Edit* link in the DITA Map WebHelp Responsive output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Edit a **DITA Map WebHelp Responsive** transformation scenario and open the **Parameters** tab.
2. Set values for the following parameters:

- **editlink.ditamap.edit.url** - The URL of the Oxygen XML Web Author that have opened the DITA map for editing.
- **editlink.additional.query.parameters** - Optional query parameters to be appended to each generated edit link. Must start with *&*, for e.g.: *&tags-mode=no-tags*.
- **editlink.remote.ditamap.url** (deprecated) - The URL of the DITA map suitable for opening in Oxygen XML Web Author.
- **editlink.web.author.url** (deprecated) - The URL of the Oxygen XML Web Author installation.

3. Run the transformation scenario.

Result: In the WebHelp output, all topics will have an **Edit** link to the right side of the title and clicking the link will launch that particular document in Oxygen XML Web Author.

Related Information:

Web Author Customization Guide: Embedding an Edit Link that will Launch Web Author

How to 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 elements (on page 1323)* (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 the text.

Using a Publishing Template

To flag content in DITA Map to WebHelp output using an *Oxygen Publishing Template (on page 816)*, follow this procedure:

1. Create a DITA filter file (DITAVAL) and add it in a directory of your choice (for example, named `myFile.ditaval`.
2. Define the property for the elements you want to be flagged. For example, if you want to flag any element that has the `@audience` attribute set to `programmer`, the content of the DITAVAL file should look like this:

```
<?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: For an element to be flagged, at least one attribute-value pair needs to have a property declared in the DITAVAL file.

3. Open the template descriptor file *(on page 819)* associated with your publishing template and add the `args.filter` parameter in the *parameters* section with its value set to the path of the DITAVAL file you created.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <parameters>
            <parameter name="args.filter" value="resources/myFile.ditaval"/>
        </parameters>
    </webhelp>
```

4. Open the *DITA Map WebHelp Responsive* transformation scenario.
5. Click the **Choose Custom Publishing Template** link and select your template.
6. Click **OK** to save the changes to the transformation scenario.
7. Run the transformation scenario.

Using a Transformation Scenario in Oxygen XML Editor/Author

To flag content in the DITA Map to WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Create a DITA filter file (DITAVAL) and add it in a directory of your choice (for example, named `myFile.ditaval`.
2. Define the property for the elements you want to be flagged. For example, if you want to flag any element that has the `@audience` attribute set to `programmer`, the content of the DITAVAL file should look like this:

```
<?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: For an element to be flagged, at least one attribute-value pair needs to have a property declared in the DITAVAL file.
3. Edit a **DITA Map to WebHelp** transformation scenario.
4. Specify the DITAVAL file in the **Filters** tab (with the **Use DITAVAL File** option).
5. Run the transformation scenario.

How to Deploy the 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.

Using a Script Outside of Oxygen XML Editor/Author

To publish WebHelp Responsive output on a SharePoint site using an *Oxygen Publishing Template (on page 816)*, 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`.
3. Open the template descriptor file *(on page 819)* associated with your publishing template and add the `args.outext` parameter in the *parameters* section with its value set to `.aspx`.

```
<publishing-template>
    ...
    <webhelp>
        ...
        <parameters>
            <parameter name="args.outext" value=".aspx"/>
        </parameters>
    </webhelp>
```

4. Open the *DITA Map WebHelp Responsive* transformation scenario.
5. Click the **Choose Custom Publishing Template** link and select your template.
6. Click **OK** to save the changes to the transformation scenario.
7. Run the transformation scenario.

Using a Transformation Scenario in Oxygen XML Editor/Author

To publish WebHelp Responsive output on a SharePoint site using a transformation scenario from within **Oxygen XML Editor/Author**, 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.
    a. Edit the WebHelp transformation scenario and open the **Parameters** tab.
    b. Set the `args.outext` parameter to `.aspx`.
    c. Run the transformation scenario.

How to View MathML Equations in HTML Output

Currently, only **Firefox** can render **MathML** equations embedded in the **HTML** code. MathJax is a solution to properly view MathML equations embedded in **HTML** content in a variety of browsers.

If you have DocBook or DITA content that has embedded **MathML** equations and you want to properly view the equations in published HTML output types (WebHelp, CHM, EPUB, etc.), you need to add a reference to the MathJax script in the **head** element of all HTML files that have the equation embedded.

For example:

```
<script type="text/javascript"


 src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.1/MathJax.js?config=TeX-AMS-MML_HTMLorMML"
>
</script>
```

For DITA documents, you can also use the following procedure:

1. Edit the DITA Map WebHelp transformation scenario and open the **Parameters** tab.
2. Set the following parameter to point to an XML resource file that contains your script, depending on your type of WebHelp system.
   - **WebHelp Responsive Systems** - Set the `webhelp.fragment.head` parameter to point to your resource file.
   - **WebHelp Classic Systems** - Set the `webhelp.head.script` parameter to point to your resource file.
3. Run the transformation scenario.

Result: The equation should now be properly rendered in other browsers, such as Edge, IE, or Chrome.

WebHelp Responsive Transformation Parameters

In addition to the common DITA-OT transformation parameters and the HTML-based Output Parameters, there are numerous other supported parameters that are specific to the WebHelp Responsive output.

Publishing Template Parameters

webhelp.publishing.template

Specifies the path to the ZIP archive (or root folder) that contains your custom WebHelp Responsive template.

Note: The built-in templates are stored in the *DITA-OT-DIR*`/plugins/com.oxygenxml.webhelp.responsive/templates` folder.

Note: Relative paths are resolved based on the current working directory.

webhelp.publishing.template.descriptor

Specifies the name of the descriptor to be loaded from the WebHelp Responsive template package. If it is not specified, the first encountered descriptor will be automatically loaded.

Custom Resource Parameters

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.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.logo.image.alt

Specifies a value that will be set in the `@alt` attribute of the logo image. If the parameter is not specified, the `@alt` attribute will contain the publication title. Note that this parameter makes sense only in conjunction with the `webhelp.logo.image` parameter.

Oxygen Feedback Parameter

webhelp.fragment.feedback

You can integrate **Oxygen Feedback** with your WebHelp Responsive output to provide a comments area at the bottom of each page where readers can offer feedback. When you create an **Oxygen Feedback** site configuration, an HTML fragment is generated during the final step of the creation process and that fragment should be set as the value for this parameter.

HTML Fragment Extension Parameters

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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** 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 a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment.

Important: This parameter should only be used if you are using a valid, purchased license of Oxygen XML Developer Eclipse plugin (do not use it with a trial license).

webhelp.fragment.head

In the generated output it inserts a given XHTML fragment in the `<head>` element. The value of the parameter can be either a **well-formed** 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 a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment.

Output Component Parameters

webhelp.default.collection.type.sequence

Specifies if the **sequence** value will be used by default when the `@collection-type` attribute is not specified. This option is helpful if you want to have *Next* and *Previous* navigational buttons generated for all HTML pages. Allowed values are **no** (default) and **yes**.

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.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.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.full.size.image

Specifies if responsive images that are displayed with a smaller dimension than their original size can be clicked to see an enlarged version of the image. The default value is `yes`.

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.publication.toc

Specifies if a table of contents will be presented on the left side of each topic in the output. The default value is `yes`.

webhelp.show.topic.toc

Specifies if a topic table of contents will be presented on the right side of each topic in the output. This table of contents contains links to each `<section>` within the current topic that contains an `@id` attribute and the section corresponding to the current scroll position is highlighted. 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.publication.toc.hide.chunked.topics

Specifies if the table of contents will contain links for *chunked* topics. The default value is `yes`.

webhelp.publication.toc.links

Specifies which links will be included in the table of contents. The possible values are:

- **chapter** (default) - The TOC will include links for the current topic, its children, its siblings, and its direct ancestor (including the direct ancestor's siblings), and the parent chapter.
- **topic** - The TOC will only include links for the current topic and its direct children.
- **all** - The TOC will include all links.

webhelp.publication.toc.tooltip.position

By default, if a topic contains a `<shortdesc>` element, its content is displayed in a tooltip when the user hovers over its link in the table of contents. This parameter controls whether or not this tooltip is displayed and its position relative to the link. The possible values are:

- **left** (default)
- **right**
- **top**
- **bottom**
- **hidden** - The tooltip will not be displayed.

webhelp.top.menu.depth

Specifies the maximum depth level of the topics that will be included in the top menu. The default value is `3`. A value of `0` means that the menu has unlimited depth.

webhelp.topic.collapsible.elements.initial.state

Specifies the initial state of collapsible elements (tables with titles, nested topics with titles, sections with titles, index term groups). The possible values are `collapsed` or `expanded` (default value).

Search-Related Parameters

webhelp.search.japanese.dictionary

The file path of the dictionary that will be used by the *Kuromoji* morphological engine for indexing Japanese content in the WebHelp pages. The encoding for the dictionary must be **UTF8**.

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.search.stop.words.exclude

Specifies a list of words that will be excluded from the default list of *stop words* that are filtered out before the search processing. Use comma separators to specify more than one word (for example: `if,for,is`).

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.enable.search.autocomplete

Specifies if the *Autocomplete* feature is enabled in the WebHelp search text field. The default value is `yes`.

webhelp.search.enable.pagination

Specifies whether or not search results will be displayed on multiple pages. Allowed values are `yes` or `no`.

webhelp.search.page.numberOfItems

Specifies the number of search results items displayed on each page. This parameter is only used when the **webhelp.search.enable.pagination** parameter is enabled.

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`.

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.

Note: The `fix.external.refs.com.oxygenxml` parameter is not supported when running the transformation from a command line. 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.

Related Information:

Generating WebHelp Responsive Output *(on page 847)*

Setting DITA-OT Parameters

WebHelp Responsive XSLT-Import and XSLT-Parameter Extension Points

XSLT extension points can be used from either from an *Oxygen Publishing Template* or from a DITA-OT extension plug-in.

Extension Points from an Oxygen Publishing Template

The publishing template allows you to specify one or more XSLT extension points. These extension points will affect only the transformations that use the particular template.

For an example of how to use these extensions in a publishing template, see: How to Use an XSLT Extension Point from a Publishing Template *(on page 893)* topic.

Example:

```
<publishing-template>
    ...
    <webhelp>
        ...
        <xslt>
            <extension
              id="com.oxygenxml.webhelp.xsl.createMainPage"
```

```
            file="xsl/customMainPage.xsl"/>
        </xslt>
```

Extension Points from a DITA-OT Extension Plug-in

The DITA-OT plug-in installer adds an XSLT import statement in the default WebHelp XSLT so that the XSLT stylesheet referenced by the extension point becomes part of the normal build. You can use these extension points to override XSLT processing steps.

Example:

```
<plugin id="com.oxygenxml.webhelp.responsive.extension">
  <feature extension="com.oxygenxml.webhelp.xsl.dita2webhelp"
           file="xsl/fixup.xsl"/>
</plugin>
```

XSLT-Import Extension Points

The following extension points are available:

com.oxygenxml.webhelp.xsl.dita2webhelp

Extension point to override the XSLT stylesheet (`dita2webhelp.xsl`) that produces an HTML file for each DITA topic. The location of this file is *DITA-OT-DIR*`\plugins\com.oxygenxml.webhelp.responsive\xsl\dita2webhelp\dita2webhelp.xsl`

com.oxygenxml.webhelp.xsl.createMainPage

Extension point to override the XSLT stylesheet (`createMainPage.xsl`) that produces the WebHelp Responsive main HTML page (`index.html`). The location of this file is *DITA-OT-DIR*`\plugins\com.oxygenxml.webhelp.responsive\xsl\mainFiles\createMainPage.xsl`

com.oxygenxml.webhelp.xsl.createNavLinks

Extension point to override the XSLT stylesheets that are used to generate navigation links in the WebHelp Responsive pages. These stylesheets can be found in the `navLinks` folder: *DITA-OT-DIR*`\plugins\com.oxygenxml.webhelp.responsive\xsl\navLinks\`

com.oxygenxml.webhelp.xsl.createSearchPage

Extension point to override the XSLT stylesheet (`createSearchPage.xsl`) that produces the WebHelp Responsive search HTML page (`search.html`). The location of this file is *DITA-OT-DIR*`\plugins\com.oxygenxml.webhelp.responsive\xsl\mainFiles\createSearchPage.xsl`

com.oxygenxml.webhelp.xsl.createIndexTermsPage

Extension point to override the XSLT stylesheet (`createIndextermsPage.xsl`) that produces the WebHelp Responsive index terms HTML page (`indexterms.html`). The location of this file is *DITA-OT-DIR*`\plugins\com.oxygenxml.webhelp.responsive\xsl\mainFiles\createIndextermsPage.xsl`

com.oxygenxml.webhelp.xsl.createTocXML

Extension point to override the XSLT stylesheet (`tocDita.xsl`) that produces the `toc.xml` file. This file contains information extracted from the *DITA map (on page 1324)* and it is mainly used to construct the WebHelp Table of Contents and navigational links. The path to this stylesheet is: *`DITA-OT-DIR`*`\plugins\com.oxygenxml.webhelp.responsive\xsl\navLinks\tocDita.xsl`.

XSLT-Parameter Extension Points

If your customization stylesheet declares one or more XSLT parameters and you want to control their values 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 *(on page 913)*.

com.oxygenxml.webhelp.xsl.createMainPage.param

Use this extension point to pass parameters to the stylesheet specified using the **com.oxygenxml.webhelp.xsl.createMainPage** extension point *(on page 913)*.

com.oxygenxml.webhelp.xsl.createNavLinks.param

Use this extension point to pass parameters to the stylesheet specified using the **com.oxygenxml.webhelp.xsl.createNavLinks** extension point *(on page 913)*.

com.oxygenxml.webhelp.xsl.createSearchPage.param

Use this extension point to pass parameters to the stylesheet specified using the **com.oxygenxml.webhelp.xsl.createSearchPage** extension point *(on page 913)*.

com.oxygenxml.webhelp.xsl.createIndexTermsPage.param

Use this extension point to pass parameters to the stylesheet specified using the **com.oxygenxml.webhelp.xsl.createIndexTermsPage** extension point *(on page 913)*.

com.oxygenxml.webhelp.xsl.createTocXML.param

Use this extension point to pass parameters to the stylesheet specified using the **com.oxygenxml.webhelp.xsl.createTocXML** extension point *(on page 914)*.

Related Information:

[DITA-OT] XSLT-Import Extension Points

[DITA-OT] XSLT-Parameter Extension Points

WebHelp Classic Output for DocBook

The **WebHelp Classic** variant is designed for desktop systems when feedback from users is not necessary and it is available for DocBook. The **WebHelp Classic with Feedback** variant also includes a feedback system

that allows your users to make comments and allows you to manage and reply to them. This section contains information about configuring a WebHelp Classic system and customizing the output.

This type of WebHelp system can be generated by using one of the following transformation scenarios:

- **DocBook WebHelp Classic** transformation scenario *(on page 686)*
- **DocBook WebHelp Classic with Feedback** transformation scenario *(on page 687)*

WebHelp Classic Output Layout and Features

Layout of the WebHelp Classic System Interface

The layout of the **WebHelp Classic** system consists 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 Output chapter *(on page 927)* 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 *(on page 918)*.

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 *(on page 918)*.

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

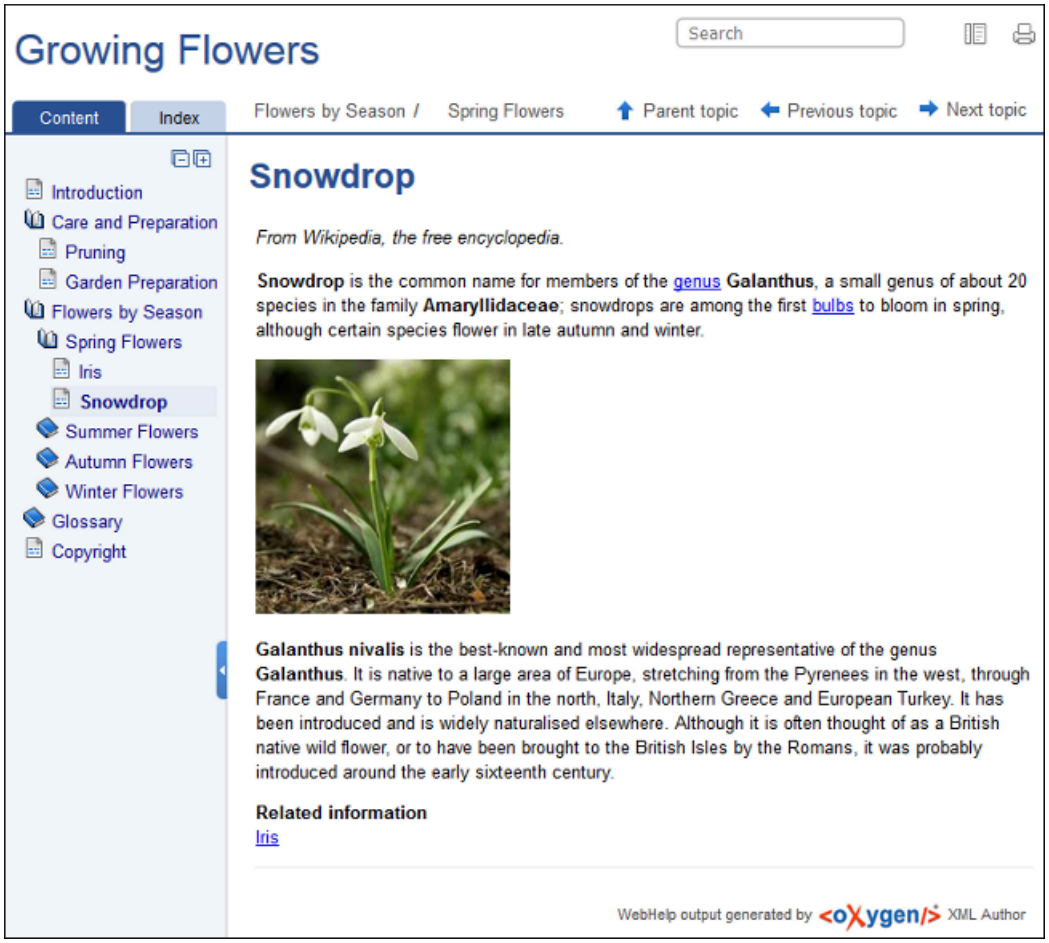
You can navigate through the content of your output using the navigation links or arrows in the upper-right part of the page. These arrows allow you to move to the **Parent topic**, **Previous topic**, or **Next topic**. Links to the parent topics of the currently open topic are also presented at the top of the page.

Tip: To hide the **Parent**, **Next**, and **Previous** links, you can edit the transformation scenario and set the value of the `args.hide.parent.link` parameter to `yes`.

Main Pane or Frame

The content of the help pages are rendered and displayed in this main section.

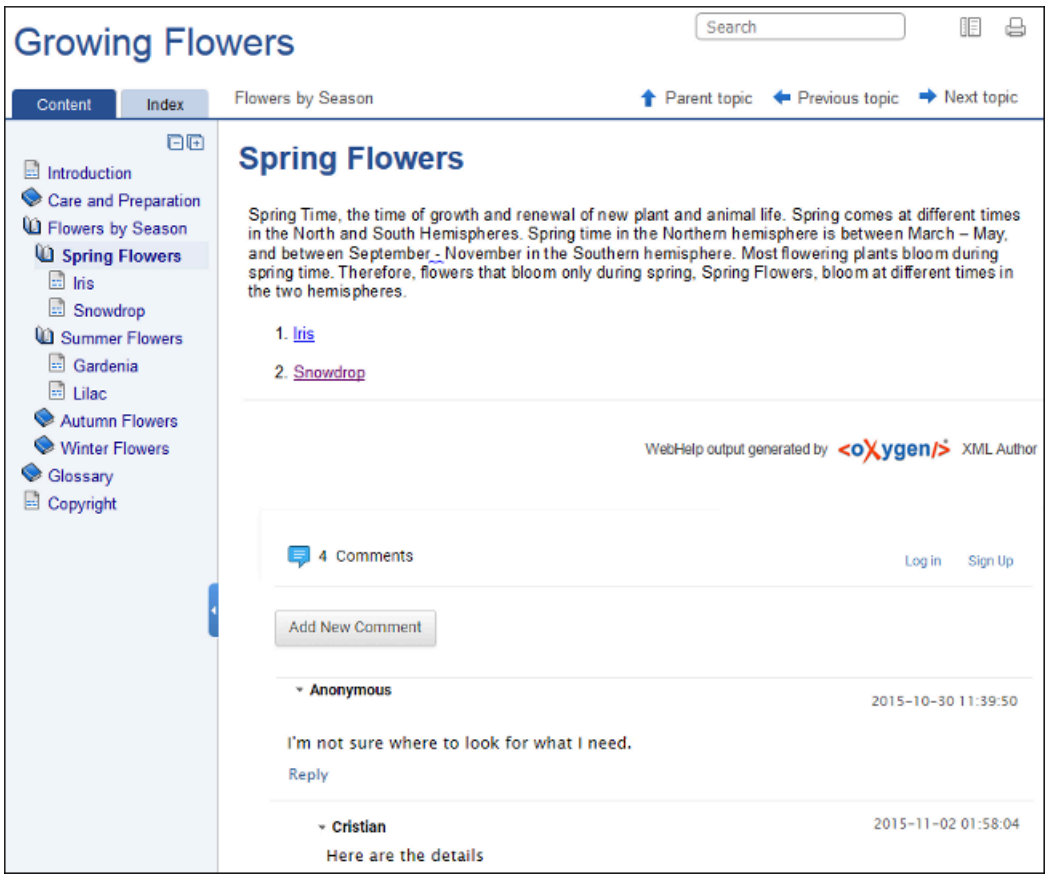
Figure 307. WebHelp Classic Output



WebHelp Classic with Feedback Enabled

The **WebHelp Classic with Feedback** variant also contains a **Comments** section at the bottom of each topic. This section is where you can interact with users through a comment system. For information about deploying a feedback-enabled system, see Deploying a Feedback-Enabled System.

Figure 308. WebHelp Classic with Feedback Comments Section



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 System.

WebHelp Classic Search Engine

Search Rules

Rules that are applied during a search include:

- You can use quotes to perform an exact search for multiple word phrases (for example, *"grow flowers"* will only return results if both words are found consecutively and exactly as they are typed in the search field). This type of search is known as a *phrase search*.
- *Boolean Search* is 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.
- Words composed by merging two or more words with colon (":"), minus ("-"), underline ("_"), or dot (".") characters count as a single word.
- Your search terms should contain two or more characters (note that stop words will be ignored). This rule does not apply to CJK (Chinese, Japanese, Korean) languages.
- When searching for multi-word phrases in CJK (Chinese, Japanese, Korean) languages that often have multiple words appear in strings without a space separator, you need to add a space to separate the words. Otherwise, WebHelp will not find results. For example, Chinese uses a specialized character for space separators, but the current WebHelp implementation cannot detect such specialized characters, so to search for **开始之前** (it translates as "before you begin" or "before start"), you have to enter **开始 之前** (notice the space between the second and third symbols) in the search field.

5-Star Rating Mechanism and Sorting

The **Search** feature is also enhanced with a rating mechanism that computes scores for every result that matches the search criteria. These scores are then translated into a 5-star rating scheme and the stars are displayed to the right of each result. The search results are sorted depending on the following:

- Search entries that satisfy the phrase search criterion are presented first.
- 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 term is included in a meta keyword.
    - The search term is in the title of the page.
    - The search term is in bold text in a paragraph.
    - The search term is in normal text in a paragraph.

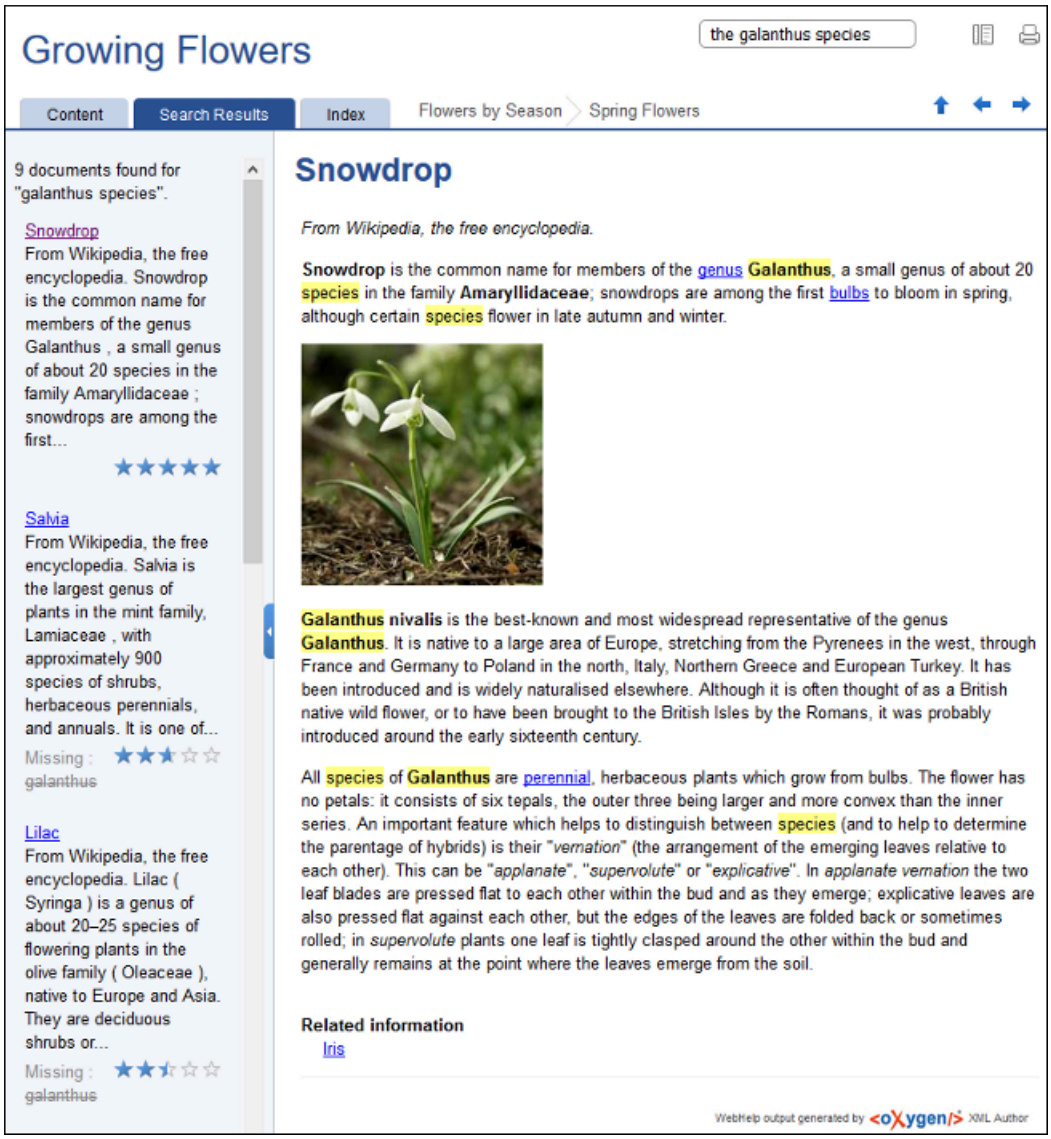
Excluded Terms

To improve performance, the **Search** feature excludes certain *stop words*. For example, the English version of the *stop words* includes: *a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the, their, then, there, these, they, this, to, was, will, with*.

WebHelp Classic Search Results Tab

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. When you click on a result in the **Search Results** tab, that result is displayed in the main pane with the search terms highlighted. If you press **Enter** with the **Search** field empty, the highlights are removed.

Figure 309. WebHelp Classic Search Results Tab



Missing Terms

If you enter multiple search terms (other than *stop words*), for any result that the search engine found at least one term but not one or more of the other terms, the **Missing** terms will be listed below each result.

Tag Element Scoring Values

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 How to Change Element Scoring in Search Results *(on page 938)*.

Browser Compatibility

This output format is compatible with the most recent versions of the following common browsers:

- Edge
- Internet Explorer (IE 11 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. It is recommended 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 regarding 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:///...`).

Deploying a Feedback-Enabled System

System Requirements

The feedback-enabled WebHelp system requires a standard server deployment. You can request this from your server administrator and it requires 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

The 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.

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. Click **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: Selecting 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, select 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 selected, 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, select 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. Click **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 System

Refreshing the Content of a Feedback-Enabled 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.

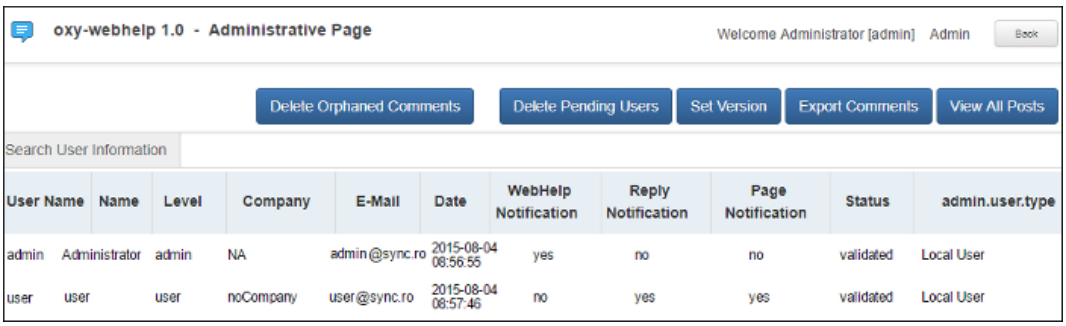
Managing Users and Comments in a Feedback-Enabled System

When you installed the **WebHelp with Feedback** system the first time (assuming the **Create new database structure** option was selected), 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.

Figure 310. 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 selected, the user receives notifications when comments are posted anywhere in your feedback-enabled WebHelp system.

Reply Notification

When selected, the user receives notifications when comments are posted as a reply to one of their comments.

Page Notification

When selected, 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).

Generating WebHelp Classic Output for DocBook

The publishing process can be initiated from a transformation scenario within **Oxygen XML Editor/Author** or from a command line outside **Oxygen XML Editor/Author**.

Running from Oxygen XML Editor/Author

To publish DocBook content to WebHelp Classic output from a transformation scenario inside **Oxygen XML Editor/Author**, use one of the following procedures, depending on whether or not you want a feedback section in your output.

WebHelp Classic Output

To publish a DocBook 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** or **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 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** or **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 System.

For more information about the feedback-enabled WebHelp system, watch our video demonstration:

https://www.youtube.com/embed/eoQ2uxHvppE

Automating the WebHelp Classic Output for DocBook

DocBook-based WebHelp output can be generated from an automated publishing process using a command line outside of **Oxygen XML Editor/Author**. **However, to do this, you must purchase an additional Oxygen XML WebHelp license**.

Related Information:

Generating WebHelp Classic Output for DocBook

Customizing WebHelp Classic Output

Oxygen XML WebHelp provides support for customizing the **WebHelp Classic** output to suit your specific needs. The **WebHelp Classic** type of output is designed for desktop systems and features a familiar *tri-pane* layout. You can use this system to publish DocBook documents. The **WebHelp Classic with Feedback** variant allows your users to add comments to your output.

To change the overall appearance of the **WebHelp Classic** output, you can use the visual WebHelp Skin Builder tool *(on page 927)*, 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.

Changing the Layout and Styles

This section contains some topics that explain how to customize the layout and style of your WebHelp Classic output using custom CSS, inserting custom HTML content, and more.

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** 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** website.
- A late version web browser.

To start the **Skin Builder**, use a web browser to go to https://www.oxygenxml.com/webhelp-skin-builder.

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.

Creating a Customization Skin

1. You can start with one of the built-in 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, click the **Reset** button.
3. When you are happy with the skin customizations you have made, click the **Export** button. All settings will be saved in a CSS file.

Apply a Customization Skin to a DocBook to WebHelp Classic Transformation Scenario

1. Start Oxygen XML Developer Eclipse 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.

For more information about using the WebHelp Skin Builder, watch our video demonstration:

https://www.youtube.com/embed/32PGX--PQx0

How to Use CSS Styling to Customize WebHelp Output

The most common way to customize the look and style of your 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 or script. This custom file will be the final CSS to be applied so its content will override the styles in the other pre-existing CSS files.

Using the CSS Inspector to Identify Content for Custom CSS File

You can use your browser's CSS inspector to identify the pertinent code in the current CSS files and you can even make changes directly in the CSS inspector to test the results so that you know exactly what content to use in your custom CSS file.

In most popular browsers (such as Chrome, Firefox, and Edge), you can access the CSS inspector by using **F12** or by selecting **Inspect Element** (or simply **Inspect**) from the contextual menu.

Tip: When using Safari on Mac OS X, you must first enable the Develop menu by going to the Advanced settings and selecting **Show Develop menu in menu bar**. Then you can select **Show Web Inspector** from the Develop menu or click **Command + Option + I**.

Referencing the Custom CSS Using Oxygen XML Editor/Author

To use a custom CSS to style WebHelp output and use a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Create your custom CSS file.
2. Edit the WebHelp transformation scenario and open the **Parameters** tab. Set the `html.stylesheet` parameter to the path of your custom CSS file.
3. Run the WebHelp transformation scenario to generate the output.

Referencing the Custom CSS Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To use a custom CSS to style WebHelp output and use a script outside of **Oxygen XML Editor/Author** *(on page 925)*, follow this procedure:

1. Create your custom CSS file.
2. Reference your custom CSS file. Use the `html.stylesheet` parameter in your transformation script and set its value to the path of your custom CSS file.
3. Execute the transformation script.

How to Add 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. 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.

Using Oxygen XML Editor/Author

To include custom HTML content in the WebHelp Classic output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Insert the HTML content in a well-formed XML file considering the following notes:

- **Well-Formedness** - If the file is not a *Well-formed XML* document *(on page 277)* (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.
- **Referencing Resources in the XML File** - You can include references to local resources (such as JavaScript or CSS files) by using the built-in `${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: How to Copy Additional Resources to Output Directory *(on page 937)*.
- **Inline JavaScript or CSS Content**:

  JavaScript:

```
<script type="text/javascript">
    /* Include JavaScript code here. */

    function myFunction() {
      return true;
    }
</script>
```

  CSS:

```
<style>
    /* Include CSS style rules here. */
    *{
      color:red
    }
</style>
```

  Note: If you have special characters (for example, `&`, `<`) that break the well-formedness of the XML fragment, it is important to place this content inside an XML comment:

```
<script type="text/javascript">
  <!--
    /* Include JavaScript code here. */
```

```
    function myFunction() {
      return true;
    }
  -->
</script>
```

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 reference the URL 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 and run the transformation scenario.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To include custom HTML content in the WebHelp Classic output using a script outside of **Oxygen XML Editor/Author** *(on page 925)*, follow this procedure:

1. Insert the HTML content in a well-formed XML file considering the following notes:
   - **Well-Formedness** - If the file is not a *Well-formed XML* document *(on page 277)* (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.
   - **Referencing Resources in the XML File** - You can include references to local resources (such as JavaScript or CSS files) by using the built-in `${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: How to Copy Additional Resources to Output Directory *(on page 937)*.

- **Inline JavaScript or CSS Content**:

  JavaScript:

```
<script type="text/javascript">
    /* Include JavaScript code here. */

    function myFunction() {
      return true;
    }
</script>
```

  CSS:

```
<style>
    /* Include CSS style rules here. */
    *{
      color:red
    }
</style>
```

  Note: If you have special characters (for example, `&`, `<`) that break the well-formedness of the XML fragment, it is important to place this content inside an XML comment:

```
<script type="text/javascript">
  <!--
    /* Include JavaScript code here. */

    function myFunction() {
      return true;
    }
  -->
</script>
```

2. Use the `webhelp.head.script` parameter in your transformation script and set its value to reference the URL 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.
3. Execute the transformation script.

Related Information:

How to Copy Additional Resources to Output Directory *(on page 937)*

How to Change Number Styles for Ordered Lists

Ordered lists (`<ol>`) are usually numbered in XHTML output using numerals. If you want to change the numbering to alphabetical, follow these steps:

1. Define a custom `@outputclass` value and set it as an attribute of the ordered list, as in the following example:

```
<ol outputclass="number-alpha">
    <li>A</li>
    <li>B</li>
    <li>C</li>
</ol>
```

2. Add the following code snippet in a custom CSS file:

```
ol.number-alpha{
    list-style-type:lower-alpha;
}
```

3. Edit the WebHelp transformation scenario and open the **Parameters** tab. Set the `html.stylesheet` parameter to the path of your custom CSS file
4. Run the transformation scenario.

How to Change 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, these icons 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;
}
```

Using Oxygen XML Editor/Author

To assign other icons and use a transformation scenario from within **Oxygen XML Editor/Author**, follow this 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 (*[OXYGEN_INSTALL_DIR]*`\frameworks\docbook\xsl\com.oxygenxml.webhelp.classic\oxygen-webhelp\resources\img\`).
3. Edit the WebHelp transformation scenario and open the **Parameters** tab. Set the `html.stylesheet` parameter to the path of your custom CSS file.
4. Run the WebHelp transformation scenario to generate the output.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To assign other icons and use a script outside of **Oxygen XML Editor/Author** *(on page 925)*, follow this 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 (`[DocBook XSL directory]\com.oxygenxml.webhelp.classic\oxygen-webhelp\resources\img\`).
3. Reference your custom CSS file. Use the `html.stylesheet` parameter in your transformation script and set its value to the path of your custom CSS file.
4. Execute the transformation script.

How to 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: `[DocBook XSL directory]\com.oxygenxml.webhelp.classic\oxygen-webhelp\resources\css`.
2. Edit that CSS file, find the `menuItemSelected` class, and change the value of the `background` property.
3. Run the transformation.

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 by using the `html.stylesheet` parameter and set its value to the path of your custom CSS file.

Adding Graphics and Media Resources

This section contains topics that explain how to add media resources to the published WebHelp Class output or to the output directory.

How to Add a Favicon in WebHelp Systems

You can add a custom *favicon* to your WebHelp output by simply using a parameter in the transformation scenario to point to your *favicon* image. This is available for DocBook WebHelp output using **WebHelp Classic** or **WebHelp Classic with Feedback** transformations.

Using Oxygen XML Editor/Author

To add a *favicon* to your WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

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.

Result: Browsers that provide *favicon* support will display the *favicon* (typically in the browser's address bar, in the list of bookmarks, and in the history).

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To add a *favicon* to your WebHelp output using a script outside of **Oxygen XML Editor/Author**, follow this procedure:

1. Specify the file path that points to the image that will be use as the *favicon* using the `webhelp.favicon` parameter.
2. Execute the transformation script.

Result: Browsers that provide *favicon* support will display the *favicon* (typically in the browser's address bar, in the list of bookmarks, and in the history).

How to Add a Logo Image in the Title Area

You can customize **WebHelp Classic** output to include a logo in the title area. It will be displayed before the publication title. You can also specify a URL that can be used to send users to a specific website when they click the logo image.

This customization can be done using a transformation scenario from within **Oxygen XML Editor/Author** or using a command-line script outside of **Oxygen XML Editor/Author**.

Using Oxygen XML Editor/Author

To add a logo in the title area of your WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Edit a **WebHelp Classic** 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.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To add a logo in the title area of your WebHelp output using a script outside of **Oxygen XML Editor/Author**, follow this procedure:

1. Specify the path to your logo using the `webhelp.logo.image` parameter.
2. If you also want to add a link to your website when you click the logo image, set the URL using the `webhelp.logo.image.target.url` parameter.
3. Execute the transformation script.

How to Add 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.

How to Copy Additional Resources to Output Directory

You can copy additional resources (such as JavaScript, CSS or other resources) to the output directory of a WebHelp system by using the `webhelp.custom.resources` parameter.

Using Oxygen XML Editor/Author

To copy additional resources to the output directory using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Place all your resources in the same directory.
2. Edit the WebHelp transformation scenario, then open the **Parameters** tab.
3. Edit the value for the `webhelp.custom.resources` parameter and set it to the absolute path of the directory in step 1.
4. Click **OK** to save the changes to the transformation scenario.
5. Run the transformation scenario.

Result: All files from the new directory will be copied to the root of the WebHelp output directory.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To copy additional resources to the output directory using a script outside of **Oxygen XML Editor/Author** *(on page 925)*, follow this procedure:

1. Place all your resources in the same directory.
2. Specify the absolute path to that directory using the `webhelp.custom.resources` parameter.
3. Execute the transformation script.

Result: All files from the new directory will be copied to the root of the WebHelp output directory.

How to Add MathML Equations in WebHelp Output

Currently, only **Firefox** can render **MathML** equations embedded in the **HTML** code. MathJax is a solution to properly view MathML equations embedded in **HTML** content in a variety of browsers.

If you have DocBook content that has embedded **MathML** equations and you want to properly view the equations in published HTML output types (such as WebHelp), you need to add a reference to the MathJax script in the **head** element of all HTML files that have the equation embedded.

For example:

```
<script type="text/javascript"


 src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.1/MathJax.js?config=TeX-AMS-MML_HTMLorMML"
>
</script>
```

Result: The equation should now be properly rendered in the WebHelp output for other browsers.

Searching the Output

This section contains topics that explain how to customize some of the search features in WebHelp Classic output.

How to Change Element Scoring 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. The WebHelp directory 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 DocBook WebHelp systems (`[DocBook XSL directory]\com.oxygenxml.webhelp.classic\indexer\scoring.properties`). The properties file includes instructions and examples to help you with your customization.
   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 transformation.

How to Index Japanese Content in WebHelp Classic

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 Editor/Author** installation kit.

Using Oxygen XML Editor/Author

To activate the Japanese indexing in your WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Set the language for your content to Japanese. 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 WebHelp transformation scenario to generate the output.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To activate the Japanese indexing in your WebHelp output using a script outside of **Oxygen XML Editor/Author**, follow this procedure:

1. Set the language for your content to Japanese. Use the `l10n.gentext.default.language` parameter in your transformation script and set its value to `ja`.

2. Execute the transformation script.

Related Information:

How to Localize the Interface of DocBook to WebHelp Classic Output *(on page 941)*

Localization in WebHelp Classic Output

This section contains topics that explain the localization support for DocBook WebHelp Classic transformations.

How to Localize Email Notifications of WebHelp Classic with Feedback Output

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.

Using Oxygen XML Editor/Author

To change the language of the emails for WebHelp Classic with Feedback output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Create the following directory: `[DocBook XSL directory]\com.oxygenxml.webhelp.classic\oxygen-webhelp\resources\php\templates\[Language code]` (where the *Language code* is the 2-digit code, such as *nl* for Dutch).
2. Copy all English template files from the following directory and paste them into the directory you just created: `[DocBook XSL directory]\com.oxygenxml.webhelp.classic\oxygen-webhelp\resources\php\templates\en`.
3. Edit the HTML files from the following directory and translate the content into your language: `[DocBook XSL directory]\com.oxygenxml.webhelp.classic\oxygen-webhelp\resources\php\templates\[Language code]` (where the *Language code* is the 2-digit code, such as *nl* for Dutch).
4. Edit a **WebHelp Classic with Feedback** transformation scenario and open the **Parameters** tab.
5. In the **Parameters** tab, look for the `default.language` parameter and set its value to the appropriate language code (for example, `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.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To change the language of the emails for WebHelp Classic with Feedback output using a script outside of **Oxygen XML Editor/Author** *(on page 940)*, follow this procedure:

1. Create the following directory: `[DocBook XSL directory]\com.oxygenxml.webhelp.classic\oxygen-webhelp\resources\php\templates\[Language code]` (where the *Language code* is the 2-digit code, such as *nl* for Dutch).
2. Copy all English template files from the following directory and paste them into the directory you just created: `[DocBook XSL directory]\com.oxygenxml.webhelp.classic\oxygen-webhelp\resources\php\templates\en`.
3. Edit the HTML files from the following directory and translate the content into your language: `[DocBook XSL directory]\com.oxygenxml.webhelp.classic\oxygen-webhelp\resources\php\templates\[Language code]` (where the *Language code* is the 2-digit code, such as *nl* for Dutch).
4. Use the `args.default.language` parameter in your transformation script and set its value to the appropriate language code (for example, `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.
5. Execute the transformation script.

How to Localize the Interface of DocBook to WebHelp Classic Output

Static labels that are used in the WebHelp output are kept in translation files in the `[DocBook XSL directory]/com.oxygenxml.webhelp.classic/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 `[DocBook XSL directory]/com.oxygenxml.webhelp.classic/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 `[DocBook XSL directory]/com.oxygenxml.webhelp.classic/oxygen-webhelp/resources/localization/strings.xml`. For example, a Canadian French file would 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) 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:

How to Index Japanese Content in WebHelp Classic *(on page 939)*

How to Activate Support for Right-to-Left (RTL) Languages

To activate support for RTL (right-to-left) languages in WebHelp output, set the `@xml:lang` attribute with the corresponding attribute value:

- **ar-eg** - Arabic
- **he-il** - Hebrew
- **ur-pk** - Urdu

Integrating Social Media and Google Tools in the WebHelp Classic Output

Oxygen XML Developer Eclipse 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

It is possible to integrate Facebook™ into your **WebHelp Classic** output and the widget will appear in the footer sections of your WebHelp page.

Using Oxygen XML Editor/Author

To add a Facebook™ *Like* widget to your WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Go to the Facebook Developers website.
2. Fill-in the displayed form, then click the **Get Code** button.
3. Copy the two code snippets and paste them into a `<div>` element inside an XML file called `facebook-widget.xml`. Make sure you follow these rules:
   - The file must be well-formed.
   - The code for each `<script>` element must be included in an XML comment.
   - The start and end tags for the XML comment must be on a separate line.

   The content of the XML file should look like this:

```
<div id="facebook">
    <div id="fb-root"/>
    <script>
        <!--
```

```
            (function(d, s, id) {
        var js, fjs = d.getElementsByTagName(s)[0];
        if (d.getElementById(id)) return;
        js = d.createElement(s); js.id = id;
        js.src = "//connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.0";
        fjs.parentNode.insertBefore(js, fjs);
        }(document, 'script', 'facebook-jssdk'));
        -->
    </script>
    <div class="fb-like" data-layout="standard" data-action="like"
        data-show-faces="true" data-share="true"/>
</div>
```

4. In **Oxygen XML Editor/Author**, click the **Configure Transformation Scenario(s)** action from the toolbar.
5. Select an existing WebHelp Classic 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 and edit the `webhelp.footer.file` parameter to reference the `facebook-widget.xml` file that you created earlier.
7. Click **Ok** and run the transformation scenario.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To add a Facebook™ *Like* widget to your WebHelp output using a script outside of **Oxygen XML Editor/Author** *(on page 925)*, follow this procedure:

1. Go to the Facebook Developers website.
2. Fill-in the displayed form, then click the **Get Code** button.
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. Use the `webhelp.footer.file` parameter in your transformation script and set its value to reference the `facebook-widget.xml` file that you created earlier.
5. Execute the transformation script.

How to Add Tweet Button in WebHelp Classic Output

It is possible to integrate *Twitter* into your **WebHelp Classic** output and the widget will appear in the footer section of your WebHelp page.

Using Oxygen XML Editor/Author

To add a Twitter™ *Tweet* widget to your WebHelp Classic output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Go to the Tweet button generator page.
2. Fill in the displayed form. The **Preview and code** area displays the code that you will need.
3. Copy the code snippet displayed in the **Preview and code** area and paste it into a `<div>` element inside an XML file called `tweet-button.xml`. Make sure you follow these rules:
   - The file must be well-formed.
   - The code for each `<script>` element must be included in an XML comment.
   - The start and end tags for the XML comment must be on a separate line.

   The content of the XML file should look like this:

```
<div id="twitter">
  <a href="https://twitter.com/share" class="twitter-share-button">Tweet</a>
  <script>
    <!--
      !function (d, s, id) {
      var
      js, fjs = d.getElementsByTagName(s)[0], p = /^http:/.test(d.location)
 ? 'http': 'https';
```

```
      if (! d.getElementById(id)) {
      js = d.createElement(s);
      js.id = id;
      js.src = p + '://platform.twitter.com/widgets.js';
      fjs.parentNode.insertBefore(js, fjs);
      }
      }
      (document,
      'script', 'twitter-wjs');
     -->
  </script>
</div>
```

4. In **Oxygen XML Editor/Author**, 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) 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** and run the transformation scenario.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To add a Twitter™ *Tweet* widget to your WebHelp Classic output using a script outside of **Oxygen XML Editor/Author** *(on page 925)*, follow this procedure:

1. Go to the Tweet button generator page.
2. Fill in the displayed form. The **Preview and code** area displays the code that you will need.
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. Use the `webhelp.footer.file` parameter in your transformation script and set its value to reference the `tweet-button.xml` file that you created earlier.
5. Execute the transformation script.

How to Integrate Google Analytics in WebHelp Classic Output

You can use *Google Analytics* to track and report site data for your **WebHelp Classic** output.

Using Oxygen XML Editor/Author

To integrate *Google Analytics* into your WebHelp Classic output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Create a new Google Analytics account (if you do not already have one) and log on.
2. Choose the Analytics solution that best fits the needs of your website.
3. Follow the on-screen instructions to obtain a **Tracking Code** that contains your *Tracking ID*. A **Tracking Code** looks like this:

```
<script>
 (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
 (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
 m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
 })(window,document,'script','//www.google-analytics.com/analytics.js','ga');
 ga('create', 'UA-XXXXXXXX-X', 'auto');
 ga('send', 'pageview');
</script>
```

4. Save the Tracking Code (obtained in the previous step) in a new XML file called `googleAnalytics.xml`. Note that the file should only contain the tracking code.
5. In **Oxygen XML Editor/Author**, 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) 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** and run the transformation scenario.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To integrate *Google Analytics* into your WebHelp Classic output using a script outside of **Oxygen XML Editor/Author** *(on page 925)*, follow this procedure:

1. Create a new Google Analytics account (if you do not already have one) and log on.
2. Choose the Analytics solution that best fits the needs of your website.
3. Follow the on-screen instructions to obtain a **Tracking Code** that contains your *Tracking ID*. A **Tracking Code** looks like this:

```
<script>
 (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
 (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
 m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
 })(window,document,'script','//www.google-analytics.com/analytics.js','ga');
 ga('create', 'UA-XXXXXXXX-X', 'auto');
 ga('send', 'pageview');
</script>
```

4. Save the Tracking Code (obtained in the previous step) in a new XML file called `googleAnalytics.xml`. Note that the file should only contain the tracking code.
5. Use the `webhelp.footer.file` parameter in your transformation script and set its value to reference the `googleAnalytics.html` file that you created earlier.
6. Execute the transformation script.

How to Integrate Google Search in WebHelp Classic Output

It is possible to integrate the *Google Search Engine* into your **WebHelp Classic** output and you can specify where you want the results to appear in your WebHelp page.

Using Oxygen XML Editor/Author

To integrate the *Google Search Engine* into your WebHelp output using a transformation scenario from within **Oxygen XML Editor/Author**, follow this procedure:

1. Go to the Google Custom Search Engine page using your Google account.
2. Select 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 that looks like this:

```
<script>
 (function()  {
  var cx =
    '000888210889775888983:8mn4x_mf-yg';
  var gcse = document.createElement('script');
  gcse.type = 'text/javascript';
  gcse.async = true;
  gcse.src = (document.location.protocol == 'https:' ?
    'https:' : 'http:') +         '//www.google.com/cse/cse.js?cx=' + cx;
  var s = document.getElementsByTagName('script')[0];
  s.parentNode.insertBefore(gcse, s);
 }
 )();
</script>
```

4. Save the script into a well-formed HTML file called `googlecse.html`.
5. In **Oxygen XML Editor/Author**, 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** and run the transformation scenario.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To integrate the *Google Search Engine* into your WebHelp Classic output using a script outside of **Oxygen XML Editor/Author** *(on page 925)*, follow this procedure:

1. Go to the Google Custom Search Engine page using your Google account.
2. Select 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 that 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. Use the `webhelp.google.search.script` parameter in your transformation script and set its value to reference the `googlecse.html` file that you created earlier.
6. 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>`.

7. Execute the transformation script.

Miscellaneous Customization Topics

This section contains miscellaneous topics about how to customize the WebHelp Classic output.

How to 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 file: *[OXYGEN_INSTALL_DIR]*`\frameworks\docbook\xsl\com.oxygenxml.webhelp.classic\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 transformation scenario.

How to Publish 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.

Using Oxygen XML Editor/Author

To publish WebHelp Classic output on a SharePoint site and use a transformation scenario from within **Oxygen XML Editor/Author**, 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. Edit the WebHelp transformation scenario and open the **Parameters** tab. Set the `html.ext` parameter to `.aspx`.
3. Run the WebHelp transformation scenario to generate the output.

Using a Script Outside of Oxygen XML Editor/Author

Important: Running WebHelp transformations from a script outside of **Oxygen XML Editor/Author** requires an additional license and some additional setup:

- You must have a valid license for the **Oxygen XML WebHelp Plugin** (https://www.oxygenxml.com/buy_webhelp.html).
- The **Oxygen XML WebHelp Plugin** must be installed and integrated.

To publish WebHelp Classic output on a SharePoint site and use a script outside of **Oxygen XML Editor/Author** *(on page 925)*, 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. Use the `html.ext` parameter in your transformation script and set its value to `.aspx`.
3. Execute the transformation script.

DITA to PDF Output Customization

Oxygen XML Developer Eclipse plugin provides support for generating PDF output using transformation scenarios for certain types of documents (for example, DITA, DocBook, TEI, and JATS) and Oxygen XML Developer Eclipse plugin supports several different types of processors. There are numerous ways to customize the published output to fit your specific needs.

CSS-based DITA to PDF Customization

Oxygen XML Developer Eclipse plugin comes bundled with a **DITA-OT CSS-based PDF Publishing Plugin** for transforming DITA maps or single topics to PDF, while styling the resulting output using CSS. It is the base of two types of transformation scenarios:

DITA Map Transformation Type (DITA Map PDF - based on HTML5 & CSS)

This transformation type converts DITA maps to PDF using a CSS-based processing engine and HTML5 as an intermediate format. For this transformation, the `pdf-css-html5` transtype is used. Because the structure of the

HTML5 intermediate format resembles the one used in WebHelp output, it is possible to reuse parts of your CSS file you developed for a WebHelp customization.

Single Topic Transformation Type (DITA PDF - based on HTML5 & CSS)

This transformation type converts a single DITA topic to PDF using a CSS-based processing engine and HTML5 as an intermediate format. For this transformation, the `pdf-css-html5-single-topic` transtype is used. This transformation is derived from the DITA Map PDF - based on HTML5 & CSS transformation type but applies on a single topic.

Related Information:

DITA Map PDF - based on HTML5 & CSS Transformation *(on page 679)*

DITA PDF - based on HTML5 & CSS Transformation

Overview

This section contains topics that provide a basic overview of the **DITA-OT CSS-based PDF Publishing Plugin**, technical details, and some additional resources to help you with your customizations.

Resources

Customizing the PDF output requires knowledge of CSS, Paged Media, and DITA. The following list provides some resources to help you:

- **CSS** - You can find a good tutorial here: https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS. Also, the specification is available on the W3C website: https://www.w3.org/Style/CSS/Overview.en.html.
- **CSS Paged Media** - This is a part of the CSS specification that shows how to organize your publication in pages, how to use headers/footers, page breaks, and other page-related issues. The specification is available here: https://www.w3.org/TR/CSS2/page.html. Also, there is a set of hands-on examples in the **Oxygen PDF Chemistry** user guide: https://www.oxygenxml.com/doc/ug-chemistry/.
- **DITA** - You will need a basic understanding of DITA elements, attributes, and structure. A good resource is *The DITA Style Guide - Best Practices for Authors* by Tony Self. It is available at: www.ditastyle.com and: https://www.oxygenxml.com/dita/styleguide/webhelp-feedback/Artefact/c_DITA_Authoring_Concepts.html.
- **HTML5** - You will need a good knowledge of HTML5. You can find resources here: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

Related Information:

DITA-OT DAY 2017: Using CSS to Style PDF Output

Supported Processors

The **DITA-OT CSS-based PDF Publishing Plugin** supports the following CSS processors:

- **Oxygen PDF Chemistry** - This is recommended processor because the built-in CSS files were fine-tuned for this processor. For example, metadata extraction *(on page 1008)* only functions with this processor. If the plugin is started from an **Oxygen XML Editor/Author** distribution, a Chemistry installation is not needed.
- **Prince XML** - A commercial product, available at: https://www.princexml.com/.
- **Antenna House** - A commercial product, available at: https://www.antennahouse.com/antenna1/formatter/.

Technical Details

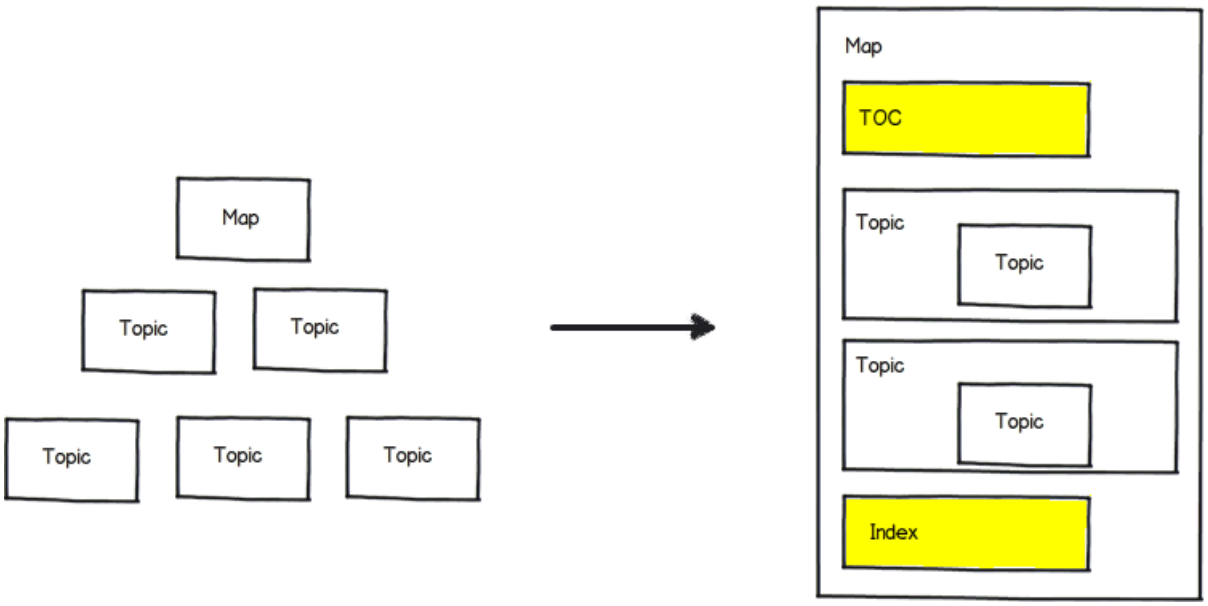
The **DITA-OT CSS-based PDF Publishing Plugin** comes bundled in the **Oxygen XML Editor/Author** distributions. The plugin ID is: **com.oxygenxml.pdf.css**. It is installed in the `[OXYGEN-INSTALL-DIR]frameworks/dita/DITA-OT3.x/plugins/com.oxygenxml.pdf.css` folder.

It has the following transformation types:

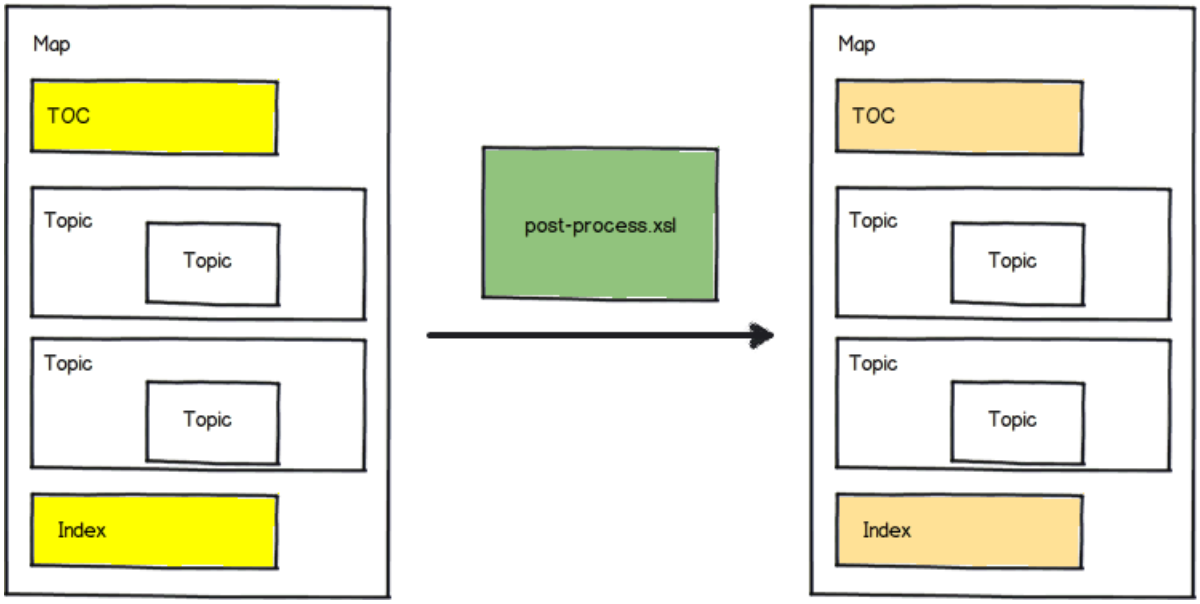
- **pdf-css-html5** (*DITA Map PDF - based on HTML5 & CSS* transformation) - CSS styling applied over a merged HTML5 document (the merged DITA map converted to HTML5).
- **pdf-css-html5-single-topic** (*DITA PDF - based on HTML5 & CSS* transformation) - CSS styling applied over a merged HTML5 document (the merged DITA topic converted to HTML5).

This is how it works:

1. It expands all the topic references into a temporary clone of the map, resolving keys and reused content. For the single topic transformation the result is a file with the keys and content resolved.
2. It generates a structure for the table of contents and index. The result is a merged map with all the references resolved. When transforming a single topic, the TOC and Index are not added to the merged file, this includes only the contents of the topic.



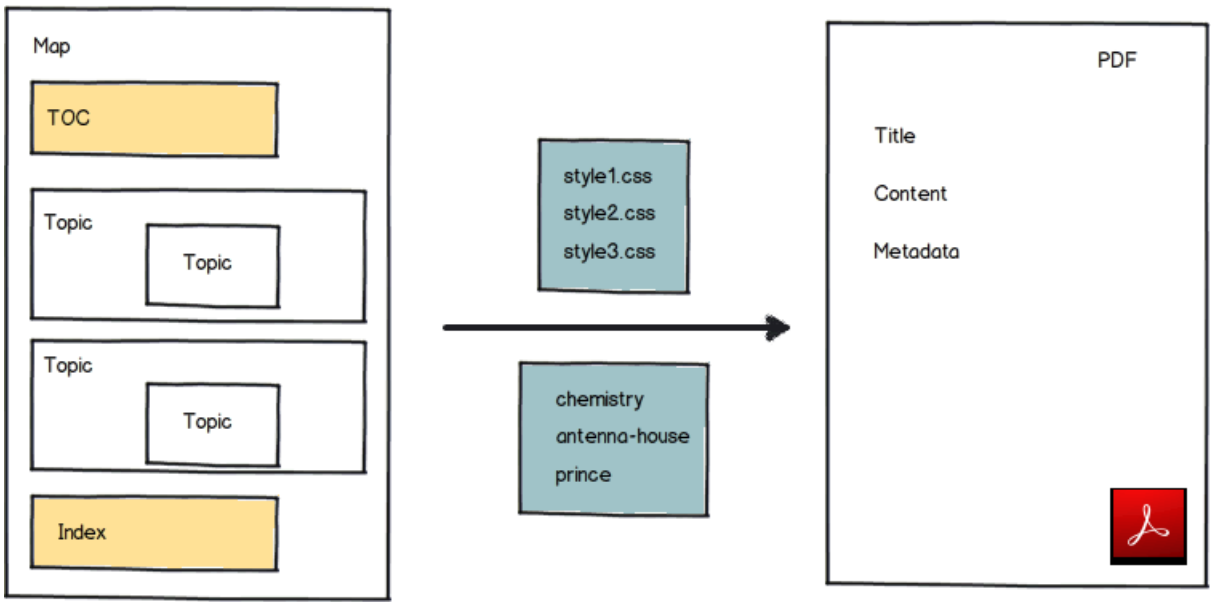
3. Then it post-processes the merged map. It fixes some of the structure in the TOC and index, moves the *frontmatter* and *backmatter* to the correct places, transforms any change tracking and review processing instructions to elements that can be styled later, etc. The result is another merged map.



Note: In the single topic transformation type (**DITA PDF - based on HTML5 & CSS**), these steps are simplified.

4. The merged DITA map or topic is transformed to a single HTML5 file.

   The generated HTML elements copy the class attributes from the original DITA elements. In this way you can either use selectors that were designed for DITA structure, or ones for the HTML structure - see: Reusing the Styling for WebHelp and PDF Output *(on page 1083)*. This step can apply customization XSLT extension points *(on page 972)* that change the default processing.
5. In the last phase, it uses a collection of CSS files to style the merged map. References to the CSS files are collected from the publishing template *(on page 963)*.



Increasing Memory Allocation for Java

If you are working with a large project with extensive metadata or key references, you may need to increase the amount of memory that is allocated to the Java process that performs the publishing.

When the Transformation is Started from Oxygen

To alter the memory allocation setting from the transformation scenario, follow these steps:

1. Open the **Configure Transformation Scenario(s)** dialog box.
2. Select your transformation scenario, then click **Edit**.
3. Go to the **Advanced** tab.
4. Locate the **JVM Arguments** and increase the default value. For instance, to set 1GB as the maximum amount of memory, you can specify: `-Xmx1024m`.

Note: This memory setting is used by both the DITA-OT process and the Chemistry CSS processor.

When the Transformation is Started from the Command Line

There can be two situations where an out of memory error can be triggered:

- From the DITA-OT basic processing (the preparation of the merged XML document).
- From the Chemistry PDF CSS processor (the transformation of the merged XML document to PDF).

To solve both of them, you can change the value of the `ANT_OPTS` environment variable from a command line for a specific session.

Example: To increase the JVM memory allocation to 1024 MB for a specific session, issue the following command from a command prompt (depending on your operating system):

- **Windows**

```
set ANT_OPTS=%ANT_OPTS% -Xmx1024M
```

- **Linux/Mac OS X**

```
export ANT_OPTS=$ANT_OPTS -Xmx1024M
```

Tip: To persistently change the memory allocation, change the value allocated to the `ANT_OPTS` environment variable on your system.

If the Chemistry PDF CSS processor fails with an **Out Of Memory Error**, try adding the `baseJVMArgLine` parameter to the DITA-OT command line. For example:

```
-DbaseJVMArgLine=-Xmx984m
```

Transformation Parameters

This list includes the most common customization parameters that are available in the **DITA Map PDF - based on HTML5 & CSS** transformation scenario. Other standard DITA-OT parameters were omitted for clarity, but they are still supported.

`args.allow.external.coderefs`	Enables the inclusion of code files that are located outside the DITA map folder hierarchy, referenced using the DITA `<coderef>` element. Allowed values are **yes** or **no** (default).
`args.chapter.layout`	Specifies whether chapter-level TOCs are generated for bookmaps. When set to **MINITOC**, a small section with links is added at the beginning of each chapter. The default is **BASIC**. For details, see: Table of Contents for Chapters (Mini TOC) *(on page 1041)*. Allowed values: • **BASIC** - No chapter TOC is created. • **MINITOC** - A chapter-level TOC will be generated. • **MINITOC-BOTTOM-LINKS** - A chapter-level TOC will be generated, with the links under the chapter description.
`args.css`	You can use this to specify a list of CSS URLs to be used in addition to those specified in the **dita.css.list** parameter or publishing template. The files must have URL syntax and be separated using semicolons.
`args.css.param.*`	You can use this parameter pattern to set attributes on the root of the merged map. This means you can activate specific CSS rules from your custom CSS using custom attributes. For examples, see: Styling Through Custom Parameters *(on page 1101)*.
`args.css.param.numbering`	You can use this parameter to change the numbering of the first-level topics (chapters) and nested topics. Allowed values:

	• **shallow** - Only the topics from the first level will be numbered (chapters). This is the default. • **deep** - All the topics from the map will be numbered (nested topics up to level 3). • **deep-chapter-scope** - Similar to deep, but in addition, the page numbers, figures, and table numbers are reset at the start of each first-level topic (chapter). The table and figure titles (and the links to them) are prefixed with the chapter numbers. The generic cross reference links contain both the first-level topic (chapter) numbers and the page numbers to avoid ambiguity. This parameter value is only available for the **DITA Map PDF - based on HTML5 & CSS** transformation scenario. • **deep-chapter-scope-no-page-reset** - Similar to `deep-chapter-scope`, but the page numbers do not reset at the start of each first-level topic (chapter). The generic cross reference links contain only the page number. This parameter value is only available for the **DITA Map PDF - based on HTML5 & CSS** transformation scenario. For more details, see Numbering Types *(on page 1031)*.
`args.css.param.show-on-page-lbl`	Controls whether or not the links will have an `on page NN` label after them. This parameter has different defaults, depending on the transformation type. For map transformations (`pdf-css-html5` trans type), the default is **yes**. For topic transformations (`pdf-css-html5-single-topic` trans type), the default is **no**.
`args.css.param.title.layout`	Changes the structure of the title element. In the output, the title area consists of two parts: one is the number of the chapter (and optionally, the sections number), and one is the title text. This parameter allows a switch between normal text flow (in-line flow) and a table layout where the number is placed in one cell and the text in the other (to avoid wrapping text under the chapter number). • **normal** • **table** (avoid wrapping text under counter)
`args.draft`	Specifies whether or not the content of `<draft-comment>` and `<required-cleanup>` elements is included in the output. Allowed values: • **no** (default) - No draft information is shown in the output. • **yes** - The draft information is shown in the output.

`args.figurelink.style`	Specifies how cross references to figures are styled in output. Allowed values: • **NUMBER** - Only the number of the figures will be shown in links. • **TITLE** - Only the title of the figures will be shown in links. • **NUMTITLE** (default) - Both the title and number of the figures will be shown in links.
`args.gen.task.lbl`	Specifies whether or not to generate headings for sections within task topics. Allowed values: **YES** or **NO** (default). When set to **YES**, headings such as "About this task", "Before you begin", "Procedure", or "What to do next", are shown in the task contents.
`args.hyph.dir`	Specifies the directory that contains custom hyphenation dictionaries. Fore more details see: Hyphenation *(on page 1065)*.
`args.input`	Specifies the master DITA map file for your documentation project.
`args.keep.output.debug.files`	Specifies whether or not the debug files generated during the transformation should be kept in the output folder. Allowed values: **YES** (default) or **NO**.
`args.output.base`	Specifies the name of the output file without a file extension. By default, the name of the PDF file is derived from the name of the DITA map file. This parameter allows you to override it.
`args.tablelink.style`	Specifies how cross references to tables are styled in output. Allowed values: • **NUMBER** - Only the number of the tables will be shown in links. • **TITLE** - Only the title of the tables will be shown in links. • **NUMTITLE** (default) - Both the title and number of the tables will be shown in links.
`clean.temp`	Specifies whether or not the DITA-OT deletes the files in the temporary directory after it finishes a build. Allowed values: **yes** (default) or **no**
`css.processor.path.antenna-house`	Path to the Antenna House executable file that needs to be run to generate the PDF (for example, `C:\path\to\AHFCmd.exe` on Windows).
`css.processor.path.chemistry`	Path to the **Oxygen PDF Chemistry** executable file that needs to be run to generate the PDF (for example, `C:\path\to\chemistry.bat` on Windows). If this parameter is not set, the plugin will use the system's PATH environment variable to locate and start **Oxygen PDF Chemistry**.

`css.processor.path.prince`	Path to the Prince executable file that needs to be run to generate the PDF (for example, `C:\path\to\prince.exe` on Windows).
`css.processor.type`	Specifies the processor to use for the transformation. Allowed values: **chemistry** (default), **antenna-house**, or **prince**.
`default.language`	Specifies the default language for source documents. Examples: **fr**, **de**, **zh**, etc. Depending on the transformation type, the actual number of supported languages can vary, see: Localization *(on page 1073)*.
`drop.block.margins.at.page-.boundary`	Specifies that the top and bottom margins associated with a block element should be discarded when the block is at the top or bottom of the page. Allowed values: **YES** (default) or **NO**.
`editlink.ditamap.edit.url`	Use this parameter to add an *Edit* link next to the topic title in the WebHelp output. When a user clicks the link, the topic is opened in **Oxygen XML Web Author** or **Content Fusion** where they can make changes that can be saved to a file server. The value should be set as the edit URL of the *main DITA map* used for publishing your output. The easiest way to obtain the URL is to open the map in Web Author or Content Fusion and copy the URL from the browser's address bar.
`editlink.additional.query.parameters`	You can use this optional parameter to add additional parameters to be appended to each generated edit link. Each parameter must start with *&* (for example: *&tags-mode=no-tags*).
`editlink.remote.ditamap.url` (deprecated)	Use this parameter in conjunction with `editlink.web.author.url` to add an *Edit* link next to the topic title in the PDF output. When a user clicks the link, the topic is opened in **Oxygen XML Web Author** where they can make changes that can be saved to a file server. The value should be set as the custom URL of the *main DITA map*. For example, a GitHub custom URL might look like this: `https://getFileContent/oxyengxml/userguide/master/UserGuide.ditamap`.
`editlink.web.author.url` (deprecated)	This parameter needs to be used in conjunction with `editlink.remote.ditamap.url` to add an *Edit* link next to the topic title in the PDF output. When a user clicks the link, the topic is opened in **Oxygen XML Web Author** where they can make changes that can be saved to a file server. The value should be set as the URL of the Web Author installation. For example: `https://www.oxygenxml.com/oxygen-xml-web-author/`.
`figure.title.placement`	Controls the title placement of the figures, relative to the image. Possible values include **top** (default) and **bottom**.
`fix.external.refs.com.oxygenxml`	The DITA Open Toolkit usually has problems processing references that point to locations outside of the processed DITA map directo-

	ry. 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).
`generate.named.destinations`	When set to **yes** (default), the `@id` attributes from the DITA content are used as PDF named destinations
`hide.frontpage.toc.index-.glossary`	When set to **yes**, the generated structures (table of contents, index list, front page, etc.) are removed from the output. The default is **no**.
`show.changes.and.comments`	When set to **yes**, the user comments, colored highlights and tracked changes are shown in the output.
`show.changes.and.comments.as-.changebars`	When set to **yes** (default) and the `show.changes.and.comments` parameter is also set to **yes**, the user comments and tracked changes are shown as change bars in the PDF output. This parameter can be used in conjunction with the `show.changes.and.comments.as.pdf.sticky-.notes` parameter to choose whether the change bars are displayed in footnotes or sticky notes. You can override this from your customization CSS *(on page 981)*.
`show.changes.and.comments.as-.pdf.sticky.notes`	When set to **yes** (default) and the `show.changes.and.comments` parameter is also set to **yes**, the user comments and tracked changes are shown in the PDF output as sticky note annotations. When set to **no**, the comments and tracked changes are left in the document model and are styled by the default CSS rules as footnotes. You can override this from your customization CSS *(on page 981)*.
`show.changed.text.in.pdf-.sticky.notes.content`	When set to **yes** (default) and both the `show.changes.and.comments` and `show.changes.and.comments.as.pdf.sticky.notes` parameters are also set to **yes**, the inserted and deleted text is shown in the sticky note annotations. When set to **no**, only the *inserted* and *deleted* labels are shown in the annotations (this is useful for search scope).
`table.title.placement`	Controls the placement of the title for tables. Possible values include **top** (default) and **bottom**.
`table.title.repeat`	Specifies if a table caption should repeat on other pages when the table spans on multiple pages. Allowed values are **yes** (default) or **no**.
`use.css.for.embedded.svg`	When set to **yes** (default), the CSS files specified in the publishing template or by the `args.css` parameter are also applied on embedded SVG elements. Allowed values are **yes** and **no**.

The following parameters can be used to specify a publishing template:

`pdf.publishing.template`	Specifies the path to the folder containing the custom PDF template.
`pdf.publishing.template.descriptor`	Specifies the name of the descriptor file to be loaded from the PDF template folder or package. If it is not specified, the first encountered descriptor file will be loaded.

The following parameter is available on all DITA transformations when using the **Oxygen Publishing Engine**:

`args.disable.security.checks`	Specifies whether or not to load external entities that are not solved through catalogs. For security reasons, the default is **no**. Allowed values: • **yes** • **no** (default)

The following parameters are only available for the **DITA PDF - based on HTML5 & CSS** single DITA topic transformation scenario (`pdf-css-html5-single-topic` trans type):

`args.root.map`	Specifies the path of the root map file used to expand the key references in the published topic.
`args.enable.root.map.key.processing`	Indicates whether or not the keys should be processed using the root map parameter. Allowed values: • **auto** (default) • **yes** • **no**

License Key

Chemistry License

If you have an **Oxygen PDF Chemistry** license key, you will be able to generate PDF output that is not stamped with the **Chemistry** logo image.

To install your **Chemistry** license key:

- If you are using the version of **Chemistry** that comes bundled in **Oxygen XML Editor/Author**, save the license key text in a file with the name `licensekey.txt` and place it in the *DITA-OT-DIR*`/plugins/com.oxygenxml.pdf.css/lib/oxygen-pdf-chemistry` folder.
- If you are using another **Chemistry** installation, make sure you place the `licensekey.txt` file in that folder.

Oxygen Publishing Engine License

If you have purchased a license for the **Oxygen Publishing Engine**, you will be able to produce both PDF and WebHelp output without any restrictions.

To install your **Oxygen Publishing Engine** license key, save the license key text in a file with the name `licensekey.txt` and place it in the *DITA-OT-DIR* folder.

Publishing Templates

An *Oxygen Publishing Template* defines all aspects of the layout and styles for output obtained from the following transformation scenarios:

- **WebHelp Responsive**
- **DITA Map PDF - based on HTML5 & CSS**

It is a self-contained customization package stored as a ZIP archive or folder that can easily be shared with others. It provides the primary method for customizing the output.

Some possible customization methods include:

- Add additional template resources to customize the output (such as logos, *Favicons*, or CSS files).
- Extend the default processing by specifying one or more XSLT extension points.
- Specify one or more transformation parameters to customize the output.
- Customize various aspects of the output through simple CSS styling.
- For **WebHelp Responsive** output, change the layout of the main page or topic pages by customizing which components will be displayed and where they will be positioned in the page.

The following graphics are possible sample structures for *Oxygen Publishing Template* packages:

Figure 311. Oxygen Publishing Template Package (WebHelp Responsive)

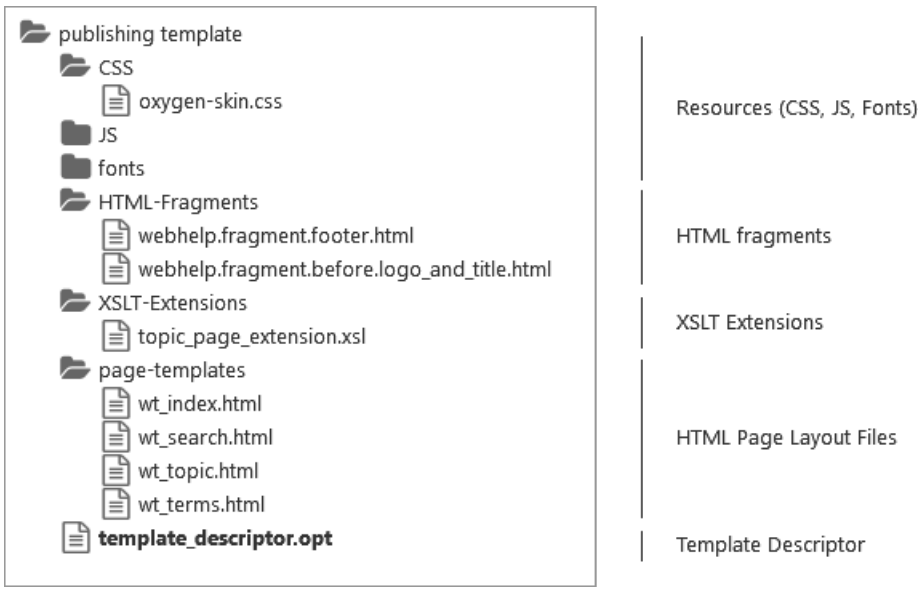


Figure 312. Oxygen Publishing Template Package (PDF)

For information about creating and customizing publishing templates, and how to adjust the WebHelp and PDF output through CSS styling and other customization methods, watch our Webinar: **Creating Custom Publishing Templates for WebHelp and PDF Output**. The Webinar slides and sample project are also available from that webpage.

Related Information:

- How to Create a Publishing Template *(on page 848)*
- How to Edit a Packed Publishing Template *(on page 850)*
- How to Add a Publishing Template to the Publishing Templates Gallery *(on page 850)*
- How to Share a Publishing Template *(on page 971)*

Publishing Template Package Contents for PDF Customizations

An *Oxygen Publishing Template* for PDF output must contain a template descriptor file and at least one CSS file, and may contain other resources (such as graphics, XSLT files, etc.). All the template resources can be stored in either a ZIP archive or in a folder. It is recommended to use a ZIP archive because it is easier to share with others.

Template Descriptor File

Each publishing template includes a descriptor file that defines the meta-data associated with template. It is an XML file with certain elements that defines all the resources included in a template (such as CSS files, images, and transformation parameters).

The template descriptor file must have the `.opt` file extension and must be located in the templates' root folder.

A PDF template descriptor might look like this:

```
<publishing-template>
    <name>Flowers</name>
    <pdf>
        <tags>
            <tag>purple</tag>
            <tag>light</tag>
        </tags>
```

```
        <preview-image file="flowers-preview.png"/>

        <resources>
            <css file="flowers.css"/>
        </resources>

        <parameters>
            <parameter name="figure.title.placement" value="top"/>
        </parameters>
    </pdf>
</publishing-template>
```

Tip: It is recommended to edit the template descriptor in **Oxygen XML Editor/Author** because it provides content completion and validation support.

Template Name and Description

Each template descriptor file requires a `<name>` element. This information is displayed as the name of the template in the transformation scenario dialog box.

Optionally, you can include a `<description>` and it displayed when the user hovers over the template in the transformation scenario dialog box.

```
<publishing-template>
    <name>Flowers</name>
    <description>Flowers themed light colored template</description>
    ...
```

Template Author

Optionally, you can include author information in the descriptor file and it displayed when the user hovers over the template in the transformation scenario dialog box. This information might be useful if users run into an issue or have questions about a certain template.

If you include the `<author>` element, a `<name>` is required and optionally you can include `<email>`, `<organization>`, and `<organizationUrl>`.

```
<publishing-template>
    ...
    <author>
      <name>John Doe</name>
      <email>jdoe@example.com</email>
      <organization>ACME</organization>
      <organizationUrl>http://www.example.com/jdoe</organizationUrl>
    </author>
    ...
```

PDF Element

The `<pdf>` element contains various details about the template and its resources that define the PDF output. It is a required element if you intend on using a DITA Map to PDF transformation scenario. The elements that are allowed in this `<pdf>` section specify the template tags *(on page 965)*, template preview image *(on page 965)*, resources *(on page 966)* (such as CSS files), transformation parameters *(on page 966)*, or XSLT extensions *(on page 967)*.

```
<pdf>
    <tags>
      ...
    </tags>
    <preview-image file="MyPreview.png"/>
    <resources>
      ...
    </resources>
    <parameters>
      ...
    </parameters>
</pdf>
```

Template Tags

The `<tags>` section provides meta information about the template (such as color theme). Each *tag* is displayed at the top of the **Templates** tab window in the transformation scenario dialog box and they help the user filter and find particular templates.

```
<publishing-template>
    ...
    <pdf>
      <tags>
        <tag>purple</tag>
        <tag>light</tag>
      </tags>
```

Template Preview Image

The `<preview-image>` element is used to specify an image that will be displayed in the transformation scenario dialog box. It provides a visual representation of the template to help the user select the right template. The image dimensions should be `200 x 115` pixels and the supported image formats are: `JPEG`, `PNG`, or `GIF`.

You can also include an `<online-preview-url>` element to specify the URL of a published sample of your template. This will display an **Online preview** icon in the bottom-right corner of the image in the transformation scenario dialog box and if the user clicks that icon, it will open the specified URL in their default browser.

```
<publishing-template>
```

```
    ...
    <pdf>
      ...
      <preview-image file="ashes/ashes-tree.png"/>
      <online-preview-url=https://www.example.com/samples/tiles/ashes</online-preview-url>
```

Template Resources

The `<resources>` section of the descriptor file specifies a set of resources (CSS files) that are used to customize various components in the generated output. These resources will be copied to the output folder during the transformation process. At least one CSS file must be included (using the `<css>` element).

```
<publishing-template>
    ...
    <pdf>
      ...
      <resources>
          <css file="css/custom_styles.css"/>
          <css file="css/custom_fonts.css"/>
      </resources>
```

Note: All relative paths specified in the descriptor file are relative to the template root folder.

Transformation Parameters

You can also set one or more transformation parameters in the descriptor file.

```
<publishing-template>
    ...
    <pdf>
      ...
      <parameters>
        <parameter name="show.changes.and.comments" value="yes"/>
      </parameters>
    </pdf>
```

The following information can be specified in the `<parameters>` element:

Parameter name

The name of the parameter. It may be one of the transformation parameters listed in the **Parameters** tab of the **DITA Map PDF - based on HTML5 & CSS** transformation scenario or a DITA-OT PDF-based output parameter.

Note: It is not recommended to specify an input/output parameter in the descriptor file (such as the input Map, DITAVAL file, or temporary directory).

Attention: JVM arguments like *-Xmx* cannot be specified as a transformation parameter.

Parameter Value

The value of the parameter. It should be a relative path to the template root folder for file paths parameters.

Parameter Type

The type of the parameter: `string` or `filepath`. The `string` value is default.

After creating a publishing template *(on page 968)* and adding it to the templates gallery *(on page 971)*, when you select the template in the transformation scenario dialog box, the **Parameters** tab will automatically be updated to include the parameters defined in the descriptor file. These parameters are displayed in italics.

XSLT Extension Points

The publishing templates support one or more XSLT extension points. They can be specified using the `<xslt>` element in the descriptor file using the following structure:

```
<publishing-template>
    ...
    <pdf>
        ...
        <xslt>
            <extension
              id="com.oxygenxml.pdf.css.xsl.merged2html5"
              file="xslt/merged2html5Extension.xsl"/>
            <extension
              id="com.oxygenxml.pdf.css.xsl.merged2merged"
              file="xslt/merged2mergedExtension.xsl"/>
        </xslt>
```

For more information about the available extension points, see: XSLT Extensions for PDF Transformations *(on page 972)*.

Combining PDF and WebHelp Responsive Customizations in a Template Package

An *Oxygen Publishing Template* package can contain both a PDF and WebHelp Responsive customization in the same template package and you can use that same template in both types of transformations. The template descriptor file can define the customization for both types by including both a `<webhelp>` and `<pdf>` element and some of the resources can be reused. Resources referenced in elements in the `<webhelp>` element will only be used for WebHelp transformations, and resources referenced in the elements in the `<pdf>` element will only be used in PDF transformations.

```
<publishing-template>
    <name>Flowers</name>
    <description>Flowers themed light-colored template</description>
    <webhelp>
```

```
        <tags>
            <tag>purple</tag>
            <tag>light</tag>
        </tags>
        <preview-image file="flowers-preview.png"/>
        <resources>
            <css file="flowers-wh.css"/>
            <css file="flowers-page-styling.css"/>
        </resources>
        <parameters>
            <parameter name="webhelp.show.main.page.tiles" value="no"/>
            <parameter name="webhelp.show.main.page.toc" value="yes"/>
        </parameters>
    </webhelp>
    <pdf>
        <tags>
            <tag>purple</tag>
            <tag>light</tag>
        </tags>
        <preview-image file="flowers-preview.png"/>
        <resources>
            <css file="flowers-pdf.css"/>
            <css file="flowers-page-styling.css"/>
        </resources>
        <parameters>
            <parameter name="show.changes.and.comments" value="yes"/>"/>
        </parameters>
    <pdf>
</publishing-template>
```

Related Information:

Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

How to Create a Publishing Template

To create a customization, you can start from scratch or from an existing template, and then adapt it according to your needs.

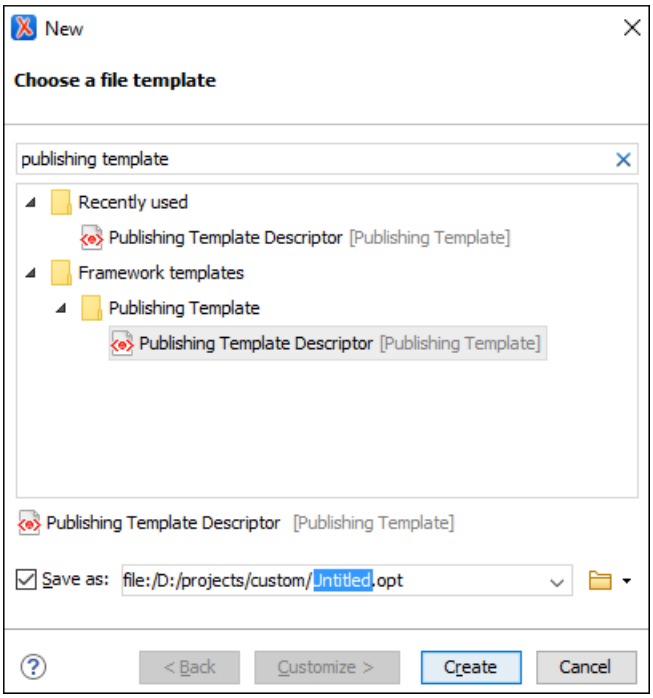
Creating a Template Starting from Scratch

To create a new *Oxygen Publishing Template*, follow these steps:

1. Create a folder that will contain all the template files.

2. In **Oxygen XML Editor/Author**, open the new document wizard (use **File > New** or the **New** toolbar button), then choose the **Publishing Template Descriptor** template.

Figure 313. Choosing the Publishing Template Descriptor Document Template



3. Save the `.opt` file into your customization directory.
4. Open the `.opt` file in the editor and customize it to suit your needs.

Creating a Template Starting from an Existing Template

If you are using a **WebHelp Responsive** or **DITA Map PDF - based on HTML5 & CSS** transformation, the easiest way to create a new *Oxygen Publishing Template (on page 1327)* is to select an existing template in the transformation scenario dialog box and use the **Save template as** button to save that template into a new template package that can be used as a starting point.

To create a new *Oxygen Publishing Template*, follow these steps:

1. Open the transformation scenario dialog box and select the publishing template you want to export and use as a starting point.
2. **Optional:** You can set one or more transformation parameters from the **Parameters** tab and the edited parameters will be exported along with the selected template. You will see which parameters will be exported in the dialog box that is displayed after the next step.
3. Click the **Save template as** button.

   Step Result: This opens a template package configuration dialog box that contains some options and displays the parameters that will be exported to your template package.
4. Specify a name for the new template.
5. **Optional:** Specify a template description.
6. **Optional:** The same publishing template package can contain both a WebHelp Responsive and PDF customization and you can use the same template in both types of transformations (**DITA**

Map WebHelp Responsive or **DITA Map to PDF - based on HTML5 & CSS**). You can use the **Include WebHelp customization** and **Include PDF customization** options to specify whether your custom template will include both types of customizations.

7. **Optional:** For **WebHelp Responsive** customizations, you can select the **Include HTML Page Layout Files** option if you want to copy the default *HTML Page Layout Files (on page 830)* in your template package. They are helpful if you want to change the structure of the generated HTML pages.
8. In the **Save as** field, specify the name and path of the ZIP file where the template will be saved.

   Step Result: A new ZIP archive will be created on disk in the specified location with the specified name.
9. Open the `.opt` file in the editor and customize it to suit your needs.

For more information about creating and customizing publishing templates, watch our video demonstration:

https://www.youtube.com/embed/zNmXfKWXwO8

Related Information:

- Publishing Template Package Contents for PDF Customizations *(on page 963)*
- Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

How to Edit a Packed Publishing Template

To edit an existing *Oxygen Publishing Template (on page 1327)* package, follow these steps:

1. Unzip the ZIP archive associated with the *Oxygen Publishing Template* in a separate folder.
2. Link the folder associated with the template in the **Navigator** view.
3. Using the **Navigator** view, you can modify the resources (CSS, JS, fonts) within the *Oxygen Publishing Template* folder to fit your needs.
4. Open the publishing template descriptor file (`.opt` extension) in the editor and modify it to suit your needs.
5. **Optional:** Once you finish your customization, you can archive the folder as a ZIP file.

Related Information:

- Publishing Template Package Contents for PDF Customizations *(on page 963)*
- Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

How to Use a Publishing Template in a PDF Transformation

From Oxygen XML Editor/Author

A publishing template can be used for PDF output from the **DITA Map PDF - based on HTML5 & CSS** transformation scenario (or from the **DITA PDF - based on HTML5 & CSS** transformation scenario).

The **Templates** tab in the transformation scenario dialog box displays all the templates that are available in your template gallery. To use a particular template in the transformation scenario, simply select it from this tab and then continue configuring the transformation using the other tabs to suit your needs.

To add the publishing template to your templates gallery, follow these steps:

1. Open the transformation scenario dialog box by editing a **DITA Map PDF - based on HTML5 & CSS** transformation (or a **DITA PDF - based on HTML5 & CSS** transformation scenario).
2. In the **Templates** tab, click the **Configure Publishing Templates Gallery** link to.

   Step Result: This will open the preferences page.
3. Click the **Add** button and specify the location of your template directory.

   Step Result: Your template directory is now added to the **Additional Publishing Templates Galleries** list.
4. Click **OK** to return to the transformation scenario dialog box.

Result: All the templates contained in your template directory will be displayed in the preview pane along with all the built-in templates.

From Command Line

You can use the simpler form, meaning you only use the `pdf.publishing.template` parameter to point to the **.opt* (publishing template) file:

```
dita.bat
 --input=map\test.ditamap"
 "-Dpdf.publishing.template=full_path_to_template_dir/my_template.opt"
 --format=pdf-css-html5
 ...
```

Or use the two parameters to indicate the folder containing the publishing templates and the name of the publishing template file relative to that folder:

```
dita.bat
 --input=map\test.ditamap"
 "-Dpdf.publishing.template=full_path_to_template_dir"
 "-Dpdf.publishing.template.descriptor=my_template.opt"
 --format=pdf-css-html5
 ...
```

Tip: You can also start the `dita` process by passing it a DITA OT Project File. Inside the project file you can specify as parameters for the `webhelp-responsive` transformation type the WebHelp-related parameters.

Related Information:

Transformation Parameters *(on page 956)*

How to Share a Publishing Template

To share a publishing template with others, following these steps:

1. Copy your template in a new folder in your project.
2. Go to **Options > Preferences > DITA > Publishing** and add that new folder to the list.
3. Switch the option as the bottom of that preferences page to **Project Options**.
4. Share your project file (`.xpr`).

XSLT Extensions for PDF Transformations

Since PDF output is primarily obtained by running XSLT transformations over the DITA input files, one customization method would be to override the default XSLT templates that are used by the PDF transformation.

The `pdf-css-html5` transformation type uses two stages to transform the merged DITA map (the one that aggregates all the topics) to HTML5:

1. **Stage 1**: Makes some changes on the merged map *(on page 982)* and the result is a modified merged map. This stage can be altered by implementing the **com.oxygenxml.pdf.css.xsl.merged2merged** XSLT extension point. This extension overrides the stylesheets found in the following folder: *DITA-OT-DIR*`\plugins\com.oxygenxml.pdf.css\xsl\merged2merged`.

   Note: Use this when you need to filter DITA content.
2. **Stage 2**: Transforms the merged map *(on page 982)* to HTML5 and the result is a single HTML document. This stage can be altered by implementing the **com.oxygenxml.pdf.css.xsl.merged2html5** XSLT extension point. This extension overrides the stylesheets found in the following folder: *DITA-OT-DIR*`\plugins\com.oxygenxml.pdf.css\xsl\merged2html5`.

   Note: Use this when you need to change the HTML structures generated for a specific DITA element.

These extension points can be used either from a *Publishing Template* or a DITA-OT extension plugin.

How to Use XSLT Extension Points for PDF Output from a Publishing Template

The examples in this section demonstrate how to use XSLT extension points from an *Oxygen Publishing Template*.

How to Style Codeblocks with a Zebra Effect

Suppose you want your *codeblocks* to have a particular background color for one line, and another color for the next line. One advantage of this coloring technique is that you can clearly see when text from the *codeblock* is wrapped.

This effect can be done by altering the HTML5 output, creating a `div` for each line from the code block, then styling them.

To add this functionality using an *Oxygen Publishing Template*, follow these steps:

1. If you have not already created a Publishing Template, you need to create one. For details, see How to Create a Publishing Template *(on page 968)*.
2. Link the folder associated with the publishing template to your current project in the **Navigator** view.
3. Using the **Navigator** view, create an `xslt` folder inside the project root folder.
4. In this folder, create an XSL file (for example, named `merged2html5Extension.xsl`) with the following content:

```
<?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"
    version="2.0">

    <xsl:template match="*[contains(@class, ' pr-d/codeblock ')]">
       <div class='zebra'>
          <xsl:analyze-string regex="\n" select=".">
            <xsl:matching-substring/>
            <xsl:non-matching-substring>
               <div><xsl:value-of select="."/></div>
            </xsl:non-matching-substring>
          </xsl:analyze-string>
       </div>
    </xsl:template>
</xsl:stylesheet>
```

5. Open the *template descriptor file (on page 963)* associated with your *publishing template* (the `.opt` file) and set the XSLT stylesheet created in the previous step with the `com.oxygenxml.pdf.css.xsl.merged2html5` XSLT extension point:

```
<publishing-template>
    ...
    <pdf>
        ...
        <xslt>
            <extension
              id="com.oxygenxml.pdf.css.xsl.merged2html5"
              file="xslt/merged2html5Extension.xsl"/>
        </xslt>
```

6. Create a `css` folder in the publishing template directory. In this directory, save a custom CSS file with rules that style the *codeblock* structure. For example:

```
div.zebra {
  font-family:courier, fixed, monospace;
  white-space:pre-wrap;
}
div.zebra > *:nth-of-type(odd){
  background-color: silver;
}
```

7. Open the *template descriptor file (on page 963)* associated with your *publishing template* (the `.opt` file) and reference your custom CSS file in the `resources` element:

```
<publishing-template>
    ...
    <pdf>
      ...
      <resources>
          <css file="css/custom.css"/>
      </resources>
```

8. Edit the **DITA Map PDF - based on HTML5 & CSS** transformation scenario.
9. In the **Templates** tab, click the **Choose Custom Publishing Template** link and select your template.
10. Click **OK** to save the changes and run the transformation scenario.

How to Remove the Related Links Section

Suppose that you want the *related links* sections to be removed from the PDF output.

To add this functionality using an *Oxygen Publishing Template*, follow these steps:

1. If you have not already created a Publishing Template, you need to create one. For details, see How to Create a Publishing Template *(on page 968)*.
2. Link the folder associated with the publishing template to your current project in the **Navigator** view.
3. Using the **Navigator** view, create an `xslt` folder inside the project root folder.
4. In this folder, create an XSL file (for example, named `merged2mergedExtension.xsl`) with the following content:

```
<?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"
    version="2.0">

   <xsl:template match="*[contains(@class, ' topic/related-links ')]">
      <!-- Remove. -->
   </xsl:template>
</xsl:stylesheet>
```

5. Open the *template descriptor file (on page 963)* associated with your *publishing template* (the *.opt* file) and set the XSLT stylesheet created in the previous step with the `com.oxygenxml.pdf.css.xsl.merged2merged` XSLT extension point:

```
<publishing-template>
    ...
    <pdf>
        ...
        <xslt>
            <extension
              id="com.oxygenxml.pdf.css.xsl.merged2merged"
              file="xslt/merged2mergedExtension.xsl"/>
        </xslt>
```

6. Edit the **DITA Map PDF - based on HTML5 & CSS** transformation scenario.
7. In the **Templates** tab, click the **Choose Custom Publishing Template** link and select your template.
8. Click **OK** to save the changes and run the transformation scenario.

How to Wrap Words in Markup

Suppose you want compound words that contain hyphens (or any other criteria) to be wrapped with inline elements (such as the HTML `<code>` element).

To add this functionality using an *Oxygen Publishing Template*, follow these steps:

1. If you have not already created a Publishing Template, you need to create one. For details, see How to Create a Publishing Template *(on page 968)*.
2. Link the folder associated with the publishing template to your current project in the **Navigator** view.
3. Using the **Navigator** view, create an `xslt` folder inside the project root folder.
4. In this folder, create an XSL file (for example, named `merged2htmlExtension.xsl`) with the following content:

```
<xsl:template match="text()">
  <xsl:variable name="txt">
    <xsl:next-match />
  </xsl:variable>
  <xsl:analyze-string regex="(\w|\-)+" select="$txt">
    <xsl:matching-substring>
      <!-- A word -->
      <xsl:choose>
        <xsl:when test="contains(.,'-')">
          <!-- A compound word -->
          <code class='compound-word'><xsl:value-of  select="."/></code>
        </xsl:when>
        <xsl:otherwise>
          <!-- A simple word -->
```

```
            <xsl:value-of select="." />
          </xsl:otherwise>
        </xsl:choose>
      </xsl:matching-substring>
      <xsl:non-matching-substring>
        <!-- Not a word -->
        <xsl:value-of select="." />
      </xsl:non-matching-substring>
    </xsl:analyze-string>
  </xsl:template>
```

5. Open the *template descriptor file (on page 963)* associated with your *publishing template* (the *.opt* file) and set the XSLT stylesheet created in the previous step with the `com.oxygenxml.pdf.css.xsl.merged2merged` XSLT extension point:

```
<publishing-template>
    ...
    <pdf>
        ...
        <xslt>
            <extension
              id="com.oxygenxml.pdf.css.xsl.merged2merged"
              file="xslt/merged2mergedExtension.xsl"/>
        </xslt>
```

6. Edit the **DITA Map PDF - based on HTML5 & CSS** transformation scenario.
7. In the **Templates** tab, click the **Choose Custom Publishing Template** link and select your template.
8. Click **OK** to save the changes and run the transformation scenario.

How to Convert Definition Lists into Tables

Suppose you want your definitions lists (`<dl>`) to be displayed as tables in your PDF output.

To add this functionality using an *Oxygen Publishing Template*, follow these steps:

1. If you have not already created a Publishing Template, you need to create one. For details, see How to Create a Publishing Template *(on page 968)*.
2. Link the folder associated with the publishing template to your current project in the **Navigator** view.
3. Using the **Navigator** view, create an `xslt` folder inside the project root folder.
4. In this folder, create an XSL file (for example, named `merged2html5Extension.xsl`) with the following content:

```
<xsl:template match="*[contains(@class, ' topic/dl ')]">
    <xsl:call-template name="setaname"/>
    <xsl:apply-templates select="*[contains(@class,
      ' ditaot-d/ditaval-startprop ')]" mode="out-of-line"/>
    <table>
```

```
        <xsl:call-template name="commonattributes"/>
        <xsl:call-template name="setid"/>
        <xsl:apply-templates/>
    </table>
    <xsl:apply-templates select="*[contains(@class,
      ' ditaot-d/ditaval-endprop ')]" mode="out-of-line"/>
</xsl:template>
<xsl:template match="*[contains(@class, ' topic/dlentry ')]">
    <tr>
        <xsl:apply-templates/>
    </tr>
</xsl:template>
<xsl:template
  match="
  *[contains(@class, ' topic/dd ')] |
  *[contains(@class, ' topic/dt ')]">
    <td>
        <xsl:call-template name="commonattributes"/>
        <xsl:call-template name="setidaname"/>
        <xsl:apply-templates select="../*[contains(@class,
          ' ditaot-d/ditaval-startprop ')]" mode="out-of-line"/>
        <xsl:apply-templates/>
        <xsl:apply-templates select="../*[contains(@class,
          ' ditaot-d/ditaval-endprop ')]" mode="out-of-line"/>
    </td>
</xsl:template>
```

5. Open the *template descriptor file (on page 963)* associated with your *publishing template* (the *.opt* file) and set the XSLT stylesheet created in the previous step with the `com.oxygenxml.pdf.css.xsl.merged2html5` XSLT extension point:

```
<publishing-template>
    ...
    <pdf>
        ...
        <xslt>
            <extension
              id="com.oxygenxml.pdf.css.xsl.merged2html5"
              file="xslt/merged2html5Extension.xsl"/>
        </xslt>
```

6. Edit the **DITA Map PDF - based on HTML5 & CSS** transformation scenario.
7. In the **Templates** tab, click the **Choose Custom Publishing Template** link and select your template.
8. Click **OK** to save the changes and run the transformation scenario.

How to Use XSLT Extension Points for PDF Output from a DITA-OT Plugin

The examples in this section demonstrate how to use XSLT extension points from a DITA-OT plugin.

How to Style Codeblocks with a Zebra Effect

Suppose you want your *codeblocks* to have a particular background color for one line, and another color for the next line. One advantage of this coloring technique is that you can clearly see when text from the *codeblock* is wrapped.

This effect can be done by altering the HTML5 output, creating a `<div>` for each line from the code block, then styling them.

To add this functionality using 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.pdf.custom.codeblocks`).
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.pdf.custom.codeblocks">
  <feature extension="com.oxygenxml.pdf.css.xsl.merged2html5"
             file="custom_codeblocks.xsl"/>
</plugin>
```

3. Create your customization stylesheet (for example, **custom_codeblocks.xsl**) with the following content:

```
<?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"
    version="2.0">

    <xsl:template match="*[contains(@class, ' pr-d/codeblock ')]">
       <div class='zebra'>
          <xsl:analyze-string regex="\n" select=".">
            <xsl:matching-substring/>
            <xsl:non-matching-substring>
               <div><xsl:value-of select="."/></div>
            </xsl:non-matching-substring>
          </xsl:analyze-string>
       </div>
    </xsl:template>
</xsl:stylesheet>
```

4. Use the **Run DITA-OT Integrator** transformation scenario *(on page 684)* found in the **DITA Map** section in the **Configure Transformation Scenario(s)** dialog box.
5. Create a custom CSS file with rules that style the *codeblock* structure. For example:

```
div.zebra {
  font-family:courier, fixed, monospace;
  white-space:pre-wrap;
}
div.zebra > *:nth-of-type(odd){
  background-color: silver;
}
```

6. Edit a **DITA Map PDF - based on HTML5 & CSS** transformation scenario and reference your custom CSS file (using the `args.css` parameter).
7. Run the transformation scenario.

How to Remove the Related Links Section

Suppose you want the *related links* sections to be removed from the PDF output.

To add this functionality using 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.pdf.custom.codeblocks`).
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.pdf.custom.related.links">
  <feature extension="com.oxygenxml.pdf.css.xsl.merged2merged"
             file="custom_related_links.xsl"/>
</plugin>
```

3. Create your customization stylesheet (for example, **custom_related_links.xsl**) with the following content:

```
<?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"
    version="2.0">

   <xsl:template match="*[contains(@class, ' topic/related-links ')]">
      <!-- Remove. -->
   </xsl:template>
</xsl:stylesheet>
```

4. Use the **Run DITA-OT Integrator** transformation scenario *(on page 684)* found in the **DITA Map** section in the **Configure Transformation Scenario(s)** dialog box.
5. Run the **DITA Map PDF - based on HTML5 & CSS** transformation scenario.

Sample Use-cases: Using XSLT and CSS for PDF Transformations

This section includes examples that can be used as a starting point for various customizations using XSLT and CSS.

How to Use a Bullet Instead of Numbers for Tasks Containing a Single Step

If a DITA Task only contains one step (`<step>` element), you probably want it to be rendered like an unordered list (displayed with a bullet instead of a number).

```
...
<steps>
   <step>
        <cmd>The step</cmd>
   </step>
</steps>
...
```

Should be rendered as:

```
 o The step
```

Instead of:

```
  1. The step
```

To achieve this, you need to create an XSLT template that adds a special `@outputclass` attribute (with its value set to `single`) for the single step, then you need to match this attribute from the CSS and change its aspect.

1. In the customization XSL, add:

   ```
   <xsl:template match="*[contains(@class, ' task/step ')][count(../*[contains
   (@class, ' task/step ')]) = 1]">
       <xsl:copy>
            <xsl:copy-of select="@*"/>
            <xsl:attribute name="outputclass" select="concat(@outputclass, ' single ')"/>
            <xsl:apply-templates/>
       </xsl:copy>
   </xsl:template>
   ```

2. In your customization CSS *(on page 981)*, add:

   ```
   *[outputclass ~= "single"] {
       list-style-type:circle !important;
       margin-left:2em;
   }
   ```

Related Information:

How to Use XSLT Extension Points for PDF Output from a Publishing Template *(on page 972)*

DITA-OT Extension Points

The **DITA-OT CSS-based PDF Publishing Plugin** supports DITA-OT extension points that can be used to expand the functionality of the transformation. The extension points are defined in the `plugin.xml` file. For more information, see DITA Open Toolkit Extension Points.

Related Information:

XSLT Extensions for PDF Transformations *(on page 972)*

How to Contribute a Custom CSS to the Transformation from a DITA-OT Plugin

This topic is intended for publishing architects/developers that need to deploy a customized DITA-OT.

Usually, the CSS styles can be passed to the transformation by referencing the CSS files using the `args.css` parameter. However, there are cases where you want to add some sort of "built-in" CSS that is applied in conjunction with the publishing template or CSS files referenced in the transformation.

For this, you need to use the **com.oxygenxml.pdf.css.init** extension point and set the value of the **extension.css** ANT property to the path of the custom CSS file:

1. In your **plugin.xml** file, add:

```
<feature extension="com.oxygenxml.pdf.css.init" file="init.xml"/>
```

2. Create a file named **init.xml** with the following ANT content:

```
<root>
  <property name="extension.css"
            value="${dita.plugin.[com.my.plugin.id].dir}/css/my-custom.css"/>
  <!-- add here more init stuff if needed -->
</root>
```

Note: The name of the root element does not matter. The content of this element will be copied in an initialization template.

Important: Make sure all file references begin with the ANT variable that is expanded to the base directory of your plugin.

Related Information:

How to Use XSLT Extension Points for PDF Output from a DITA-OT Plugin *(on page 978)*

Customizing PDF Output Using CSS

The publishing process is driven by a customization CSS.

Tip: If you use the default Chemistry processor in **Oxygen XML Editor/Author**, you can use LESS instead of CSS. In this case, the customization files should have the `.less` extension.

To change the styling of the output for the **DITA Map PDF - based on HTML5 & CSS** and the **DITA PDF - based on HTML5 & CSS** transformation scenarios:

1. Create the CSS file that will contain all your customizations. It is recommended to create this file in your project directory so you can edit it easily.
2. Add the custom CSS rules. See the various topics in this section for assistance with specific types of customizations.
3. For linking the CSS file, you have two options:
    a. Create a publishing template, create the customization CSS file inside the template folder and link it into the publishing template descriptor. See Publishing Templates *(on page 816)*.
    b. Choose an existing publishing template, then edit the scenario and set the full path to the custom CSS file as the value of the `args.css` parameter. The rules from custom CSS will override the rules from the template CSS files.
4. Run the transformation scenario.

Debugging the CSS

If you notice that some of the CSS properties were not applied as expected, some of the tips offered in this topic might help you with the debugging process.

Merged Map File

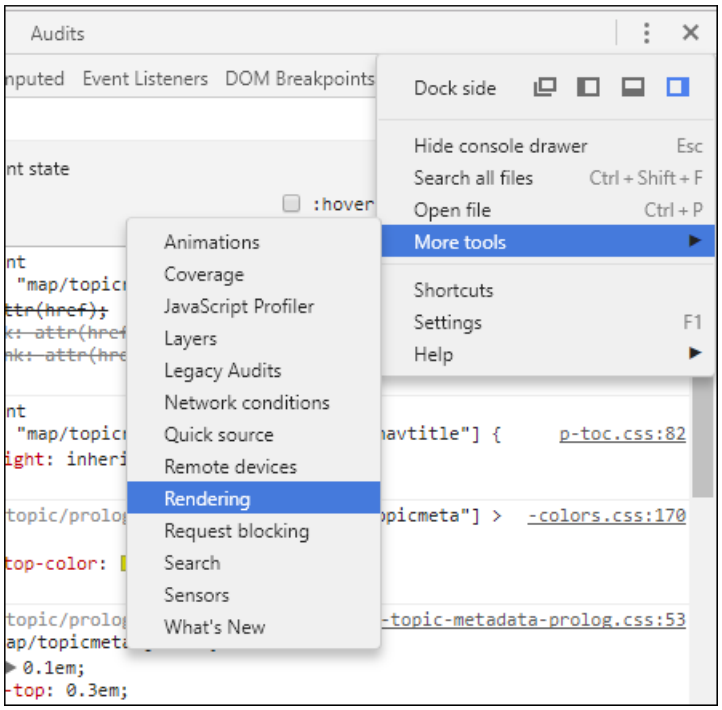
Depending on the type of transformation, one or more merged map files are created at some point during the transformation stages. These files could be used to help debug unexpected results.

1. The first thing you should try is to check the file structure of the **merged map** file. This can be found in the `out/pdf-css` directory and it has the `.merged.html` file extension (you will also find a `.merged.xml` file that aggregates the entire ditamap structure). You can open the HTML files in **Oxygen XML Editor/Author** to examine the structure. Optionally, you can use the pretty print feature (**Format and Indent**) to make the structure easier to read.
2. Check that the CSS selectors are written correctly against the document structure.
3. If you still cannot identify the problem, then inspect how the styles are applied (you can try any of the methods listed below).

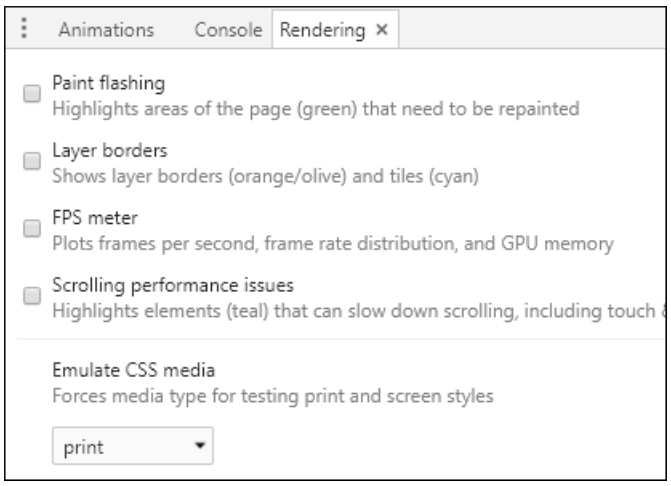
Inspecting the Applied Styles Using the Chrome Browser

To inspect the applied CSS styles using Chrome:

1. Open the file ending in `.merged.html`.
2. Click on the element you want to inspect.
3. Activate the **Chrome Developer Tools** by using ⋮ **> More Tools > Developer Tools**, or press **CTRL+SHIFT+I**.
4. Activate the **Rendering** pane by using ⋮ **> More Tools > Rendering**:



5. In the **Rendering** pane, select **print** from the **Emulate CSS media** section. This will activate the CSS selectors enclosed in `@media print {..}`.



Note: This allows you to debug the styling of elements, table of contents, and index, but not the styles of the page margin boxes (headers, footers) or page breaks.

Tip: In the left pane of the **Developer Tools** interface, you can inspect elements and their styles in the **Elements** tab. You can click on any of the links to display the applied CSS files in the **Styles** tab in the right pane. Editing the styles in that pane results in a live preview of how the change will affect the output.

CAUTION: Do not modify the built-in rules directly in the CSS files from the **Oxygen XML Editor/Author** installation. Instead, copy the rules to your own customization CSS.

Inspecting the Applied Styles Using Oxygen XML Editor/Author

To inspect styles:

1. In **Oxygen XML Editor/Author**, open the file ending in `.merged.html`.
2. **[Optional]** From the **Styles** toolbar, choose the **+ Print Ready** entry. This will activate certain CSS selectors enclosed in `@media print {..}`.
3. Click on the element you want to style. Use the **Inspect Styles** action from the Contextual Menu. A specialized **CSS Inspector** view will show the built-in CSS rules.

Tip: With this file open in **Author** mode, it might be helpful to switch the **Tags Display Mode** to **Full Tags with Attributes**. You might be able to identify the selector you need to style without using the **CSS Inspector** view.

Note: This allows you to debug styling of elements, but not of the page margin boxes (headers, footers) or page breaks.

CAUTION: Do not modify the built-in rules directly in the CSS files from the **Oxygen XML Editor/Author** installation. Instead, copy the rules to your own customization CSS.

Other techniques

These are some other techniques you may find useful:

- Add background and borders properties to the specific CSS rule. If they do not appear in the output then there is a problem with the rule selector.
- Try to use the `!important` notation to the property that is not applied, or make the selector more specific (you can add more parent selectors).
- To figure out how the elements are mapped to PDF, you can use this fragment in the customization CSS:

```
* {
   border: 1pt solid blue !important;
}
*:before(1000) {
   content:  oxy_name() !important;
   color: orange;
}
*:before(999) {
   content: "[ class= '" attr(class) "'] " !important;
   color: orange;
}
```

This will show the element name, its class attribute, and will paint a blue border around each of the elements in the output. It will not show the page margin boxes or some content elements that were hidden.

How to Speed up CSS Development and Debugging

You can speed up your CSS development considerably by not invoking the entire pipeline of transforming your DITA maps to PDF. Instead, you can use the merged map *(on page 982)* and transform it directly to PDF.

1. Transform your DITA Map to PDF using the **DITA Map PDF - based on HTML5 & CSS** transformation scenario.
2. Open the merged file *(on page 982)* (*.merged.html) that is located in the output directory in the editor.
3. Configure an **XML to PDF transformation with CSS** scenario. Do not set CSS files here since the merged file already contains pointers to the stylesheets. This scenario uses the Chemistry CSS processor.
4. **Optional:** Enable the output of the CSS processor using the following preferences page: **Options > Preferences > XML > PDF Processors > CSS Processor**.

Now you can make incremental changes to the CSS stylesheet and quickly see the results by transforming the merged file directly.

Fastpath: If your changes involve only element styling, with no specific paged media CSS rules and properties, you can simply open the merged file in a browser (such as Chrome or Firefox) and refresh at each CSS change, as shown in: Debugging the CSS *(on page 982)*.

How to Write XPath Expressions

This topic contains some guidelines for writing XPath expressions. They are used to extract the content from the merged DITA map document.

This is an example where the product name meta-information is placed before the front page title:

```
*[class~="front-page/front-page-title"]:before {
    text-align: left;
    content: oxy_xpath("(//*[contains(@class, 'topic/prodname')]/text())[1]");
    display:block;
}
```

Notes:

- Do not use the DITA element names directly. You must use the DITA `@class` attribute instead, as these attributes are propagated to the merged elements (including HTML `<div>` elements) while the element names can be lost. By using the class selectors, you also cover DITA specializations.
- Use the "[1]" XPath predicate to select the first value from the document. For example, `oxy_xpath("(//*[contains(@class, 'topic/prodname')]/text())[1]")`. The meta-information might be copied multiple times in the output, inherited by the `<topicref>` elements, so you can get many more values than expected.
- Do not use strings as values for the pseudo-elements content, as the string values are not supported for pseudo-elements. Instead, use the XPath directly.

- Use the **Oxygen XPath Builder** view to test the XPath expressions.

Related Information:

How to Debug XPath Expressions *(on page 986)*

How to Debug XPath Expressions

You can use the content extracted from the document using the `oxy_xpath` function in your pseudo-elements (`:before`, `:after`) or in `string-set` variables.

For example, the following XPath finds the publication author, set in the DITA map:

```
<map>
  <title>The Art of Bike Repair</title>
  <topicmeta>
    <author>John Doe</author>
  </topicmeta>
...
```

```
:root {
    string-set: author oxy_xpath('//*[contains(@class, "front-page/front-page")]/*[contains(@class,
 "map/topicmeta")]/*[contains(@class, "topic/author")]/text()');
}
```

To debug an XPath expression:

1. Read the XPath Expressions Guidelines *(on page 985)*.
2. Begin by transforming your document using your customization CSS.
3. In the output folder, you will find a `[MAP_NAME].merged.html` file (or if you are using the **DITA Map PDF - based on HTML5 & CSS** transformation, a `[MAP_NAME].merged.html` file).
4. Open the merged file in the **Oxygen XML Editor/Author**.
5. Activate the **XPath Builder** view (**Window > Show View > XPath/XQuery Builder**).
6. Paste your XPath expression and click the **Execute XPath** button. Check if it returns the expected results.

The XPath builder has a function that allows it to display the document path of the current element from the editor (**Settings drop-down menu > Update on cursor move**). Alternatively, you can right-click the element in the merged document and select the **Copy XPath** action, then paste it in the XPath builder.

Related Information:

XPath Builder Documentation

XPath Examples (w3schools.com)

Default Page Definitions

All page definitions are found in: `[PLUGIN_DIR]css/print/p-pages-and-headers.css`.

Note: This is listed solely for illustration purposes, as the plugin might use something different.

There are pages for the default page, chapter page, front matter page, back matter page, blank pages, index page, and large tables page.

Default Page

The default page imposes a header that contains the publication title, chapter, and section title. They alternate on the left or right side of the page:

```
   @page :left {
       @top-left {
           content: string(maptitle) string(parttitle) string(chaptertitle) string(sectiontitle) "
| " counter(page);
           font-size:8pt;
       }
   }
   @page :right{
       @top-right {
           content: string(maptitle) string(parttitle) string(chaptertitle) string(sectiontitle) "
| " counter(page);
           font-size:8pt;
       }
   }
```

Tip: To override the default rules defined for named pages (such as chapter or table of contents), you need to use more specific page rules that contain the page name:

```
  @page :left, table-of-contents:left, chapter:left {
       @top-left {
           content: "...";
       }
   }
   @page :right, table-of-contents:right, chapter:right{
       @top-right {
           content: "...";
       }
   }
```

Chapter Page

This is inherited from the default page. The chapter page is associated to the topics marked as chapters, usually direct children of the map. It clears the header from the first page of each chapter.

```
@page chapter{
 /* Currently inherit from the default page.*/
}

/* No headers on the chapter first page. */
@page chapter:first:left{
 @top-left {
  content: none;
 }
}
@page chapter:first:right{
 @top-right {
  content: none;
 }
}
```

Front Matter and Back Matter page

The bookmap front matter and back matter page. It clears the headers.

```
@page matter-page {
     @top-left-corner    {          content:none }
     @top-left           {          content:none }
     @top-center         {          content:none }
     @top-right          {          content:none }
     @top-right-corner   {          content:none }
     @bottom-left-corner {          content:none }
     @bottom-left        {          content:none }
     @bottom-center      {          content:none }
     @bottom-right       {          content:none }
     @bottom-right-corner{          content:none }
}
```

Blank Pages

The following example clears the header for the blank pages that may be created by a `page-break-before`, `page-break-after`, or by using double side pagination *(on page 1047)*:

```
@page :blank{
 @top-left {
  content: none;
 }
 @top-right {
  content: none;
 }
```

```
    }
```

Index Page

The page that contains the index terms (appears only if there are such items in your topics). It uses a lower roman page number in the footer:

```
@page index {
       @top-left-corner    {          content:none }
       @top-left           {          content:none }
       @top-center         {          content:none }
       @top-right          {          content:none }
       @top-right-corner   {          content:none }
       @bottom-left-corner {          content:none }
       @bottom-left        {          content:none }
       @bottom-center      {          content: counter(page, lower-alpha) }
       @bottom-right       {          content:none }
       @bottom-right-corner{          content:none }
}
@media oxygen-chemistry {
       @page index {
           column-count: 2;
           column-fill: auto;
       }
 }
```

When transformed, the page layout is spread on two columns.

Large Tables Page

The big tables are placed on a rotated page, with orientation landscape:

```
@page landscape-page :right {
    size: landscape;
    @top-left {
        content: none
    }
    @top-center {
        content: none
    }
    @top-right {
        content: none
    }
    @right-bottom {
        content: string(maptitle) string(parttitle) string(chaptertitle) string(sectiontitle) " | "
counter(page);
       font-size: 8pt;
```

```
        transform: rotate(90);
        vertical-align: middle;
        text-align: right;
    }
}
@page landscape-page :left {
    size: landscape;
    @top-left {
        content: none
    }
    @top-center {
        content: none
    }
    @top-right {
        content: none
    }
    @right-top {
        content: string(maptitle) string(parttitle) string(chaptertitle) string(sectiontitle) " | "
 counter(page);
        font-size: 8pt;
        transform: rotate(90);
        vertical-align: middle;
        text-align: left;
    }
}
```

Page Size

This is where you can find information on how the page sizes are defined.

Page Size - Built-in CSS rules

The `[PLUGIN_DIR]/css/print/p-page-size.css` file contains the default page rules. It uses the US-LETTER size (8.5 X 11 inches). The content of this file is:

```
@page {
    padding-top:0.2em;
    padding-bottom:0.2em;
    size: us-letter;
    margin: 1in;
}
```

Note: This is listed solely for illustration purposes, as the plugin might use something different.

How to Change the Page Size

Suppose you want to publish using the standard A4 page size, with a margin of 2cm.

In your customization CSS *(on page 981)*, use:

```
@page {
    size: A4;
    margin: 2cm;
}
```

If you need different margins depending on the page side:

```
@page {
    size: A4;
    margin: 2cm;
}
@page :left{
    margin-right:4cm;
}
@page :right{
    margin-left:4cm;
}
```

This would only increase the gutter margins or the inside margins needed for binding of the final book. The other margins would remain 2cm.

How to Change the Page Orientation

Suppose you want to publish on a landscape page orientation. The default is portrait, so you need to change it by using the size property. This will contain both the physical measurements and the orientation. In your customization CSS *(on page 981)*, use:

```
@page {
    size: us-letter landscape;
}
```

How to Change the Page Settings for a Specific Element

Suppose your publication mainly uses a portrait page orientation, but there are some topics that have wide images. To avoid having the images bleed outside of the page, you could use a wider page setting (landscape).

1. Mark the topic with an `@outputclass` attribute and give it a distinct value (for example, **wide**), you can set the attribute on the root element of the topic or on the `<topicref>` element from the map.

   Note: The `@outputclass` values from the `<topicref>` automatically propagate to the root of the topic from the merged map *(on page 982)*.

2. In your customization CSS *(on page 981)*, match the output class and associate it with a named page. In the following example, the page has a landscape orientation and small margins. This technique works for any element (e.g. a table or a list) not just for a topic.

```
@page wide-page {
    size: letter landscape;
    margin: 0.5in;
}
*[outputclass = 'wide'] {
    page: wide-page !important;
}
```

Note: The `!important` rule is necessary to override the default page settings.

Page Headers and Footers

The page headers and footers use the string sets defined for publication, chapter, and section titles. These string-sets are defined in the numbering CSS *(on page 1027)*:

parttitle

Set to the title of the current part (only for DITA bookmaps that use parts).

chaptertitle

Set to the title of the current chapter (Shallow and Deep numbering).

sectiontitle

Set to the title of each section (Deep numbering only).

Although you may define string sets in your customization CSS, you need to take into account the fact that the string-set CSS property is not additive, and matching the same elements will end up breaking the current definitions. A very common use-case is to match the title element that is also used in the default CSS. The best approach, in this case, is to take a look at the rules from the numbering CSS *(on page 1027)*, copy the ones dealing with string sets to your customization, then alter the property definition by adding your definition to the existing ones (and not removing the existing ones).

Related Information:

Numbering *(on page 1027)*

Page Headers and Footers - Built-in CSS

The headers and footers are part of the page definitions. To see how the default page layouts are defined, see: Default Page Definitions *(on page 987)*.

How to Position Text in the Headers and Footers

By default, the name of the publication and chapter titles are placed in the `top-left` or `top-right` page margin boxes:

```
  @page :left {
       @top-left {
           content: string(maptitle) string(parttitle) string(chaptertitle) string(sectiontitle) "
| " counter(page);
       }
   }
   @page :right{
       @top-right {
           content: string(maptitle) string(parttitle) string(chaptertitle) string(sectiontitle) "
| " counter(page);
       }
   }
```

If you want to change this, you should use the `content` CSS properties of other page margin boxes, and inhibit the ones in the above content. For example, to set the chapter title in the page top left corner, you can use:

```
  @page :left {
       @top-left {
           content:none;
       }
       @top-left-corner {
           content: string(maptitle) string(parttitle) string(chaptertitle) string(sectiontitle) "
| " counter(page);
           white-space: nowrap;
           text-align:left;
       }
   }
   @page :right{
       @top-right {
           content: none;
       }
       @top-right-corner {
           content: string(maptitle) string(parttitle) string(chaptertitle) string(sectiontitle) "
| " counter(page);
           white-space: nowrap;
           text-align:right;
       }
   }
```

Note: The corner page margin boxes are fixed and limited as the available space. Above, the `text-align` and `white-space` properties are used to make the text bleed out of these boxes towards the center of the page. If you plan to add an image or artwork background, you should consider using the technique described in: How to Decorate the Header by Using a Background Image on the Entire Page *(on page 1002)*.

How to Change the Size of Headers and Footers

This is directly related to the page margins and size.

The headers and footers are placed in the so-called page margin boxes, a series of rectangular areas residing in the page margins.

To affect the margins of all page definitions, you may use the following rule:

```
@page {
 margin-top:3cm !important;
 margin-bottom:3cm !important;
 margin-left:2cm !important;
 margin-right:2cm !important;
}
```

If you want to affect only a specific page, like the first page from chapters for instance, you must use more specific page selectors. See the Default Page Definitions *(on page 987)* for details.

Note that the page margin boxes fill the entire page margin. This means the `margin-top`, for example, dictates the height of the `@top-left-corner`, `@top-left`, `@top-center`, `@top-right`, `@top-right-corner` margin boxes. These cannot have margins on themselves, so to change the position of the content inside them, you must use `padding` properties:

```
@page {
 @top-left {
  content:"..."
  padding: 1cm;
 }
 ..
}
```

How to Change the Font of the Headers and Footers

To change the font for all the headers and footers, in your customization CSS *(on page 981)*, add a CSS rule similar to this:

```
@page {
    font-size: 12pt;
    font-family: "Arial";
}
```

Important: These settings apply to all page margin boxes, but not to the text inside the page.

If you want to change the settings only for a specific page type (for example, the table of contents), use the name of the page:

```
@page table-of-contents {
    font-size: 12pt;
    font-family: "Arial";
}
```

Related Information:

How to Change the Header of the Table of Contents *(on page 1038)*

How to Change Header Text for Each Topic

It is possible to dynamically change the header depending on the content in a topic. The following example assumes that the data to be presented in the header is located in the metadata section of each topic. One way is to specify it in the DITA map is by using the `<topicmeta>` element for the `<topicref>` topic reference:

```
...
 <topicref href="topics/installing.dita">
    <topicmeta>
      <data name="header-data" value="ID778-3211"/>
    </topicmeta>
    ...
```

In the above example, there is set of key value pairs with the name `header-data`. This information is automatically copied into the content in the merged map file *(on page 982)*, like this:

```
<topic ... >
    <title class="- topic/title ">Installing</title>
    <shortdesc class="- topic/shortdesc ">You install components to make them available for your
      solution.</shortdesc>
    <prolog class="- topic/prolog ">
      ...
      <data class="- topic/data " name="header-data" value="ID778-3211"/>
      ...
```

This information can be extracted from the CSS:

```
/* Define the string set variable that contains the text extracted from the data element */
*[class ~= "topic/topic"] *[class ~= "topic/data"][name="header-data"] {
  string-set: hdrstr attr(value);
}
/* Using the value='none' stops applying the image. */
*[class ~= "topic/topic"] *[class ~= "topic/data"][name="header-data"][value="none"] {
  string-set: hdrstr "";
}
```

```
/* Use the string set variable in one of the page margin boxes. */
@page chapter {
  @top-left-corner {
    content: string(hdrstr);
  }
}
```

Notes: The string set is applied to all pages that follow the data element, until another data element changes it:

```
...
 <topicref href="topics/installing.dita">
    <topicmeta>
      <data name="header-data" value="ID778-3211"/>
    </topicmeta>
 </topicref>
 <topicref href="..."> <!-- Uses the same value -->
 <topicref href="..."> <!-- Uses the same value -->
 <topicref href="..."> <!-- Uses the same value -->
 <topicref href="topics/change.dita">
    <topicmeta>
      <data name="header-data" value="ID990-3200"/>
    </topicmeta>
 </topicref>
 <topicref href="..."> <!-- The string set is changed now -->
 <topicref href="..."> <!-- The string set is changed now -->
 <topicref href="..."> <!-- The string set is changed now -->
```

To clear the text, use the `none` value:

```
...
     <topicref href="..."> <!-- The string set is void now -->
...
```

How to Change Header Images for Each Chapter

It is possible to dynamically change an image in the header depending on the chapter. For this, you need to define an image reference in the metadata section of each chapter. One way is to specify it in the DITA map by using the `<topicmeta>` element for the `<chapter>` topic reference:

```
...
 <chapter href="topics/installing.dita">
    <topicmeta>
      <data name="header-image" value="img/installing.png"/>
    </topicmeta>
```

```
    ...
```

In the above example, there is set of key value pairs with the name `header-image`. The `img/installing.png` is an image reference relative to the DITA map URI. This information is automatically copied into the content in the merged map file *(on page 982)*, like this:

```
<topic is-chapter="true" ... >
    <title class="- topic/title ">Installing</title>
    <shortdesc class="- topic/shortdesc ">You install components to make them available for your
      solution.</shortdesc>
    <prolog class="- topic/prolog ">
      ...
      <data class="- topic/data " name="header-image" value="img/installing.png"/>
      ...
```

This information can be picked up from CSS:

```
/* Define the string set variable that contains an URL */
*[class ~= "topic/topic"] *[class ~= "topic/data"][name="header-image"] {
  string-set: imgst oxy_url(oxy_xpath('/*/@xtrf'), attr(value));
}
/* Using the value='none' stops applying the image. */
*[class ~= "topic/topic"] *[class ~= "topic/data"][name="header-image"][value="none"] {
  string-set: imgst "";
}
/* Use the string set variable in one of the page margin boxes. */
@page chapter {
  @top-left-corner {
    content: string(imgst);
    font-size:0; /* remove the font ascent and descent */
  }
}
```

Details: The `@value` attribute is used to build a URL relative to the URI of the DITA map. To determine the base URI of the DITA map, the `@xtrf` attribute was used from the root element of the merged map document, extracted using the `oxy_xpath` function.

Notes:

- The image is always aligned vertically to the middle of available space from the page margin box.
- Make sure you use an image of the correct size. For example, if you want to place the image in the top-left corner of the page, assuming the top and left page margins are 1 in, then make sure the image is a square having a size of 1 in.
- The image is applied to all pages that follow the data element, until another data element changes it:

```
...
 <chapter href="topics/installing.dita">
```

```
    <topicmeta>
      <data name="header-image" value="img/installing.png"/>
    </topicmeta>
  </chapter>
  <chapter href="..."> <!-- Uses the same installing.png image -->
  <chapter href="..."> <!-- Uses the same installing.png image -->
  <chapter href="..."> <!-- Uses the same installing.png image -->
  <chapter href="topics/change.dita">
    <topicmeta>
      <data name="header-image" value="img/change.png"/>
    </topicmeta>
  </chapter>
  <chapter href="..."> <!-- Uses the same change.png image -->
  <chapter href="..."> <!-- Uses the same change.png image -->
  <chapter href="..."> <!-- Uses the same change.png image -->
```

To clear the image, use the `none` value:

```
...
      <data name="header-image" value="none"/>
...
```

How to Add a Multi-line Copyright Notice to the Footer

Suppose you want to add a footer with the following two lines of text at the end of each page that is shown on the right side:

```
© 2017 - My Company Ltd
All rights reserved
```

For this, you need to specify a rule that matches all the right pages and adds that content in the `bottom-center`. In your customization CSS *(on page 981)*, add the following CSS rule:

```
@page :right{
    @bottom-center {
        content: "© 2017 - My Company Ltd \A All rights reserved";
        font-size: 0.5em;
        color: silver;
    }
}
```

Note: Other page rules (such as the *table-of-contents*) override the contents of the `@bottom-center` because they are more specific. If you need to also print the copyright in the TOC pages, then use this as the selector:

```
@page :right, table-of-contents:right {
```

```
  ...
}
```

Note: To use new lines (`\n` characters) in your headers or footers, use the `\A` notation, as in the example above.

How to Add a Group of Topics to the Footer

To create a footer that contains the content of several topic files, but only on the last page, there are two possible approaches:

Method 1: Using the position:fixed CSS Property

1. Group all the footer topics under a single parent topic, under the last topic from your DITA map. For example, you can have the following map structure:

   ```
   ...
   End topic
      Footer container topic
         Footer content topic 1
         Footer content topic 2
   ```

2. Add an `@outputclass=footer` on the `<topic>` root element of the footer container topic, or on its `<topicref>` in the map.
3. Use the CSS `position: fixed` property to position this topic to the bottom of the page:

   ```
   *[outputclass ~= "footer"] {
     position: fixed;

     bottom: 0.5in;
     left: 0.5in;

     width:5in;
     height:200pt;
   }
   ```

Note: Make sure the width and height are enough for the content of the footer to fit. Be careful because the content might bleed out of the page. Use bottom and left values to position the block in the page.

Method 2: Using the float:footnote CSS Property

The second approach would be to declare the footer block as a footnote. Assuming the same DITA Map structure as above, you can use the following CSS fragment:

```
*[outputclass ~= "footer"] {
  float:footnote;
}
```

```
*[outputclass ~= "footer"]:footnote-call{
  color:transparent;
  font-size:0;
}
*[outputclass ~= "footer"]:footnote-marker{
  color:transparent;
  font-size:0;
}
```

Note: Use transparent colors and/or zero size font to avoid the display of the footnote counters.

How to Add a Background Image to the Header

A common use-case is to add a background image to one of the page corners.

```
@page :left {
    @bottom-left-corner{
        content: " ";
        background-image:
 url('https://www.oxygenxml.com/resellers/resources/OxygenXMLEditor_icon.svg');
        background-repeat:no-repeat;
        background-position:50% 50%;
    }
}
```

Important: Always specify a `content` property. If not, the page margin box will not be generated.

Another use-case is to use the `@top-left` or `@top-right` page margin boxes. These boxes have an automatic layout and they can be very small if they have no content. If there is no text to be placed over the image, use a series of non-breaking spaces (`\A0`) to increase the box width as in the following example (alternatively, you can use the technique described in How to Decorate the Header by Using a Background Image on the Entire Page *(on page 1002)*):

```
@page :left {
    @top-left{
        content: '\A0\A0\A0\A0\A0\A0\A0\A0\A0\A0';
        background-image:
 url('https://www.oxygenxml.com/resellers/resources/OxygenXMLEditor_icon.svg');
        background-repeat:no-repeat;
        background-position:50% 50%;
    }
}
```

Note: You can use raster image formats (such as PNG or JPEG), but it is best to use vector images (such as SVG or PDF). They scale very well and produce better results when printed. In addition, the text from these images is searchable and can be selected (if the glyphs have not been converted to shapes) in the PDF viewer.

Related Information:

Images and Figures *(on page 1088)*

How to Add a Background Image for the Cover *(on page 1018)*

How to Add a Link in Headers and Footers *(on page 1001)*

How to Add a Link in Headers and Footers

Method 1: Using an SVG Link Attribute

It is possible to add a link inside the document header (or footer) by using the `<a>` element inside an SVG document. For example, suppose you have the following SVG document named *custom.svg*:

```
<svg viewBox="0 0 100 40" xmlns="http://www.w3.org/2000/svg">
  <a xlink:href="https://www.oxygenxml.com/chemistry-html-to-pdf-converter.html">
    <text x="10" y="25">PDF Chemistry</text>
  </a>
</svg>
```

This creates an SVG link with *PDF Chemistry* displayed as its text (the content of the `<text>` element).

Note: If you just want to add a link without text, you can define a rectangle that contains the link instead of text.

To display the link, you just need to set your SVG file as the content of one of the page margin boxes:

```
@page {
  @top-left {
    content: url("custom.svg");
  }
}
```

Method 2: Using the CSS *-oxy-link* Property

It is also possible to add a link inside the document header (or footer) by using the `-oxy-link` property on the `@page` margin box declaration. The entire page margin box will behave as a link and will be clickable.

```
@page {
  @top-left {
    content: "Link";
    -oxy-link: "https://www.oxygenxml.com/";
    color:blue;
  }
}
```

How to Decorate the Header by Using a Background Image on the Entire Page

If you want to precisely position artwork and the page margin boxes are not sufficient, it is possible to use a background image for the entire page.

This technique consists of creating an image (SVG is the best since it is a vector image) as wide as the page that would contain the logo and placing other decorations at the desired locations. This offers the best results and the position of the artwork does not depend on the page margin contents.

Example:

```
@page :left, chapter:left, chapter:first:left {
 background-image: url('img/page_background_image_with_logos_and_artwork_for_left_page.svg');
 background-repeat: no-repeat;
 background-position: 50% 50%;
}
```

For a list of all the possible page names, see: Default Page Definitions *(on page 987)*.

Related Information:

How to Add a Background Image for the Cover *(on page 1018)*

How to Style a Part of the Text from the Header

If you need to style a fragment of text (for example, a company slogan) with certain colors or font styles, you have several options:

- Use an SVG image as the background for a page margin box or for the entire page. See: How to Add a Background Image to the Header *(on page 1000)*.
- Use the `oxy_label` constructor. This is a function that creates a text label with a set of styles.

  ```
  @page {
      @top-right {
          content: oxy_label(text, "My Company", styles, "color:red; font-size: larger;")
                   ' '
                   oxy_label(text, "Product", styles, "color:blue;
   text-decoration:underline;"));
      }
  }
  ```

  You can combine the `oxy_label` with `oxy_xpath`, to extract and style a piece of text from the document:

  ```
  content: oxy_label(text, oxy_xpath("/some/xpath"), styles, "color:blue; "));
  ```

  Note: These functions work only with the Chemistry CSS processor.

  Note: You cannot use `string()` inside an `oxy_label()`. As a workaround, to apply styling on the dynamic text retrieved by a `string()` function you can define some overall styles for the entire page margin box and then use the `oxy_label` to style differently the static text.

```
@page {
    @top-right {
  color: red;
        content: oxy_label(text, "My Company", styles, "color:black")
                 ' '
                 string(chaptertitle); /* This inherits the styling from @top-right*/
    }
}
```

- Use two adjacent page margin boxes, and style them differently:

```
@page {
  @top-center {
    content: "First part";
    color: red;
    text-align:right;
  }
  @top-left {
    content: "- Second part";
    color: blue;
    text-align:left;
  }
}
```

How to Simplify the Header (Keep Only the Chapter Title)

The headers display information such as *map title*, *part title*, *chapter title*, and *section title*, ending in the page number.

```
content: string(maptitle) string(parttitle) string(chaptertitle) string(sectiontitle) " | "
 counter(page);
```

This might be too much if you have long titles. The solution is to override the default header content.

In your customization CSS *(on page 981)*, add the following CSS rule:

```
@page :left {
    @top-left {
        content: string(chaptertitle) " | " counter(page);
    }
}
@page :right{
    @top-right {
        content: string(chaptertitle) " | " counter(page);
    }
}
```

Important: Some of the CSS default page rules are more important. If you see that the content does not change:

- Try to also specify the name of the page, to increase the specificity of the rules:

```
@page :left, table-of-contents:left, chapter:left{
  ...
}
@page :right, table-of-contents:right, chapter:right{
  ...
}
```

- Add an `!important` classifier just before the semi-colon.

```
@top-right {
    content: string(chaptertitle) " | " counter(page) !important;
}
```

How to Change the Header Separators (Between Sections and Chapter Titles)

There are some `strings` defined for part, chapter, and sections. Each of these strings start with the " | " character as a separator. For example, in the header of a page, you may find a sequence of strings:

```
My Publication | Introduction | Getting Started
```

1. "My Publication" is the value of the `maptitle` string.
2. "Introduction" is the value of the `chaptertitle` string.
3. "Getting Started" is the value of the `sectiontitle` string.

There might be cases where you want to change this separator. You will need to recompose the header content using the above string sets. Suppose you want to use " - " as a separator. In your customization CSS *(on page 981)*, add the following CSS rule:

```
*[class ~= "topic/topic"][is-part] > *[class ~= "topic/title"] {
    string-set: parttitle " - " counter(part, upper-roman) " - " content(), chaptertitle  ""; /*
 Avoid propagating a past chapter title on a new part */
}
*[class ~= "topic/topic"][is-chapter]:not([is-part]) > *[class ~= "topic/title"] {
    string-set: chaptertitle " - " counter(chapter) " - " content();
}
```

If you enabled the deep numbering for chapters and subsections *(on page 1031)*, then use:

```
/*
 * Alter the string sets that are shown in the header of the page.
 */
```

```
*[class ~= "map/map"][p|numbering^='deep'] *[class ~= "topic/topic"][is-part] > *[class
 ~= "topic/title"] {
    string-set: parttitle " - " counter(part, upper-roman) " - " content(), chaptertitle  "" ; /*
 Avoid propagating a past chapter title on a new part */
}
*[class ~= "map/map"][p|numbering^='deep'] *[class ~= "topic/topic"][is-chapter]:not([is-part]) >
 *[class ~= "topic/title"] {
    string-set: chaptertitle " - " counter(chapter) " - " content(), sectiontitle "" ;
}
*[class ~= "map/map"][p|numbering^='deep'] *[class ~= "topic/topic"][is-chapter]:not([is-part]) >
 *[class ~= "topic/topic"] > *[class ~= "topic/title"] {
    string-set: sectiontitle " - " counter(chapter) "." counter(section1) " - " content();
}
```

How to Change the Header Styling Depending on Page Side

To modify the styling of the default page headers, add the following CSS rule in your customization CSS *(on page 981)*:

```
@page :left {
    @top-left {
        color:navy;
        font-style:italic;
    }
    @top-right {
        color:red;
    }
}
```

If you intend to modify **just the headers of the table of contents**, use the `table-of-contents` page rule selector:

```
@page table-of-contents:left {
    @top-left {
        color:navy;
        font-style:italic;
    }
    @top-right {
        color:red;
    }
}
```

How to Use XPath Computed Data or Images in the Header or Footer

A very simple approach is to use the `oxy_xpath` directly in the `content` property:

```
@page front-page {
     @top-center {
          content: "Created: " oxy_xpath('//*[contains(@class, " topic/created "][1]');
     }
}
```

Example 1: Compute the Number of Words

The following example computes the number of words from the publication. It counts all the words, including the ones from the TOC, but does not take the static labels into account:

```
@page front-page {
     @bottom-center {
          content: "Number of words: "
                    oxy_xpath("string-length(normalize-space(/)) - \
                              string-length(translate(normalize-space(/),' ','')) +1");
     }
}
```

Note: The XPath expression from the page rules is evaluated in the context of the document root element, so you will need to use absolute expressions starting with `/` or `//`. This is different from the case when the `oxy_xpath` is used in CSS rules that match an element. In this case, the XPath expressions are evaluated in the context of the matched element and you can use relative paths.

Tip: XPath 2.0 is supported (not schema aware).

Example 2: Retrieve Image from a Document and Insert it in the Header

Another example is to use an image from the document in the publication header:

```
<bookmeta>
  <metadata>
   ...
    <data name="cover">
        <image href="product-cover.png" outputclass="cover-image"/>
    </data>
   ...
  </metadata>
</bookmeta>
```

```
@page {
   @top-center {
        content: url("oxy_xpath('//*[contains(@outputclass, "cover-image")]/@href')");
   }
}
```

If the URL returned by `oxy_xpath` is not absolute, it is considered to be relative to the CSS file. To obtain an absolute URL from one relative to the XML document, you can use in the XPath expression functions like `resolve-uri` and `document-uri`:

```
@page {
   @top-center {
        content: url(oxy_xpath("resolve-uri(//*[contains(@outputclass, 'cover-image')]/@href),
 document-uri(/))"));
   }
}
```

Example 3: Insert the Current Date in the Footer

Another example is to use the `oxy_xpath` function to compute the current date and insert it in the publication footer:

```
@page {
    @bottom-left {
        content: oxy_xpath('current-date()');
    }
}
```

Example 4: Picking up Metadata from the Original Map

Another example is to use the `oxy_xpath` function to extract the title, or any other element text value from the original processed DITA map file. For this, you can use the `@xtrf` attribute that is set on the root element of the merged map. This attribute contains the URL of the input map.

```
:root{
        string-set: maptitle oxy_xpath('document(@xtrf)/*[contains(@class, " map/map
 ")]/*[contains(@class, " topic/title ")]/text()');
}
```

Related Information:

- Oxygen PDF Chemistry User Guide: Headers and Footers
- http://zvon.org/xxl/XPathTutorial/General/examples.html
- Oxygen User Guide: oxy_xpath() Function

How to Add a Line Under the Header

There are two ways to add a horizontal line under the header.

Method 1: Add a Border in the Page Margin Boxes

To add a horizontal line that would stretch across the width of the page, add a bottom border to each of the 5 margin boxes in the top side of the page (`top-left-corner`, `top-left`, `top-center`, `top-right`, `top-right-corner`).

If you consider that the space between the header and the bottom border is too large, you could also change the alignment by adding a `vertical-align: bottom;` declaration in the page margin boxes.

For example, if you need to set some text as a header in the top-left margin box and insert a horizontal line under it, the customization CSS would look something like this:

```
@page chapter, chapter:first:left:right, front-page{
   padding-top: 1em;

  @top-left {
    content: "Custom header";
    color: gray;
    border-bottom: 1px solid black;
    vertical-align: bottom;
  }
  @top-center{
    content:" ";
    border-bottom: 1px solid black;
    vertical-align: bottom;
  }
  @top-right{
    content:" ";
    border-bottom: 1px solid black;
    vertical-align: bottom;
  }
  @top-right-corner{
    content:" ";
    border-bottom: 1px solid black;
    vertical-align: bottom;
  }
  @top-left-corner{
    content:" ";
    border-bottom: 1px solid black;
    vertical-align: bottom;
  }
```

Note: The `padding-top: 1em;` is used to avoid the border at the bottom of the header that joins with the page content.

Method 2: Use a Background Image

An alternative method is to add a horizontal line/border under an existing header (or in any other part of the page) using an SVG image, as described in How to Add a Background Image to the Header *(on page 1000)*.

Metadata

DITA has a solid vocabulary for specifying metadata. There are `<prolog>` elements in the topics, and `<topicmeta>`, `<bookmeta>` elements in the bookmaps. They can be used to define authors, dates, audiences, organizations, etc. See: https://www.oxygenxml.com/dita/1.3/specs/archSpec/base/metadata-in-maps-and-topics.html

It is up to you to decide where this information should be presented, in the PDF content or in the PDF document properties.

Metadata - XML Fragment

In the merged map file *(on page 982)*, the metadata section is placed inside the `<oxy:front-page>` element. This is different from the original placement in the map or bookmap - after the title, but allows usage of information from it in the title page.

Bookmaps

This is an example of a section taken from a merged bookmap. It only contains some of the possible metadata elements. The `bookmeta` metadata section is inherited from `topicmeta`:

```
<bookmap xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/"
  xmlns:opentopic-index="http://www.idiominc.com/opentopic/index" cascade="merge"
    class="- map/map bookmap/bookmap "
  ditaarch:DITAArchVersion="1.3" >

  <oxy:front-page xmlns:oxy="http://www.oxygenxml.com/extensions/author">

   <bookmeta xmlns:dita-ot="http://dita-ot.sourceforge.net/ns/201007/dita-ot"
     class="- map/topicmeta bookmap/bookmeta ">
     <author class="- topic/author ">Howe Tuduit</author>
     <bookid class="- topic/data bookmap/bookid ">
       <isbn class="- topic/data bookmap/isbn ">071271271X</isbn>
       <booknumber class="- topic/data bookmap/booknumber ">SG99-9999-00</booknumber>
        <maintainer class="- topic/data bookmap/maintainer ">
        <organization class="- topic/data bookmap/organization ">ACME Tools</organization>
        <person class="- topic/data bookmap/person "/>
       </maintainer>
     </bookid>
     <bookrights class="- topic/data bookmap/bookrights ">
        ...
         <bookowner class="- topic/data bookmap/bookowner ">
  <organization class="- topic/data bookmap/organization ">ACME Tools, Inc.</organization>
         </bookowner>
     </bookrights>
        </bookmeta>
```

```
        <oxy:front-page-title>
            ...
...
```

For the **DITA Map PDF - based on HTML5 & CSS** transformation type, the merged map is further processed resulting in a collection of HTML5 `<div>` elements. These elements preserve the original DITA `@class` attribute values and add a new value derived from the DITA element name.

```
<div
    class="- map/map bookmap/bookmap bookmap" ... >

   <div class=" front-page/front-page front-page">

   <div
     class="- map/topicmeta bookmap/bookmeta boometa">
     <div class="- topic/author author">Howe Tuduit</div>
     <div class="- topic/data bookmap/bookid bookid">
       <div class="- topic/data bookmap/isbn isbn">071271271X</div>
       <div class="- topic/data bookmap/booknumber booknumber">SG99-9999-00</div>
       <div class="- topic/data bookmap/maintainer maintainer">
          <div class="- topic/data bookmap/organization organization">ACME Tools</div>
          <div class="- topic/data bookmap/person person"/>
        </div>
      </div>
      <div class="- topic/data bookmap/bookrights bookrights">
        ...
         <div class="- topic/data bookmap/bookowner bookowner">
              <div class="- topic/data bookmap/organization organization">
                                        ACME Tools, Inc.
              </div>
         </div>
       </div>
     </div>

     <div class=" front-page/front-page-title front-page-title">
            ...
...
```

Maps

The maps have a more simple structure, they use the `<topicmeta>` element for metadata sections. This is also a simplified example, as there may be many more elements in the metadata section:

```
<map xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/"
```

```
    xmlns:opentopic-index="http://www.idiominc.com/opentopic/index"
          cascade="merge" class="- map/map "
    ditaarch:DITAArchVersion="1.3">
    ...

<oxy:front-page xmlns:oxy="http://www.oxygenxml.com/extensions/author">

  <topicmeta class="- map/topicmeta ">
      <author class="- topic/author ">Dan C</author>
      <metadata class="- topic/metadata ">
         <prodinfo class="- topic/prodinfo ">
             <prodname class="- topic/prodname ">oXygen PDF CSS DITA Plugin</prodname>
         </prodinfo>
      </metadata>
      <audience class="- topic/audience "/>
  </topicmeta>
...
```

For the **DITA Map PDF - based on HTML5 & CSS** transformation type, the merged map is further processed resulting in a collection of HTML5 `<div>` elements:

```
<div xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/"
    xmlns:opentopic-index="http://www.idiominc.com/opentopic/index"
          cascade="merge" class="- map/map "
    ditaarch:DITAArchVersion="1.3">
    ...

<div class=" front-page/front-page front-page">

  <div class="- map/topicmeta topicmeta">
      <div class="- topic/author author">Dan C</div>
      <div class="- topic/metadata metadata">
         <div class="- topic/prodinfo prodinfo">
             <div class="- topic/prodname prodname">oXygen PDF CSS DITA Plugin</div>
         </div>
      </div>
      <div class="- topic/audience audience"/>
  </topicmeta>
...
```

Metadata - Built-in CSS rules

The `[PLUGIN_DIR]/css/print/p-meta.css` file contains the rules that extract metadata.

How to Add the Publication Audience to the Custom PDF Metadata

The audience element indicates the users the publication is addressing. This can be placed inside a `<topicmeta>` element in a `<map>` as in the following example:

```
<map>
    ...
    <topicmeta>
        ...
        <audience type="programmer" job="programming" experiencelevel="expert"/>
```

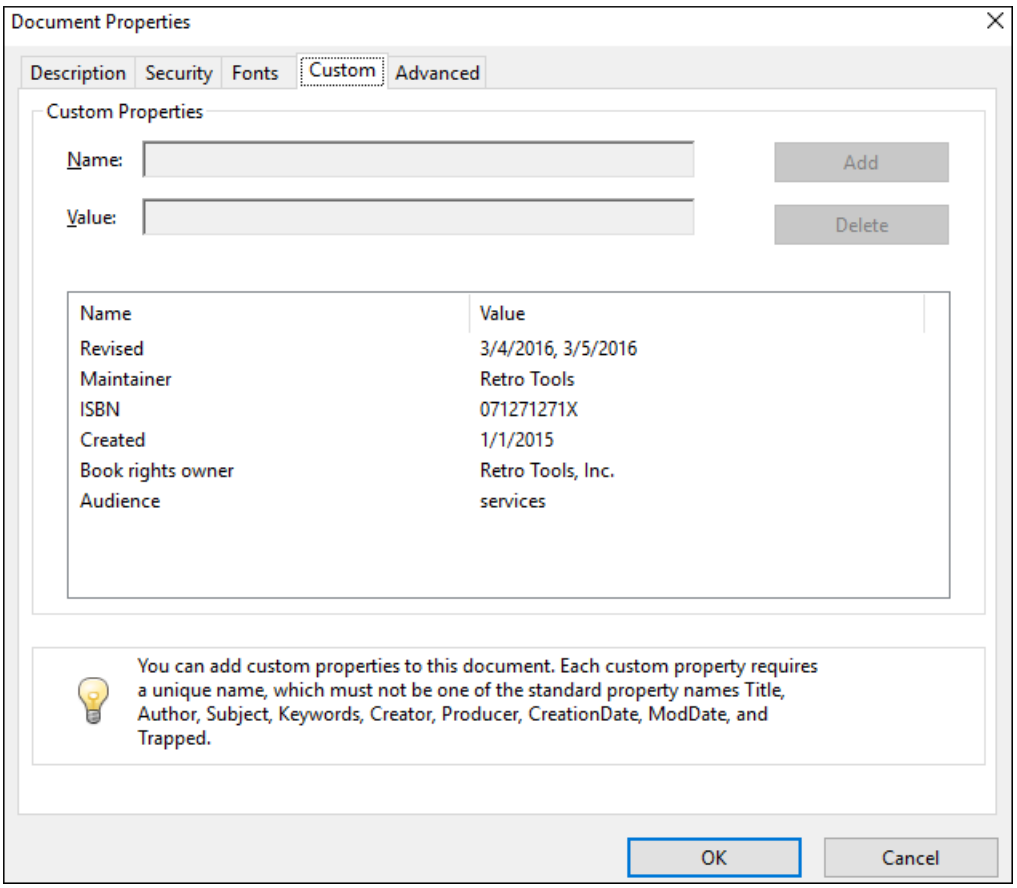
To collect the `@type` attribute, add the following in your customization CSS *(on page 981)*:

```
*[class ~= "map/map"] > *[class ~= "map/topicmeta"] > *[class ~= "topic/audience"] {
    -oxy-pdf-meta-custom: "Audience" attr(type);
}
```

Note: It is best to use the class selector (such as `*[class ~= "map/topicmeta"]`) instead of `topicmeta` to cover cases where the elements are specialized (for instance, in a bookmap the `bookmeta` is a `topicmeta`, so your selector will also function for bookmaps, not only simple maps.

Note: The selector begins with `map >` to choose the `<topicmeta>` that is a direct child of the map, not other `<topicmeta>` elements from other `<topicref>` elements.

The metadata is displayed in the **Custom** tab of the **Document Properties** dialog box from Acrobat Reader:



How to Show Metadata in the Cover Page

Suppose you need to present the **author** and the **ISBN** just under the publication title. Your bookmap contains:

```
<bookmap id="taskbook">
    <booktitle>
        <booklibrary>Retro Tools</booklibrary>
        <mainbooktitle>Product tasks</mainbooktitle>
        <booktitlealt>Tasks and what they can do</booktitlealt>
    </booktitle>
    <bookmeta>
        <author>Howe Tuduit</author>
        <critdates>
            <created date="1/1/2015"/>
            <revised modified="3/4/2016"/>
            <revised modified="3/5/2016"/>
        </critdates>
        <bookid>
            <isbn>071271271X</isbn>
            <booknumber>SG99-9999-00</booknumber>
...
```

The entire `<booktitle>` element content is displayed on the first page of the PDF, so suppose you need to add the information after it.

In your customization CSS *(on page 981)*, add the following CSS rules:

```
*[class ~= "bookmap/booktitle"]:after {
    display:block;
    content: "by " oxy_xpath('//*[contains(@class, " bookmap/bookmeta ")]/*[contains(@class, "
 topic/author ")]/text()');
    margin-top: 4em;
    text-align: center;
    color: gray;
}
*[class ~= "bookmap/booktitle"]:after(2) {
    display:block;
    content: "ISBN "oxy_xpath('//*[contains(@class, " bookmap/isbn ")]/text()');
    text-align: center;
    color: gray;
}
```

The following CSS extensions were used in this example:

- **oxy_xpath** - Executes an XPath expression and returns a string content. Use this whenever you need to extract data from elements other than the one matched by the CSS rule selector.

- `:after(N)` - Creates more than one *after* pseudo-element. The argument value represents how far the generated content is from the real content. In the example above, the content of the `:after` is closer to the title (upper) than the content of the `:after(2)`.

Note: The `attr()` CSS function can also be used but is limited to extracting attribute values from the matched element.

Generating Synthetic Pages for Metadata.

Suppose you need to show this information on a page that follows the title page, instead of on the title page. In this case, you need to prepare a named page and place the content on it. In your customization CSS *(on page 981)*, add the following CSS rules:

```
@page page-for-meta {
    background-color: yellow; /* Just to see it better*/
    @top-left-corner {
        content:""; /* Remove the default header */
    }
    @top-right-corner {
        content:""; /* Remove the default header */
    }
}
*[class ~= "bookmap/booktitle"]:after {
    page: page-for-meta;
}
*[class ~= "bookmap/booktitle"]:after(2) {
    page: page-for-meta;
}
```

How to Show Metadata in the Header or Footer

The header and footer are composed of page margin boxes that can be populated with static text by using *string-sets*.

If you need to add some of the map metadata to the header of the front page (for example, the **creation date**), add the following CSS rules in your customization CSS *(on page 981)*:

```
*[class ~= "front-page/front-page"] >
  *[class ~= "map/topicmeta"] >
    *[class ~= "topic/critdates"] >
      *[class ~= "topic/created"]{
    string-set: mapcreated attr(date);
}

@page front-page {
     @top-center {
```

```
            content: "Created: " string(mapcreated);
        }
}
```

Note: The *front-page* is the name of a page that used to present the element with the class "front-page/front-page". The above page rule is combined with the default styles.

How to Remove or Change the PDF Keywords

The keywords defined in the prolog sections of topics are automatically collected and set as PDF keywords. These are shown by the readers in the PDF document properties window.

If you need to remove them, you can use the following CSS snippet in your customization CSS *(on page 981)*:

```
:root {
 -oxy-pdf-meta-keywords:"";
}
```

To change them, if you have a hard-coded list, you just enumerate each of them in the property content, separating them with comma:

```
:root {
 -oxy-pdf-meta-keywords:"alpha, beta, gamma";
}
```

If you need to extract them by other criteria from the merged map, you can use the `oxy_xpath()` function instead of the hard-coded list.

How to Change the PDF Publication Title Property

The `<title>` element of a bookmap is quite complex and contains elements for the book library and an alternate title:

```
<booktitle>
   <booklibrary>Retro Tools</booklibrary>
   <mainbooktitle>Main Book Title</mainbooktitle>
   <booktitlealt>Book Title Alternative</booktitlealt>
 </booktitle>
```

For the publication title, the built-in CSS uses only the content of the `<mainbooktitle>`. If you want to collect all of the text from the `<booktitle>`, you can add the following rule to your customization CSS *(on page 981)*:

```
:root {
    -oxy-pdf-meta-title: oxy_xpath('(//*[contains(@class, "bookmap/booktitlealt")])[1]/text()');
    -oxy-pdf-meta-description: "";
}
```

An XPath expression is used to collect all the `<booktitlealt>` elements from the merged map, select the first one, then use its text.

The built-in CSS uses the `<booktitlealt>` as the PDF description. In the example above, this property is cleared since it was moved as a title.

How to Use a Key Value from the Map in the CSS

To use a key value in the CSS, the key must be referenced from the content (either a topic or map).

If you do not have it referenced, you may force a reference by using the `<topicmeta>` or `<bookmeta>` section of your map and a `<data>` element. This has no effect on the published content, but allows the CSS rules to use its content.

```
<bookmeta>
  ....
  <data keyref="my_key"/>
  ....
</bookmeta>
```

This is expanded in the merged HTML file to:

```
<div class="- map/topicmeta bookmap/bookmeta topicmeta bookmeta">
 ...
 <div keyref="my_key" class="- topic/data data">
  <div class="- topic/keyword keyword">KEY VALUE</div>
 </div>
 ...
</div>
```

Suppose that you need the expanded key value in the footer of the publication. You can define a string-set on this `data` element:

```
*[class ~= "topic/data"][keyref="my_key"] {
 string-set: key-string content(text);
}
@page {
 @bottom-left {
  content: "My key is: " string(key-string) !important;
 }
}
```

Or you can use the value from a `:before` pseudo-element, like the one for the title:

```
*[class ~= "topic/title"]:before {
 content: oxy_xpath("//*[contains(@class, 'topic/data')][@keyref = 'my_key']//text()");
}
```

Another use-case is to use the key as a source for a custom PDF document property:

```
*[class ~= "topic/data"][keyref="my_key"] {
 -oxy-pdf-meta-custom: attr(keyref) content(text);
}
```

Cover (Title) Page

Customizing the cover page is one of the most requested customization requests.

Cover Page - XML Fragment

The merged map file *(on page 982)* contains the `<oxy:front-page>` element, as a child of the root element. This contains the metadata and an `<oxy:front-page-title>` element with the title structure.

```
<bookmap xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/" ...>
  <oxy:front-page xmlns:oxy="http://www.oxygenxml.com/extensions/author">
   <bookmeta xmlns:dita-ot="http://dita-ot.sourceforge.net/ns/201007/dita-ot"
            ...
   </bookmeta>
   <oxy:front-page-title>
    <booktitle xmlns:dita-ot="http://dita-ot.sourceforge.net/ns/201007/dita-ot"
       class="- topic/title bookmap/booktitle ">
       <booklibrary class="- topic/ph bookmap/booklibrary ">Retro Tools</booklibrary>
       <mainbooktitle class="- topic/ph bookmap/mainbooktitle ">Tasks</mainbooktitle>
       <booktitlealt class="- topic/ph bookmap/booktitlealt ">Product tasks</booktitlealt>
    </booktitle>
   </oxy:front-page-title>
</oxy:front-page>
```

For the **DITA Map PDF - based on HTML5 & CSS** transformation type, the merged map is further processed resulting in a collection of HTML5 `<div>` elements. These elements preserve the original DITA `@class` attribute values and add a new value derived from the DITA element name.

```
<div class="- map/map bookmap/bookmap bookmap" ... >
   <div class=" front-page/front-page front-page">
      <div class="- map/topicmeta bookmap/bookmeta boometa">
        ...
      </div>
     <div class=" front-page/front-page-title front-page-title">
       <div class="- topic/title bookmap/booktitle booktitle">
         <div class="- topic/ph bookmap/booklibrary booklibrary">Retro Tools</div>
         <div class="- topic/ph bookmap/mainbooktitle mainbooktitle">Tasks</div>
         <div class="- topic/ph bookmap/booktitlealt booktitlealt">Product tasks</div>
      </div>
...
```

Cover Page - Built-in CSS rules

The element with the class `frontpage/frontpage` is associated with a page named *front-page* with no headers or footers. The front page title is styled with a bigger font. The built-in CSS rules are in `[PLUGIN_DIR]/css/print/p-front-page.css`.

```
@media print {
    *[class~="front-page/front-page"] {
        page: front-page;
    }
    *[class~="front-page/front-page-title"] {
        display:block;
        text-align:center;
        margin-top:3in;
        font-size:2em;
        font-family:arial, helvetica, sans-serif;
        font-weight:bold;
    }
    @page front-page {
        @top-left-corner    {          content:none }
        @top-left           {          content:none }
        @top-center         {          content:none }
        @top-right          {          content:none }
        @top-right-corner   {          content:none }
        @bottom-left-corner {          content:none }
        @bottom-left        {          content:none }
        @bottom-center      {          content:none }
        @bottom-right       {          content:none }
        @bottom-right-corner{          content:none }
    }
}
```

Note: This is listed solely for illustration purposes, as the plugin might use something different.

How to Add a Background Image for the Cover

The simplest way is to create an SVG image as large as the entire physical page and set it as the background for the *front-page*. This makes it very easy to accomplish a good positioning of the graphical elements or artwork. In the foreground, you can place text fragments using a series of `:after` pseudo-elements bound to the front page title.

To set the size to an SVG image, you should specify the `@width` and `@height` attributes on the `<svg>` root element using specified unit values (in, cm, etc.) This should be enough only if all the coordinates from your drawing have unit identifiers.

If you are using unit-less coordinates in your drawing like the following:

```
<polygon points="17.78 826.21 577.51 ....
```

then make sure you also specify a `@viewBox` attribute on the `<svg>` root element that defines the abstract rectangle that contains the drawing:

```
<svg xmlns="http://www.w3.org/2000/svg" width="8.5in" height="11in" viewBox="0 0 600 850">
```

The following SVG document has the `@width`, `@height`, and `@viewBox` attributes. The width and height have physical units (in inches), while the view box and rectangle coordinates are unit-less.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
       "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="8.5in" height="11in" viewBox="0 0 110 110">
  <desc>A gradient as big as a page.</desc>
  <defs>
    <linearGradient id="lc"
      x1="0%" y1="0%"
      x2="0%" y2="100%"
      spreadMethod="pad">
      <stop offset="0%"   stop-color="#00DD00" stop-opacity="1"/>
      <stop offset="100%" stop-color="#00AA00" stop-opacity="1"/>
    </linearGradient>
  </defs>
  <rect x="5" y="5" width="100" height="100" rx="10" ry="10"
    style="fill:url(#lc);
    stroke: #005000;
    stroke-width: 3;"/>
  <text x="33%" y="50%"  color="#FFFFAA"> Sample </text>
</svg>
```

This example shows a gradient. It is the size of a US-LETTER page and can be used in a publication using this page size.

Note: You can use raster image formats (such as PNG or JPEG), but it is best to use vector images (such as SVG or PDF). They scale very well and produce better results when printed. In addition, the text from these images is searchable and can be selected (if the glyphs have not been converted to shapes) in the PDF viewer.

In your customization CSS *(on page 981)*, add the following:

```
@page front-page {
    background-image: url("us-letter.svg");
    background-position:center;
}
```

For smaller artworks, you can use `background-position` with percentage values to position and center the artwork (for example, a company logo):

```
@page front-page {
    background-image:url("company-logo.svg");
    background-position:50% 5%; /* The first is the alignement on the X axis, the second on the Y axis.*/
    background-repeat:no-repeat;
}
```

Note: The text from the SVG or PDF background images are searchable in the PDF reader.

Related Information:

Oxygen PDF Chemistry: Graphics

How to Change Styling of the Cover Page Title

Match the front page title element in your customization CSS *(on page 981)* based on its class attribute:

```
*[class ~= "front-page/front-page-title" {
  margin-top: 1in;
  font-size: 3em;
}
```

How to Add Text to the Cover Page

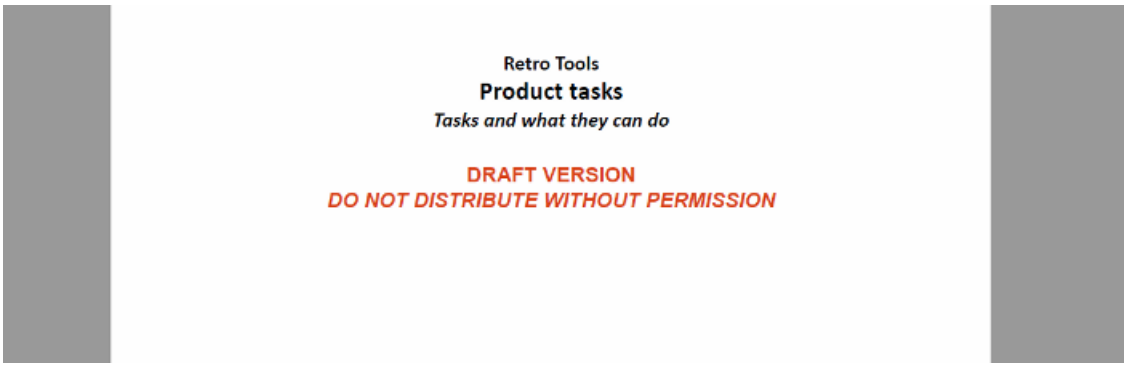
If you need to add arbitrary text to the cover page, you can use the front page title element as an anchor and add as many blocks of text as you need after it, and style them differently.

In your customization CSS *(on page 981)*, add the following:

```
*[class ~= "front-page/front-page-title"]:after(1) {
    display:block;
    content: "DRAFT VERSION";
    font-size: large;
    color: red;
    text-align:center;
}
*[class ~= "front-page/front-page-title"]:after(2) {
    display:block;
    content: "DO NOT DISTRIBUTE WITHOUT PERMISSION";
    font-size: large;
    color: red;
    text-align:center;
    font-style: italic;
```

```
}
```

The result is:



To use content from the document, you can use the `oxy_xpath` function in the `content` property. For a more complex example, including the generation of a new page for the synthetic `:after` elements, see: How to Show Metadata in the Cover Page *(on page 1012)*.

Related Information:

How to Show Metadata in the Cover Page *(on page 1012)*

How to Place Cover on the Right or Left Side

In your customization CSS *(on page 981)*, add the following CSS rules:

```
*[class ~= "front-page/front-page"]{
    page-break-before:left;
}
```

Note: This will create an empty page at the beginning of the publication, moving the cover content on the needed side.

For more information, see: Oxygen PDF Chemistry: Controlling Page Breaks.

Related Information:

Double Side Pagination *(on page 1047)*

How to Add a Specific Number of Empty Pages After the Cover Page

In your customization CSS *(on page 981)*, add the following CSS rules:

```
@page my-blank-page {
    /* Hide the page numbers */
    @top-left {content: none};
    @top-right {content: none};
}
*[class ~= 'front-page/front-page']:after(1){
```

```
    page:my-blank-page;
    display:block;
    content: '\2002';
    color:transparent;
    page-break-after:always;
}
*[class ~= 'front-page/front-page']:after(2){
    page:my-blank-page;
    display:block;
    content: '\2002';
    page-break-after:always;
}
*[class ~= 'front-page/front-page']:after(3){
    page:my-blank-page;
    display:block;
    content: '\2002';
    page-break-after:always;
}
```

Note: The `\2002` character is a space that is not shown on the pages, but gives a value for the content property.

Related Information:

How to Force an Odd or Even Number of Pages in a Chapter *(on page 1049)*

How to Add a Copyright Page after the Map Cover (Not for Bookmaps)

Regular DITA maps do not have the concept of a copyright notice. This is available only in the DITA *bookmap* structure.

If you are constrained to using a regular map and you need to add a copyright page between the front cover and the TOC, use the following technique:

In your customization CSS *(on page 981)*, declare a new page layout:

```
@page copyright-notice-page {
  @top-left {
    content:none; /* Clear the headers for the copyright page */
  }
  @top-right {
    content:none;
  }
}
```

The element with the class `front-page/front-page` element contains the title of the publication and generates the cover page. A synthetic `:after` element is created that follows this element and it is placed on a different page.

```
*[class~="front-page/front-page"]:after{
  display:block;

  page: copyright-notice-page; /* Moves the synthetic element on a new page. */

  margin-top:90%; /* use margins to position the text in the page */
  margin-left: 5em;
  margin-right: 5em;
  content: "Copyright 2018-2019 MyCorp Inc. \A All rights reserved";
  text-align:center; /* More styling */
  color:blue;
}
```

If you need to add more content as blocks, use the `:after(2)`, `:after(3)` pseudo-elements:

```
*[class~="front-page/front-page"]:after(2){
  display:block;
  page: copyright-notice-page;  /* Continue on the same page as the first ':after'. */
  content: "Some more styled text";
  color:red;
}
```

If you want to extract information from the document, use the `oxy_xpath()` function. For example, if the copyright info is stored in the map like this:

```
<map ...>
 <topicmeta>
   <copyright>
     <copyryear year="2018"/>
     <copyrholder>MyCorp Inc.</copyrholder>
   </copyright>
 </topicmeta>
...
```

then use this:

```
*[class ~= "front-page/front-page"]:after(3) {
   display: block;
   page: copyright-notice-page;
   content:
   "Year: "
```

```
        oxy_xpath('//*[contains(@class, " front-page/front-page ")]/*[contains(@class, "
 map/topicmeta ")]/*[contains(@class, " topic/copyright ")]/*[contains(@class, " topic/copyryear
 ")]/@year')
    "\A Holder: "
        oxy_xpath('//*[contains(@class, " front-page/front-page ")]/*[contains(@class, "
 map/topicmeta ")]/*[contains(@class, " topic/copyright ")]/*[contains(@class, " topic/copyrholder
 ")]/text()');
    color: green;
}
```

Related Information:

How to Debug XPath Expressions *(on page 986)*

How to Remove the Cover Page and TOC

If you need to hide or remove the cover page, the table of contents or other structures, match the elements with a "front-page/front-page" and "toc/toc" classes in your customization CSS *(on page 981)*:

```
*[class ~= 'map/map'] > *[class ~= 'toc/toc'] {
    display:none !important;
}
*[class ~= 'map/map'] > *[class ~= 'front-page/front-page']{
    display:none !important;
}
*[class~='topic/topic'][is-chapter] {
    -oxy-page-group : auto;
}
```

Front Matter and Back Matter

The *front matter* is a series of topics that are usually placed after the cover page and before the TOC or the content.

The *back matter* is a series of topics that are usually placed after the content of the book.

Front Matter and Back Matter - XML Fragment

In the merged map file *(on page 982)*, the frontmatter topic references are wrapped in a `<frontmatter>` element that has the class `bookmap/frontmatter`. Then, the referenced content is marked with the attribute `@is-frontmatter="true"`:

```
<bookmap xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/" ...>
    <oxy:front-page class="- front-page/front-page ">
       ...
    </oxy:front-page>
```

```
<opentopic:map xmlns:ot-placeholder="http://suite-sol.com/namespaces/ot-placeholder"
class="- toc/toc ">
    ...
    <frontmatter xmlns:dita-ot="http://dita-ot.sourceforge.net/ns/201007/dita-ot"
    class="- map/topicref bookmap/frontmatter ">
      ...
      <topicref class="- map/topicref " href="#unique_1" type="concept">
      ...
    </frontmatter>
</opentopic:map>
<concept
 class="- topic/topic concept/concept "
       is-frontmatter="true"
       topicrefclass="- map/topicref bookmap/bookabstract " ...>
```

For the **DITA Map PDF - based on HTML5 & CSS** transformation type, the merged map is further processed resulting in a collection of HTML5 `<div>` elements. These elements preserve the original DITA `@class` attribute values and add a new value derived from the DITA element name.

```
<div xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/" ...>
    <div class=" front-page/front-page front-page">
       ...
    </div>
    <div class="- toc/toc toc">
      <div class="- map/topicref bookmap/frontmatter topicref frontmatter">
        <div href="#unique_2" type="topic" class="- map/topicref topicref">
        ...
        </div>
    </div>
    <article
     class="- topic/topic concept/concept topic concept nested0"
     is-frontmatter="true"
     topicrefclass="- map/topicref bookmap/bookabstract "  ...>
```

Note: The process also applies for the backmatter topic references inside a `<backmatter>` element with the `bookmap/backmatter` class and referenced content with the `@is-backmatter="true"` attribute both in the merged map and merged HTML files.

Front Matter and Back Matter - Built-in CSS

The built-in CSS rules are in `[PLUGIN_DIR]/css/print/p-bookmap-frontmatter-backmatter.css`. By default, it associates the top-level topics that do not represent chapters to a `matter-page` style of page layout. Each child topic starts on a new page.

Related Information:

Page Headers and Footers *(on page 992)*

How to Remove Page Breaks Between Front Matter Child Topics

If you do not like the fact that all the topics that enter a bookmap frontmatter start on a new page, you can disable this by using the following rules in your customization CSS *(on page 981)*:

```
*[class ~= "map/map"] > *[class ~= "topic/topic"][is-frontmatter]{
   page-break-before: auto;
}
```

How to Style the Front Matter and Back Matter Topics

Style all the Topics with the Same Aspect

All the topics referenced from the `<frontmatter>` and `<backmatter>` bookmap elements are formatted using the `matter-page` as defined in Default Page Definitions *(on page 987)*. In the merged file, the `<backmatter>` and `<frontmatter>` elements are omitted, and their child topic content is matched using a CSS rule like the one below:

```
*[class ~= "map/map"] > *[class ~= "topic/topic"][is-backmatter],
*[class ~= "map/map"] > *[class ~= "topic/topic"][is-frontmatter]{
  page: matter-page;
  ...
}
```

Style the Topics Depending on Their Role

There might be cases when you need to distinguish between certain types of topics that have different roles in your publication:

- Preface
- Notice
- Abstract
- Copyright

These are referenced from the DITA map by specialized `<topicref>` elements, with different class attribute values.

The class attribute values are then passed by the transformation process onto the corresponding topic elements from the merged map content. For example, a topic that was referenced by a `<preface>` map element now has a " bookmap/preface " value in its `@topicrefclass` attribute:

```
<topic
 class="- topic/topic "
 id="unique_1"
```

```
 topicrefclass="- map/topicref bookmap/preface " .. >
...
</topic>
```

This can be used to match and apply various styling choices, or even a particular page layout:

```
@page preface-page {
 background-color:silver;
 @top-center{
  content: "Custom Preface Header";
 }
}
*[class ~= "topic/topic"][@topicrefclass ~= "bookmap/preface"] {
 page: preface-page;
}
```

Numbering

The topics in this section contain some technical details in case you need to fine-tune the way the numbering works.

Numbering - Built-in CSS

The built-in CSS rules are in:

- `[PLUGIN_DIR]/css/print/p-numbering-shallow.css`
- `[PLUGIN_DIR]/css/print/p-numbering-deep.css`
- `[PLUGIN_DIR]/css/print/p-numbering-deep-chapter-scope.css`
- `[PLUGIN_DIR]/css/print/p-numbering-deep-chapter-scope-no-page-reset.css`

The first CSS (shallow) contains rules that add a "Chapter NN" before the first-level topics from the publication, the second one (deep) contains rules that add a deep structure of counters on all topics referenced from the map (at any level), the third one (chapter-scope) creates a chapter scope-oriented numbering (meaning that the numbering for pages, tables, figures, and links to them are reset for each chapter), and the last one is similar to the third except that page numbers do not reset. For more details, see Numbering Types *(on page 1031)*.

Numbering - Input XML Fragments

The numbering affects multiple logical parts of your publication, the table of contents, headers/footers, chapter titles, figures and tables titles:

The Table of Contents

The table of contents is a tree of `<topicref>` elements.

```
<map xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/" ...>
```

```
  <oxy:front-page xmlns:oxy="http://www.oxygenxml.com/extensions/author"
        class=" front-page/front-page ">
        ...
  </oxy:front-page>
  <opentopic:map xmlns:opentopic="http://www.idiominc.com/opentopic" class=" toc/toc ">
    <title class="- topic/title ">Publication Title</title>
    <topicref is-chapter="true" class="- map/topicref " ... >
      <topicmeta class="- map/topicmeta " ... >
        <navtitle href="#unique_1" class="- topic/navtitle ">Overview</navtitle>
            ...
      </topicmeta>
      <topicref class="- map/topicref " ...>
        <topicmeta class="- map/topicmeta " data-topic-id="dcpp_resources">
          <navtitle href="#unique_2" class="- topic/navtitle ">Resources</navtitle>
            ...
        </topicmeta>
      </topicref>
        ...
  </opentopic:map>
...
</map>
```

Note: The `<opentopic:map>` element contains the effective table of contents structure.

Note: The TOC items are the elements with the class: `- map/topicref`.

Note: The ones identified as chapters have the `@is-chapter` attribute set.

For the **DITA Map PDF - based on HTML5 & CSS** transformation type, the merged map is further processed resulting in a collection of HTML5 `<div>` elements. These elements preserve the original DITA `@class` attribute values and add a new value derived from the DITA element name.

```
<div class="- map/map map" ...>
  <div
        class=" front-page/front-page front-page">
        ...
  </div>
  <div class=" toc/toc toc">
    <div class="- topic/title title">Publication Title</title>
    <div is-chapter="true" class="- map/topicref topicref" ... >
      <div class="- map/topicmeta topicmeta" ... >
        <div href="#unique_1" class="- topic/navtitle navtitle">Overview</div>
            ...
      </div>
      <div class="- map/topicref " ...>
```

```
        <div class="- map/topicmeta " data-topic-id="dcpp_resources">
          <div href="#unique_2" class="- topic/navtitle ">Resources</div>
            ...
        </div>
      </div>
        ...
  </div>
...
</div>
```

The Header and Footers

These are based on string sets generated for the titles. The complete set of strings is defined in: `[INSTALLATION_DIR]/css/print/p-pages-and-headers.css`.

The CSS rules that build the string sets are matching the map title from the front page and the titles from the content.

```
<oxy:front-page xmlns:oxy="http://www.oxygenxml.com/extensions/author">
        <oxy:front-page-title>
            <title class="- topic/title ">Publication Title</title>
        </oxy:front-page-title>
</oxy:front-page>
```

For the **DITA Map PDF - based on HTML5 & CSS** transformations:

```
<div class=" front-page/front-page front-page">
        <div class=" front-page-title/front-page-title front-page-title">
            <div class="- topic/title title ">Publication Title</div>
        </div>
</div>
```

The main content is organized as follows:

```
<map xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/" ...>
    ...
    <opentopic:map xmlns:opentopic="http://www.idiominc.com/opentopic">
      ...
    </opentopic:map>
    <topic is-chapter="true" oid="dcpp_overview">
        <title class="- topic/title ">Overview</title>
        <body class="- topic/body ">
            ...
        </body>
        <topic class="- topic/topic " id="unique_2" oid="dcpp_resources">
            <title class="- topic/title ">Resources</title>
            ...
```

```
        </topic>
        <topic class="- topic/topic " id="unique_2" oid="dcpp_parameters">
            <title class="- topic/title ">Parameters</title>
            ...
        </topic>
    </topic>
```

For the **DITA Map PDF - based on HTML5 & CSS** transformations:

```
<div class=" map/map map" ...>
    ...
   <div class=" toc/toc toc">
      ...
    </div>
    <div is-chapter="true" oid="dcpp_overview" class="- topic/topic topic">
        <div class="- topic/title title">Overview</title>
        <div class="- topic/body body">
            ...
        </div>
        <div class="- topic/topic topic" id="unique_2" oid="dcpp_resources">
            <div class="- topic/title title">Resources</div>
            ...
        </div>
        <div class="- topic/topic topic" id="unique_2" oid="dcpp_parameters">
            <div class="- topic/title title">Parameters</div>
            ...
        </div>
    </div>
```

Note: The topic content comes after the `<opentopic:map>` element.

Note: The child topics are the elements that have the class `- topic/topic` included in the parents.

Note: The ones identified as chapters have the `@is-chapter` attribute set.

The Titles of Chapters

The titles from the content are children of the topics:

```
<topic class="- topic/topic " id="unique_2" oid="dcpp_parameters">
        <title class="- topic/title ">Parameters</title>
            ...
</topic>
```

For the **DITA Map PDF - based on HTML5 & CSS** transformations:

```
<div class="- topic/topic topic" id="unique_2" oid="dcpp_parameters">
        <div class="- topic/title title ">Parameters</div>
```

```
            ...
</div>
```

Note: The title elements have the class: `- topic/title`. The actual element name can be different.

Numbering Types

The type of numbering is controlled by the parameter `args.css.param.numbering`.

This parameter activates various sets of CSS rules from the built-in CSS. By default, only the first-level topics (the chapters) are numbered (`shallow` numbering). The following values are accepted:

Table 35. Types of Numbering

Value	Chapters	Sections	Figures & Tables	Pages
`shallow`	numbered	no	counted from the start of the publication	from the start of the publication
`deep`	numbered	numbered	counted from the start of the publication	from the start of the publication
`deep-chapter-scope`	numbered	numbered	numbering is restarted at the beginning of each chapter, adds the chapter number in their titles (and in the links to them), and in the list of tables and list of figures sections	restarted at the beginning of each chapter
`deep-chapter-scope-no-page-reset`	numbered	numbered	numbering is restarted at the beginning of each chapter, adds the chapter number in their titles (and in the links to them), and in the list of tables and list of figures sections	from the start of the publication

Note: The `deep-chapter-scope` and `deep-chapter-scope-no-page-reset` values are only available for the **DITA Map PDF - based on HTML5 & CSS** transformation scenario.

Examples

Shallow

Each chapter (or first-level topic) is numbered, but nested topics are not numbered. Figures, tables, and pages are numbered sequentially from the start of the publication and they do not reset.

```
1. First Chapter
    Page 1
```

```
      Topic
        Table 1
        Table 2
      Topic
    Page 2
        Table 3
2. Second Chapter
    Page 3
      Topic
        Table 4
        Table 5
      Topic
    Page 4
```

Deep

Each chapter (or first-level topic) and nested topics are numbered (these are also prefixed with the chapter number). Figures, tables, and pages are numbered sequentially from the start of the publication and they do not reset.

```
1. First Chapter
    Page 1
      Topic 1.1
        Table 1
      Topic 1.2
        Table 2
    Page 2
        Table 3
2. Second Chapter
    Page 3
      Topic 2.1
        Table 4
        Table 5
      Topic 2.2
    Page 4
```

Deep Chapter Scope

Each chapter (or first-level topic) is independent (so it can be read separately, as a separate part of your publication). The nested topics, page, figure, and table counters (and links to them) restart at each chapter. The general cross reference links also display the chapter number before the page number to clearly specify the target.

```
1. First Chapter
    Page 1.1
      Topic 1.1
```

```
        Table 1.1
        Link to page 2.2
      Topic 1.2
    Page 1.2
        Table 1.2
2. Second Chapter
    Page 2.1
      Topic 2.1
        Table 2.1
        Table 2.2
        Table 2.3
      Topic 2.2
        Table 2.4
    Page 2.2
       Link to page 1.1
```

Deep Chapter Scope No Page Reset

Each chapter (or first-level topic) is independent (so it can be read separately, as a separate part of your publication). The nested topics, figure, and table counters (and links to them) restart at each chapter, but the page numbers do not reset. The generic cross reference links contain only the page number.

```
1. First Chapter
    Page 1
      Topic 1.1
        Table 1.1
        Link to page 4
      Topic 1.2
    Page 2
        Table 1.2
2. Second Chapter
    Page 3
      Topic 2.1
        Table 2.1
        Table 2.2
        Table 2.3
      Topic 2.2
        Table 2.4
    Page 4
       Link to page 1
```

How to Remove the 'Chapter NN' Prefix from Table of Contents and Content.

If you are using the shallow numbering CSS (this is the default), and you want to hide the "Chapter" prefix, use the following rules in your customization CSS *(on page 981)*:

For the TOC:

```
*[class ~= "map/topicmeta"] > *[class ~= "topic/navtitle"]:before {
    display:none !important;
}
```

For the titles in the content, change the content of the title `:before` (this example just uses the chapter number):

```
*[class ~= "topic/topic"][is-chapter]:not([is-part]) > *[class ~= "topic/title"]:before {
    content: counter(chapter) ". " !important;
}
```

How to Activate Continuous Page Numbering

The page numbers are reset to one of two cases:

- On the first topic that follows the TOC.
- On the first topic from the index.

To avoid the page counter from being reset, use the following rules in your customization CSS *(on page 981)*:

```
*[class ~= "map/map"] > *[class ~= "toc/toc"] + *[class ~= "topic/topic"],
*[class ~= "index/groups"] {
  counter-reset: none;
}
```

The index page also changes the format of the page numbers to lower alpha characters. To switch to decimal characters, use:

```
@page index {
  @bottom-center {
    content: counter(page, decimal)
  }
}
```

How to Use Part, Chapter, and Subtopics Numbers in Links

This topic is applicable if you have enabled deep numbering *(on page 1031)*. Suppose you have a link in the third chapter that points to a paragraph in the second subtopic of the first chapter and you need this structural information (1.2) presented to the user, just after the link text. To do this, you can use the `target-counters` CSS function to extract the entire context of the counters from the target. The `chapter-and-sections` built-in counter is already updated with both the chapter number and the nested topics:

```
*[class ~= "topic/xref"]:after {
 content: target-counters(attr(href), chapter-and-sections, ".") !important;
}
```

This counter does not include the part number, so take care when linking between parts (you may consider adding the target part number explicitly):

```
*[class ~= "topic/xref"]:after {
 content: "[" target-counter(attr(href), part, upper-roman) "/" target-counters(attr(href),
 chapter-and-sections, ".") "]" !important;
 color:blue;
}
```

Related Information:

Numbering Types *(on page 1031)*

Table of Contents

The table of contents is a hierarchy of topic titles with links to the topic content.

For plain maps, the TOC is automatically generated. For DITA bookmaps, you will need to add a `<toc>` element in the `<booklists>` element (inside the `<frontmatter>`):

```
<bookmap>
  ...
  <frontmatter>
    <booklists>
      <toc/>
      <figurelist/>
      <tablelist/>
    </booklists>
  </frontmatter>
  ...
  ...
```

Related Information:

Table of Contents for Chapters (Mini TOC) *(on page 1041)*

List of Tables/Figures *(on page 1046)*

Index *(on page 1052)*

Table of Contents - XML Fragment

In the merged map file *(on page 982)*, the `<opentopic:map>` contains a hierarchy of `<topicref>` elements, or other elements (such as `<chapter>` or `<part>`) that are specializations of `<topicref>`.

Each of the `<topicref>` elements include a *metadata* section that includes the topic title.

```
<bookmap ...>
<oxy:front-page> ... </oxy:front-page>
<oxy:front-matter> ... </oxy:front-matter>
<opentopic:map xmlns:opentopic="http://www.idiominc.com/opentopic" class="- toc/toc ">
  <oxy:toc-title xmlns:oxy="http://www.oxygenxml.com/extensions/author" empty="true"
      class="- toc/title "/>
  <booktitle class="- topic/title bookmap/booktitle ">
      <booklibrary class="- topic/ph bookmap/booklibrary ">Retro Tools</booklibrary>
      <mainbooktitle class="- topic/ph bookmap/mainbooktitle ">Tasks</mainbooktitle>
      <booktitlealt class="- topic/ph bookmap/booktitlealt ">Product Tasks</booktitlealt>
  </booktitle>
  <chapter is-chapter="true"
      class="- map/topicref bookmap/chapter " href="#unique_5" type="topic">
      <topicmeta class="- map/topicmeta " data-topic-id="installing">
          <navtitle href="#unique_5" class="- topic/navtitle ">Installing</navtitle>
          ...
      </topicmeta>
      <topicref class="- map/topicref " href="#unique_6" type="task">
          <topicmeta class="- map/topicmeta " data-topic-id="installstorage">
              <navtitle href="#unique_6" class="- topic/navtitle ">Installing</navtitle>
                  ...
          </topicmeta>
               ...
      </topicref>
           ...
  </chapter>
```

For the **DITA Map PDF - based on HTML5 & CSS** transformation type, the merged map is further processed resulting in a collection of HTML5 `<div>` elements. These elements preserve the original DITA `@class` attribute values and add a new value derived from the DITA element name.

```
<div class="- bookmap/bookmap map/map map bookmap" ...>
<div class="- front-page/front-page front-page"> ... </div>
<div class="- bookmap/frontmatter frontmatter"> ... </div>
<div class=" toc/toc toc">
 <div class="toc/toc-title toc-title" empty="true"/>
 <div class="- topic/title bookmap/booktitle booktitle">
      <div class="- topic/ph bookmap/booklibrary booklibrary">Retro Tools</div>
      <div class="- topic/ph bookmap/mainbooktitle mainbooktitle">Tasks</div>
      <div class="- topic/ph bookmap/booktitlealt booktitlealt">Product Tasks</div>
 </div>
 <div is-chapter="true"
```

```
   class="- map/topicref bookmap/chapter topicref chapter " href="#unique_5" type="topic">
   <div class="- map/topicmeta topicmeta" data-topic-id="installing">
       <div href="#unique_5" class="- topic/navtitle navtitle">Installing</div>
       ...
  </div>
  <div class="- map/topicref  topicref chapter " href="#unique_6" type="task">
       <div class="- map/topicmeta topicmeta" data-topic-id="installstorage">
           <div href="#unique_6" class="- topic/navtitle navtitle">Installing</div>
                ...
  </div>
                ...
 </div>
            ...
</div>
```

Note: The `<oxy:toc-title>` element is used as a placeholder for the name of the TOC. For instance, you can use the string "Contents", specified on a pseudo-element, in the CSS.

Table of Contents - Built-in CSS

The built-in CSS rules are in: `[PLUGIN_DIR]/css/print/p-toc.css`.

Related Information:

Page Headers and Footers *(on page 992)*

How to Increase TOC Depth

By default, only the first two levels of topics are displayed in the Table of Contents of the PDF output.

The CSS rule (see Table of Contents - Built-in CSS *(on page 1037)*) that hides topics on higher levels is:

```
/* Hide sections below level 3. */
*[class ~= "map/topicref"][is-chapter] >
    *[class ~= "map/topicref"]:not([is-chapter]) >
    *[class ~= "map/topicref"] >
    *[class ~= "map/topicref"] {
    display: none;
}
```

If you want to increase the TOC depth so that topic references on level 3 or higher are visible, you can overwrite this rule in your customization CSS like this:

```
*[class ~= "map/topicref"][is-chapter] >
    *[class ~= "map/topicref"]:not([is-chapter]) >
    *[class ~= "map/topicref"] >
    *[class ~= "map/topicref"]{
```

```
    display:block;
}
```

How to Style the Table of Contents Entries

Note: Each of the items from the table of contents is an element that has the `map/topicref` class.

The following example uses the italic font for the label and changes the color and style of the connecting line between the title and the page number.

In your customization CSS *(on page 981)*, add the following two selectors:

```
/* The toc item label - the topic title */
*[class ~= "map/topicref"] *[class ~= "topic/navtitle"] {
    font-style:italic;
    color: navy;
}
/* The dotted line between the topic name and the page number. */
*[class ~= "map/topicref"] *[class ~= "topic/navtitle"]:after {
    content: leader('-') target-counter(attr(href), page);
    color: navy;
}
```

And if you need to alter the indent of the nested table of content items, use the following selector:

```
*[class ~= "map/topicref"] *[class ~= "map/topicref"] {
    margin-left: 1em;
}
```

The numbers can be styled like this:

```
*[class ~= "map/topicref"] > *[class ~= "map/topicmeta"]:before,
*[class ~= "map/topicref"]
    > *[class ~= "map/topicmeta"] > *[class ~= "topic/navtitle"]:before{
  color:blue;
}
```

The following is an example of customizing the font size for the items representing chapters. The chapters are level one topics and are marked in the merged DITA document TOC with the attribute `@is-chapter`.

```
*[class ~= "map/topicref"][is-chapter = "true"]  > *[class ~= "map/topicmeta"] > *[class
 ~= "topic/navtitle"]{
  font-size:2em;
}
```

How to Change the Header of the Table of Contents

In the built-in CSS, there is a page named *table-of-contents*. The default is to have the word 'Contents' in its header (this is localized, using the `toc-header` string defined in the `p-18n.css`) alternating in the left or right side of the header:

```
@page table-of-contents:left {
    @top-left {
        content: string(toc-header) " | " counter(page, lower-roman);
        font-size: 8pt;
    }
}
@page table-of-contents:right {
     @top-right {
         content: string(toc-header) " | " counter(page, lower-roman);
         font-size: 8pt;
     }
}
```

If you need to change this string, or change the color, you should use the following `@page` selectors as a starting point in your customization CSS *(on page 981)*:

```
@page table-of-contents:left {
    @top-left {
        content: "My publication table of contents | " counter(page, lower-roman);
        color:red;
    }
}
@page table-of-contents:right {
     @top-right {
        content: "My publication table of contents | " counter(page, lower-roman);
        color:red;
     }
}
```

Related Information:

Localization *(on page 1073)*

How to Make the Table of Contents Start on an Odd Page

In your customization CSS *(on page 981)*, add the following snippet for the *table-of-contents* page:

```
@page table-of-contents{
    -oxy-initial-page-number: auto-odd;
}
```

Related Information:

Double Side Pagination *(on page 1047)*

How to Display a Topic Before the Table of Contents

To display a topic before the *table-of-contents* page, follow these steps:

1. Make sure the topic is referenced on the first level in the DITA map.
2. Set the `@outputclass` to `before-toc` on the `<topicref>`.

```
<topicref href="pathToMyTopic" outputclass="before-toc">
```

Result: When the PDF is processed, the topic will automatically appear before the table of contents.

Related Information:

Controlling the Publication Content *(on page 1103)*

How to Display Short Descriptions in the TOC

To display the short descriptions from the topics in the table of contents, you need to make the `<shortdesc>` element visible.

The following example only makes the short descriptions associated with the chapters visible. The chapters are level one topics and are marked in the merged DITA document TOC with the attribute `@is-chapter`.

In your customization CSS *(on page 981)*, add the following CSS selector:

```
*[class ~= "map/topicref"][is-chapter = "true"] > *[class ~= "map/topicmeta"] > *[class
 ~= "map/shortdesc"] {
  display:block; /* The default is none - the shortdesc is hidden. */
  color:gray;
}
```

Note: If you need all the TOC item short descriptions to be visible, remove the `[is-chapter]` condition.

How to Remove Entries from the TOC

To remove entries from the table of contents, set the `@toc="no"` attribute on the topicrefs from the map that need to be removed. This is sometimes desirable for the topics listed in the frontmatter or backmatter when using a bookmap.

How to Hide the TOC

To hide the TOC, you have multiple options:

- Use a DITA bookmap instead of a DITA map, and omit the `<toc>` element from the `<booklists>`. An example bookmap can be found in the DITA 1.3 Spec. This is the best approach.
- Use the transformation parameter: hide.frontpage.toc.index.glossary *(on page 960)*.
- Use a `display:none` property to hide the element that contains the TOC structure, and also remove it from the PDF bookmarks tree:

```
*[class ~= "map/map"] > *[class ~= "toc/toc"] {
  display:none;
}
*[class ~= "map/map"] > *[class ~= "toc/toc"] > *[class ~= "toc/title"]{
  bookmark-label: none;
  -ah-bookmark-label: none;
}
```

Related Information:

Transformation Parameters *(on page 956)*

Table of Contents for Chapters (Mini TOC)

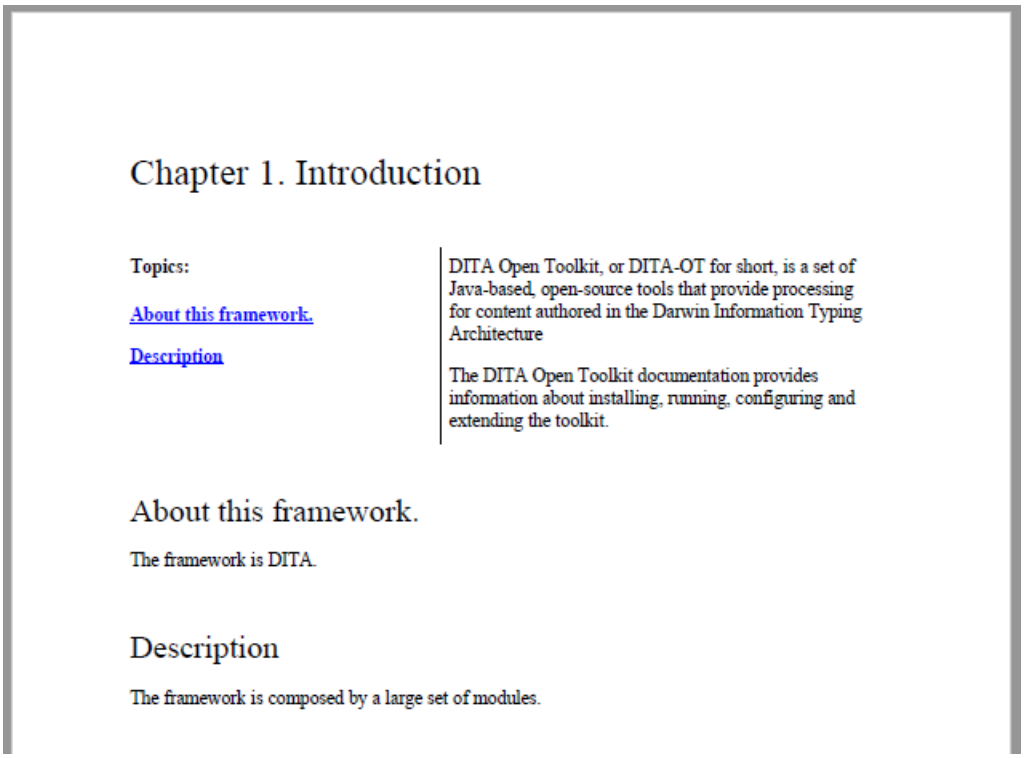
To add a mini table of contents for each chapter, you need to:

- Use DITA bookmaps instead of regular maps.
- Set the `args.chapter.layout` transformation parameter to either of the following values: **MINITOC** or **MINITOC-BOTTOM-LINKS**.

Note: If the chapter does not have child topics, it will not have a mini TOC in the PDF output.

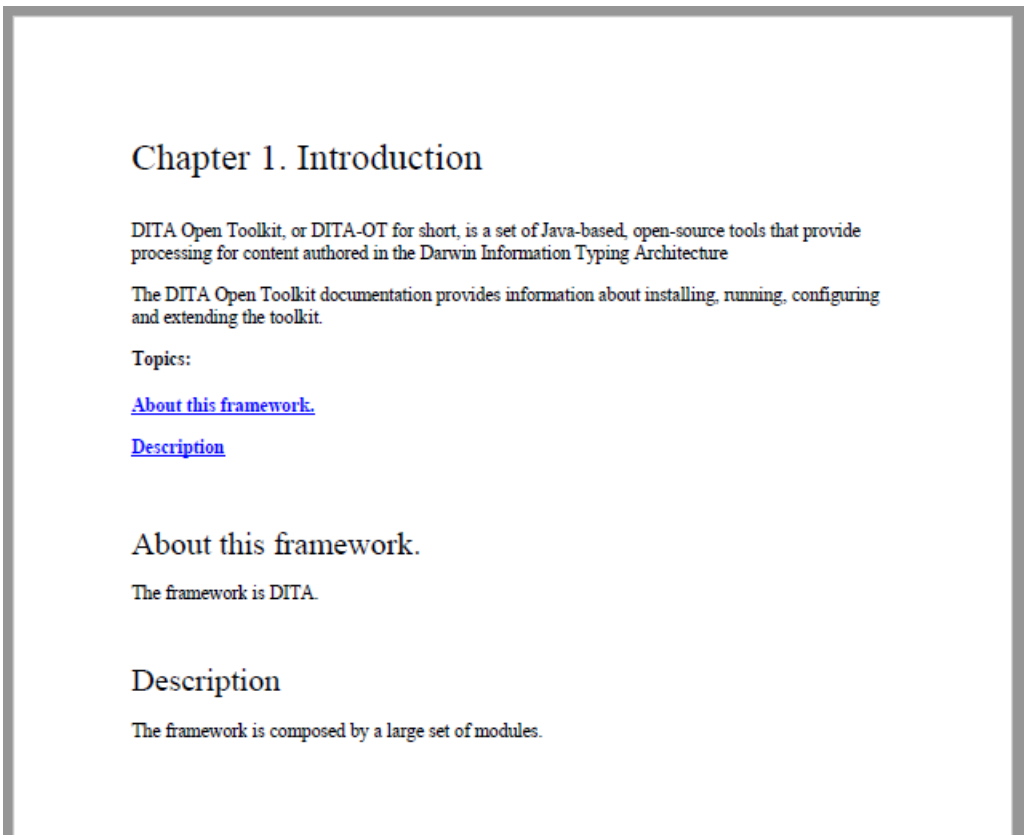
Layout for MINITOC

This table of contents is positioned between the chapter title and the chapter child topics. It consists of a list of links pointing to the child topics, positioned in the left side of the page, and a description in the right side. This content is collected from the topic file referenced by the chapter `<topicref>` in the map.



Layout for MINITOC-BOTTOM-LINKS

This table of contents is positioned between the chapter title and the chapter child topics. It consists of a chapter description and list of links pointing to the child topics, under the description. This description is collected from the topic file referenced by the chapter `<topicref>` in the map.



The above chapter example has the following DITA map fragment:

```
<chapter href="topics/chapter-introduction.dita">
  <topicref href="topics/introduction-about.dita" />
  <topicref href="topics/introduction-description.dita" />
</chapter>
```

The `chapter-introduction.dita` file provides the description content that is in the right side of the page. The children `<topicref>` elements generate the mini TOC links.

Table of Contents for Chapters (Mini TOC) - XML Fragment

In the merged XML file, the mini TOC is built from a related links section and some `<div>` elements that wrap the entire mini TOC and the description area.

chapter/minitoc

Wraps the entire structure, including the content of the chapter `<topicref>`.

chapter/minitoc-links

Wraps the `<related-links>` element. Note that the label of the related links list is internationalized.

chapter/minitoc-desc

Contains the entire content of the topic file referenced by the chapter `<topicref>` element in the map.

```
<div class="- topic/div chapter/minitoc ">
  <div class="- topic/div chapter/minitoc-links ">
    <related-links class="- topic/related-links ">
      <linklist class="- topic/linklist ">
        <desc class="- topic/desc ">
          <ph class="- topic/ph chapter/minitoc-label ">Topics: </ph>
        </desc>
        <link class="- topic/link " href="#unique_2" type="topic" role="child">
          <linktext class="- topic/linktext ">About this framework.</linktext>
        </link>
        <link class="- topic/link " href="#unique_3" type="topic" role="child">
          <linktext class="- topic/linktext ">Description</linktext>
        </link>
      </linklist>
    </related-links>
  </div>
  <div class="- topic/div chapter/minitoc-desc ">
    <shortdesc class="- topic/shortdesc ">DITA Open Toolkit, or DITA-OT for
short, is a set of Java-based, open-source tools that provide processing
for content authored in the Darwin Information Typing
Architecture</shortdesc>
    <body class="- topic/body ">
```

```
      <p class="- topic/p ">The DITA Open Toolkit documentation provides information about
        installing, running, configuring and extending the toolkit.</p>
    </body>
  </div>
</div>
```

When using the `pdf-css-html5` transformation, this structure is converted to a set of HTML elements, preserving the class values:

```
<div class="- topic/div chapter/minitoc div minitoc">
 <div class="- topic/div chapter/minitoc-links div minitoc-links">
  <div class="wh_related_links">
   <nav role="navigation" class="- topic/related-links related-links">
    <div class="- topic/linklist linklist linklistwithchild">
     <div class="- topic/desc desc">
      <span class="- topic/ph chapter/minitoc-label ph minitoc-label">Topics: </span>
      </div>
       <ul class="linklist">
        <li class="- topic/link link ulchildlink" href="#unique_2"
       type="topic" role="child">
         <strong>
          <a href="#unique_2">About this framework.</a>
         </strong>
           <br/>
        </li>
        <li class="- topic/link link ulchildlink" href="#unique_3"
       type="topic" role="child">
         <strong>
           <a href="#unique_3">Description</a>
         </strong>
           <br/>
        </li>
       </ul>
      </div>
     </nav>
    </div>
  </div>
  <div class="- topic/div chapter/minitoc-desc div minitoc-desc">
    <div class="- topic/body body">
      <p class="- topic/shortdesc shortdesc">DITA Open Toolkit, or DITA-OT for short,
  is a set of Java-based, open-source tools that provide processing for content
  authored in the Darwin Information Typing Architecture</p>
      <p class="- topic/p p">The DITA Open Toolkit documentation provides information
        about installing, running, configuring and extending the toolkit.</p>
```

```
    </div>
  </div>
</div>
```

Table of Contents for Chapters (Mini TOC) - Built-in CSS

The built-in CSS rules are in: `[PLUGIN_DIR]/css/print/p-chapters-minitoc.css`.

How to Style the Table of Contents for Chapters (Mini TOC)

Suppose that you do not want the links and the chapter description to be side by side, but instead place the links above the description. Also, you may choose to remove the label above the links and put all the links in a colored rectangle with decimal numbers before them.

In your customization CSS *(on page 981)*, add the following selectors:

```
/* Change from inline to blocks to stack them one over the other. */
*[class~="chapter/minitoc-desc"],
*[class~="chapter/minitoc-links"] {
 display: block;
 width: 100%;
}
/* No need for the 'Topics:' label. */
*[class~="chapter/minitoc-links"] *[class~="topic/desc"] {
 display:none;
}
/* Add background for the links list. */
*[class~="chapter/minitoc-links"] {
 background-color:silver;
 padding:0.5em;
}
/* Remove the border and the padding from the description. We do not need that separator. */
*[class~="chapter/minitoc-desc"] {
 border-left:none;
 padding-left:0;
}
/* Add a number before each of the links. */
*[class~="chapter/minitoc-links"] *[class~="topic/link"] {
 display:list-item;
 list-style-type:decimal;
 margin-left:1em;
}
```

Related Information:

How to Speed up CSS Development and Debugging *(on page 985)*

List of Tables/Figures

To activate these:

1. The map must be a DITA bookmap.
2. There must be a `<figurelist>` or `<tablelist>` in the frontmatter or backmatter. In the following example, both of the lists are added just after the table of contents (the `<toc>` element is the placeholder where the table of contents will be created):

```
<frontmatter>
   <booklists>
     <toc/>
     <figurelist/>
     <tablelist/>
   </booklists>
 </frontmatter>
```

How to Set a Header for a List of Tables/Figures

Suppose you want to set the headline "Figure List" on the second and subsequent pages associated to a list of figures and something similar for a list of tables.

Start by associating pages to the list of figures and tables from the merged file:

```
*[class~="placeholder/tablelist"] {
   page:tablelist;
   color:green;
}
*[class~="placeholder/figurelist"]{
   page:figurelist;
   color:green;
}
```

Note: The "placeholder/tablelist" is the class name of the output generated from the `<tablelist>` bookmap element.

Then define the pages:

```
@page figurelist {
    @top-left { content: none; }
    @top-center { content: "Figure List"; }
    @top-right { content: none; }
```

```
}
@page figurelist:first {
    @top-left { content: none; }
    @top-center { content: none; }
    @top-right { content: none; }
}
@page tablelist {
    @top-left { content: none; }
    @top-center { content: "Table List"; }
    @top-right { content: none; }
}
@page tablelist:first {
    @top-left { content: none; }
    @top-center { content: none; }
    @top-right { content: none; }
}
```

How to Remove the Numbers Before a List of Tables or Figures

Suppose you need to remove the "Figure NN" prefix before each entry of a list of figures.

An entry in the generated list of figures from the merged map looks like this:

```
<entry class="- listentry/entry " href="#unique_6_Connect_42_fig_rjy_spn_xgb">
  <prefix class="- listentry/prefix ">Figure</prefix>
  <number class="- listentry/number ">4</number>
  <title class="- topic/title ">This is another figure</title>
</entry>
```

For the HTML merged map, the element names are all `<div>` elements but they have the same class.

So, to hide the label and the number, use:

```
*[class~="listentry/prefix"],
*[class~="listentry/number"] {
    display:none;
}
```

This works for both a list of tables and list of figures since the structure of each entry is the same.

To make it more specific (for example, to apply it only for the list of figures), you can add the selector:

```
*[class~="placeholder/figurelist"] *[class~="listentry/prefix"],
*[class~="placeholder/figurelist"] *[class~="listentry/number"] {
    display:none;
}
```

Double Side Pagination

By default, the processor generates pages that are mirror images (the right page has the header on the right side, the left pages have the header on the left side). The chapters follow one another with no constraint on the page side.

Note: For a plain DITA map, the chapters are the `<topicref>` elements that are placed on the first level. For bookmaps, the chapters are the topics referenced by a `<chapter>` element.

This section contains information about how to position the start of the chapters on an odd folio number. Some of the CSS rules given here as examples are already listed in: `[INSTALLATION_DIRECTORY]/css/print/p-optional-double-side-pagination.css`. You may choose to import this file from your customization CSS *(on page 981)*.

How to Start Chapters on Odd Pages

A common use case is to arrange the chapters of the publication to start on an odd page number.

In your customization CSS *(on page 981)*, add the following:

```
@page chapter {
 -oxy-initial-page-number: auto-odd;
}
@page table-of-contents {
 -oxy-initial-page-number: auto-odd;
}
```

Supported values for `-oxy-initial-page-number` include: **auto**, **auto-even**, **auto-odd**, or a number.

How to Style the Empty (Blank) Pages

By making the chapters start on an odd page, the CSS processor might add blank pages to the previous page sequence as padding.

To style those blank pages add the following code in your customization CSS *(on page 981)*:

```
@page chapter:blank, table-of-contents:blank {
     @top-left      { content: none; }
     @top-center    { content: none; }
     @top-right     { content: none; }
     @bottom-left   { content: none; }
     @bottom-center { content: none; }
     @bottom-right  { content: none; }
}
```

Note: This just removes the headers and footers, but you can use a background image or a header with "Intentionally left blank" text.

Related Information:

How to Add a Background Image for the Cover *(on page 1018)*

How to Force an Odd or Even Number of Pages in a Chapter

Another use case is to specify a number of pages for a section. Suppose that you have a table of contents that follows the cover page and you need to have an even number of pages. Hence, the next chapter would start on an even page.

In your customization CSS *(on page 981)*, use the `-oxy-force-page-count` property with an even value:

```
@page table-of-contents {
 -oxy-force-page-count: even;
}
```

Supported values for `-oxy-force-page-count` include: **even**, **odd**, **end-on-even**, **end-on-odd**, **auto**, **no-force**.

How to Style the First page of a Chapter

You can use the `:first` page rule selector to control how the first page of a chapter looks. Suppose that you have defined the following layout for your default page and you want to put the publication title (the `maptile` string) on the header of the first page (instead of the chapter name that is displayed on this page):

In your customization CSS *(on page 981)*, add the following:

```
@page chapter:first {
 @top-right-corner { content: string(maptitle); }
 @top-left { content: none; }
}
```

Multiple Column Pages

This section contains information about how to handle pages that have multiple columns.

How to Use a Two Column Layout

Change Layout for Predefined Pages.

First you need to identify which of the pages need to be changed. Pages are already defined for the cover page, table of contents, chapter content, and others. The complete list is here: Default Page Definitions *(on page 987)*.

Next, add the `column-count` and `column-gap` properties to that page. For example:

```
@page chapter{
  column-count:2;
  column-gap:1in;
}
```

If you need some of the elements to expand on all the columns, use the `column-span:all` CSS property. The next snippet makes the chapter titles span both columns:

```
*[class ~= "topic/topic"][is-chapter] > *[class ~= "topic/title"] {
  column-span:all;
}
```

Limitation: You cannot use multiple column configurations on the same page. Oxygen XML Developer Eclipse plugin only takes the `column-count` and `column-gap` properties into account if they are set on `@page` rules, not on elements from the content.

Change Layout for a Specific Topic.

If you need to have a different column layout just for one topic, you can use the following technique:

1. Define an `outputclass` on the topic root element.

   ```
   <topic outputclass="two_columns" ...
   ```

2. Define a CSS rule that changes the `page` property for the matching element.

   ```
   *[class ~= "two_columns"],
   *[outputclass ~= "two_columns"]{
     page: two_column_page !important;
   }
   ```

   Tip: In the selector, use the `class` attribute for the HTML transformation, or `outputclass` for the direct transformation, or leave them both if you are not sure.

   Note: The topics from the first level use the `chapter` page. You must use `!important` because the built-in rules are more specific and you need to override the `page` property.

3. Define a page layout.

   ```
   @page two_column_page {
     column-count: 2;
   }
   ```

Note that the topic will be separated from other sibling topics with different page layouts by page breaks.

Related Information:

Page Formatting in Oxygen PDF Chemistry

PDF Bookmarks

The PDF Bookmarks are used to generate a hierarchical structure similar to a table of contents in a specialized view of your PDF Reader.

By default, the titles defined in the topics are used as bookmark labels.

PDF Bookmarks - Built-in CSS

The PDF bookmarks are generated by matching the titles from the topics in the content. The built-in CSS rules are in: `[PLUGIN_DIR]/css/print/p-bookmarks.css`.

How to Change the Bookmark Labels using the Navigation Title

To change the bookmark labels, you can specify a navigation title in a DITA map or topic.

This will be used as the bookmark label instead of the topic title in the table of contents and the bookmark views. There are two possibilities to do specify it:

1. Place a `<navtitle>` element in the topic reference in the DITA map:

```
...
<topicref href="topics/my_topic.dita" locktitle="yes">
   <topicmeta>
      <navtitle>Introduction</navtitle>
   </topicmeta>
</topicref>
...
```

   Note: As a best practice, a `@locktitle` attribute with the value 'yes' is needed to activate the navigation title. The plugin applies the navigation title even if the attribute is missing.

2. Place a `<navtitle>` element in the topic, as a title alternative.

```
<topic id="other_topic" xml:lang="en-us">
  <title>Normal Title</title>
  <titlealts>
    <navtitle>Navigation Title</navtitle>
  </titlealts>
  <body>
...
```

How to Control the Depth of the Sections From the PDF Bookmarks.

By default, the PDF bookmarks are generated for up to 7 levels. If you need to limit them (for example to 2 levels), you can use the following CSS rules in your customization CSS *(on page 981)*:

```
*[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"] > *[class~="topic/title"],
*[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"] >
 *[class~="topic/title"],
*[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"]
 *[class~="topic/topic"] > *[class~="topic/title"],
*[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"]
 *[class~="topic/topic"] *[class~="topic/topic"] > *[class~="topic/title"],
```

```
*[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"]
 *[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"] > *[class~="topic/title"]
 {
  bookmark-label:none;
}
```

These rules clear the labels generated by the titles starting with the depth 3 (the topic nesting level is given by the selectors `*[class~="topic/topic"]`).

How to Specify the Open/Closed PDF Bookmark State

If you want to specify the initial state for the bookmarks (opened/expanded or closed/collapsed), you can use the `bookmark-state` property in your customization CSS *(on page 981)*.

For example, to specify that all bookmarks for the first three levels are opened (expanded) in the initial state, use:

```
*[class~="topic/topic"] > *[class~="topic/title"],
*[class~="topic/topic"] *[class~="topic/topic"] > *[class~="topic/title"],
*[class~="topic/topic"] *[class~="topic/topic"] *[class~="topic/topic"] > *[class~="topic/title"] {
     bookmark-state:open;
}
```

How to Remove the Numbering From the PDF Bookmarks

By default, the PDF bookmark labels are generated while taking the text set before the chapters titles into account. Since this usually contains the part, chapter, or section numbers, the PDF Bookmarks will make use of them.

The solution is to remove the `content(before)` from the `bookmark-label`, leaving just the `content(text)`.

In your customization CSS *(on page 981)*, add the following CSS rules:

```
*[class~="topic/topic"] > *[class~="topic/title"] {
  bookmark-label: content(text);
  -ah-bookmark-label: content();
}
```

Important: This is a simple example that does not use the possible navigation titles, just the content of the `<title>` element. Copy and modify the built-in CSS for the full CSS rule that matches the `<title>` and `<titlealts>` elements:

```
*[class~="topic/topic"]:has(*[class~="topic/titlealts"]) > *[class~="topic/title"] {...}
```

Related Information:

Numbering *(on page 1027)*

Index

The content of an `<indexterm>` element is used to produce an index entry in the generated index. You can nest `<indexterm>` elements to create multi-level indexes. The content is not output as part of the topic content, only as part of the index tree.

To add an index to your publication, you just need to add `<indexterm>` elements inside the `<prolog>` section (inside a `<metadata>` element):

```
<title>The topic title.</title>
<prolog>
  <metadata>
    <keywords>
      <indexterm>Installing <indexterm>Water Pump</indexterm></indexterm>
    </keywords>
  </metadata>
</prolog>
<body>
  .....
```

or in the content itself:

```
...
<p>Open the lid then turn the body pump to the right.
<indexterm>Installing <indexterm>Water Pump</indexterm></indexterm>
</p>
...
```

If you are using a bookmap, you need to specify where the index list should be presented (for instance in the *backmatter* of the book. Technically, it is possible to also add it to the frontmatter, but this is unusual). This is done using an `<indexlist>` element in the `<booklists>` element (inside the `<backmatter>`):

```
<bookmap>
    ...
    <chapter href="tasks/troubleshooting.dita">
 ...
    </chapter>
    <backmatter>
        <booklists>
            <indexlist/>
        </booklists>
    </backmatter>
</bookmap>
```

For plain maps, the index list is automatically added at the end of the publication, with no need to modify the map.

Index - XML Fragment

In the merged map file *(on page 982)*, the structure that holds the index tree is the `<opentopic-index:index.groups>` element.

```
<map class="- map/map " >
    <oxy:front-page>
        ...
    </oxy:front-page>
    <opentopic:map xmlns:opentopic="http://www.idiominc.com/opentopic">
        ...
    </opentopic:map>
    <topic class="- topic/topic ">
        <title class="- topic/title ">Request Support</title>
       ...
    </topic>
    <opentopic-index:index.groups id="d16e5548">
      ...
    </opentopic-index:index.groups>
</map>
```

Each of the groups contain:

- A label, the starting letter ("T" in the following example).
- A tree of `<opentopic-index:index.entry>` elements.

```
<opentopic-index:index.group>
 <opentopic-index:label>T</opentopic-index:label>
  <opentopic-index:index.entry value="table of contents">
    <opentopic-index:formatted-value>table of contents</opentopic-index:formatted-value>
    <opentopic-index:refID value="table of contents:">
        <oxy:index-link xmlns:oxy="http://www.oxygenxml.com/extensions/author"
               href="#d16e3988"> [d16e3988]
        </oxy:index-link>
    </opentopic-index:refID>
    <opentopic-index:index.entry value="change header">
        <opentopic-index:formatted-value>change header</opentopic-index:formatted-value>
        <opentopic-index:refID value="table of contents:change header:">
           <oxy:index-link xmlns:oxy="http://www.oxygenxml.com/extensions/author"
                href="#d16e4176">
                [d16e4176] </oxy:index-link>
        </opentopic-index:refID>
    </opentopic-index:index.entry>
    <opentopic-index:index.entry value="style">
        <opentopic-index:formatted-value>style</opentopic-index:formatted-value>
```

```
        <opentopic-index:refID value="table of contents:style:">
            <oxy:index-link xmlns:oxy="http://www.oxygenxml.com/extensions/author"
                href="#d16e4120">
                [d16e4120] </oxy:index-link>
        </opentopic-index:refID>
    </opentopic-index:index.entry>
  </opentopic-index:index.entry>
</opentopic-index:index.group>
```

Each of the entries contain:

- The formatted value (`<opentopic-index:formatted-value>`).
- A link to the publication content (`<opentopic-index:refID>`/`<oxy:index-link>`).
- Possibly other child entries.

For the **DITA Map PDF - based on HTML5 & CSS** transformation type, the merged map is further processed resulting in a collection of HTML5 `<div>` elements. These elements preserve the original DITA `@class` attribute values and add a new value derived from the DITA element name.

```
<div class="- map/map map" >
   <div class="front-page/front-page">
       ...
   </div>
   <div class="toc/toc toc">
       ...
   </div>
   <div class="- topic/topic topic">
       <div class="- topic/title title">Request Support</title>
      ...
   </div>
   <div class=" index/groups groups">
     ...
   </div>
</map>
```

The index group content becomes:

```
<div class=" index/group group">
  <div class=" index/label label">T</div>
  <div class=" index/entry entry">
    <div class=" index/formatted-value formatted-value">table of contents</div>
    <div class=" index/refid refid">
        <div class=" index/link link"
               href="#d16e3988"> [d16e3988]
        </div>
```

```
    </div>
    <div class=" index/entry entry">
        <div class=" index/formatted-value formatted-value">change header</div>
         <div class=" index/refid refid">
             <div class=" index/link link"
                href="#d16e4176"> [d16e4176] </div>
        </div>
    </div>
    <div class=" index/entry entry">
        <div class=" index/formatted-value formatted-value">style</div>
        <div class=" index/refid refid">
            <div class=" index/link link"
                href="#d16e4120"> [d16e4120] </div>
        </div>
    </div>
  </div>
</div>
```

Index - Built-in CSS

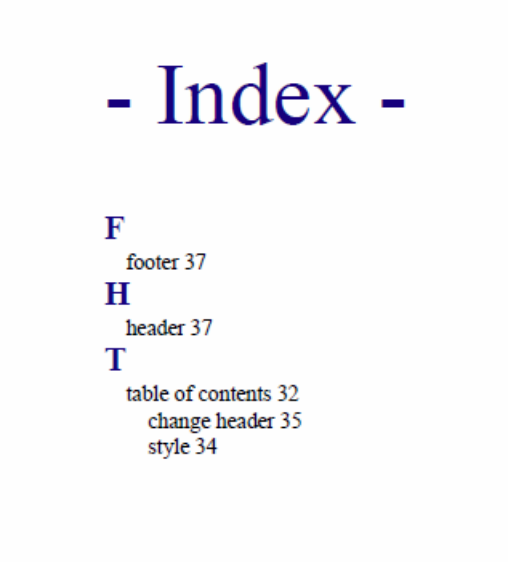
All index styling is found in: `[PLUGIN_DIR]css/print/p-index.css`.

How to Style the Index Page Title and the Grouping Letters

In your customization CSS *(on page 981)*, add the following CSS rules:

```
*[class ~= "index/groups"] *[class ~= "index/group"] *[class ~= "index/label"] {
    font-size:1.5em;
    color:navy;
}
*[class ~= "index/groups"]:before {
    content: "- Index - ";
    color:navy;
    font-size: 4em;
}
```

The result is:

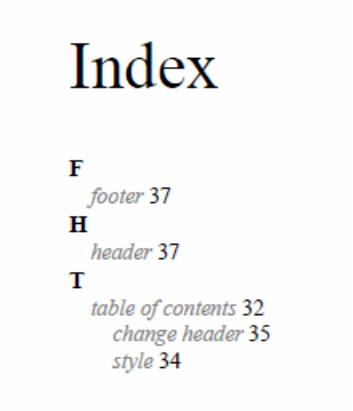


How to Style the Index Terms Labels

In your customization CSS *(on page 981)*, add the following CSS rule:

```
*[class ~= 'index/groups'] *[class ~= 'index/formatted-value'] {
    font-style:oblique;
    color:gray;
}
```

The result is:



How to Add Filling Dots Between the Index Labels and the Page Numbers

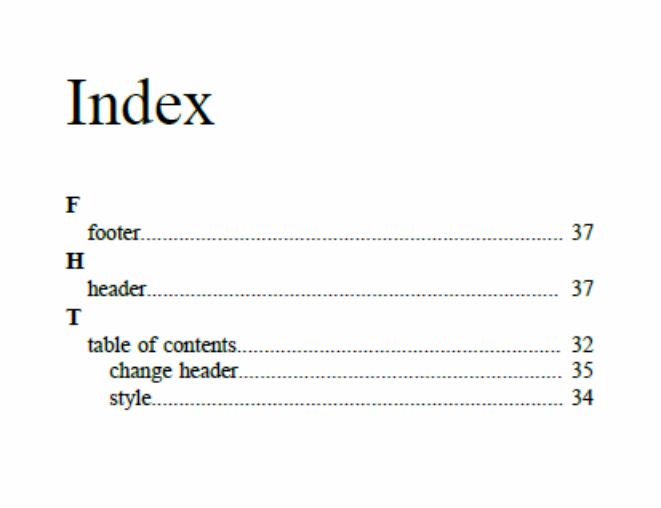
Suppose you want the leader CSS content to generate a row of dots. It is necessary that the parent entry has the text justified.

In your customization CSS *(on page 981)*, add the following CSS rule:

```
*[class~="index/formatted-value"],
*[class~="index/refid"] {
```

```
    display:inline;
}
/* Hide the sequences of links that actually do not contain links. */
*[class~="index/group"] *[class ~= "index/entry"] > *[class~="index/refid"]{
 display:none;
}
*[class~="index/group"] *[class ~= "index/entry"] >
 *[class~="index/refid"]:has(*[class~="index/link"]){
 display:inline;
}
*[class~="index/group"] *[class~="index/entry"] {
    text-align:justify;
}
*[class~="index/group"] *[class ~= "index/entry"] > *[class~="index/refid"]:before{
    content:leader('.');
}
```

The output now contains the dots:



How to Change the Index Page Number Format and Reset its Value

The page number is reset at the beginning of the index page by the built-in CSS rule:

```
*[class ~= "index/groups"] {
        counter-reset: page 1;
}
```

If you want to start the page counter from a different initial number, just change the value of this counter. For example, to continue the normal page counting, use:

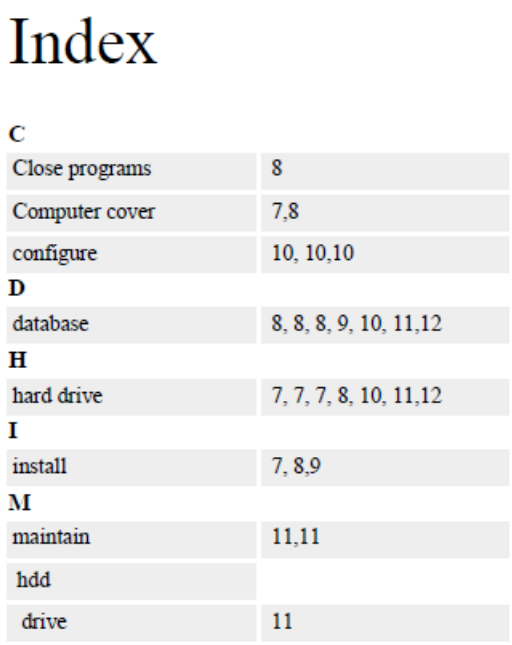
```
*[class ~= "index/groups"] {
        counter-reset: none;
}
```

If you need to style the page number differently (for example, using decimals), add the following CSS rule in your customization CSS *(on page 981)*:

```
@page index {
        @bottom-center      {          content: counter(page, decimal) }
}
```

How to Impose a Table-like Index Layout

In case you need to place the index labels and links on the same line but with some extra alignment constraints, you can use inline blocks to give the index a table-like appearance:



Index

C	
Close programs	8
Computer cover	7,8
configure	10, 10,10
D	
database	8, 8, 8, 9, 10, 11,12
H	
hard drive	7, 7, 7, 8, 10, 11,12
I	
install	7, 8,9
M	
maintain	11,11
hdd	
drive	11

You need to place the elements that have the following class on the same line:

index/formatted-value

This is the text of the index term.

index/refid

This element contains a list of links.

A fixed width is used for the formatted value and the links container (almost half of the available width). To achieve the index hierarchical layout, set progressive padding to the formatted value text.

In your customization CSS *(on page 981)*, add the following CSS rule:

```
*[class~="index/formatted-value"],
*[class~="index/refid"]{
   display:inline-block;
}
*[class~="index/formatted-value"]{
   width:45%;
}
```

```
*[class~="index/refid"] {
   width:45%;
}
/* Hide the sequences of links that actually do not contain links. */
*[class ~= "index/groups"] *[class ~= "index/entry"] > *[class~="index/refid"]{
 display:none;
}
*[class ~= "index/groups"] *[class ~= "index/entry"] >
 *[class~="index/refid"]:has(*[class~="index/link"]){
 display:inline-block;
}
/* Move the nesting of indexterms from margin to padding */
*[class ~= "index/groups"] *[class ~= "index/entry"] {
    margin-left: 0;
}
*[class ~= "index/groups"]
 *[class ~= "index/entry"]
 *[class~="index/formatted-value"]{
    padding-left: 0.2em;
}
*[class ~= "index/groups"]
 *[class ~= "index/entry"]
 *[class ~= "index/entry"]
 *[class~="index/formatted-value"]{
    padding-left: 0.4em;
}
*[class ~= "index/groups"]
 *[class ~= "index/entry"]
 *[class ~= "index/entry"]
 *[class ~= "index/entry"]
 *[class~="index/formatted-value"]{
    padding-left: 0.6em;
}
/* Some styling */
*[class~="index/formatted-value"],
*[class~="index/refid"]{
    padding:0.2em;
    background-color:#EEEEEE;
}
```

To avoid bleeding of the index term label, you may need to mark it as being hyphenated:

```
*[class~="index/formatted-value"] {
 hyphens:auto;
```

```
}
```

To activate hyphenation, see: How to Enable Hyphenation for Entire Map *(on page 1067)*.

Footnotes

Footnotes are pieces of information placed at the bottom of the page. A footnote has a number that is left in the content (the footnote call) and the same value placed before the footnote text (the footnote marker).

Footnotes - Built-in CSS

Footnote properties are defined in `[PLUGIN_DIR]css/print/p-foot-notes.css`.

How to Change Style of the Footnote Markers and Footnote Calls

To bold the footnotes numbers and use some colors, add the following rules to your customization CSS *(on page 981)*:

```
*[class ~= "topic/fn"]:footnote-call {
 font-weight: bold;
       color:red;
}
*[class ~= "topic/fn"]:footnote-marker {
 font-weight: bold;
       color:red;
}
```

Related Information:

https://www.oxygenxml.com/doc/ug-chemistry/topics/ch_footnotes.html

How to Add a Separator Above the Footnotes

The `@footnote` part of a `@page` declaration controls the style of the separator between the page content and the footnotes. For the content, you should set a `leader`. The leader uses a letter or a line style to fill the entire width of the page.

```
@page {
 margin:0.5in;
       ....
 @footnote {
  content: leader(solid);
  color:silver;
 }
}
```

To create a dotted line, you can use the dot character: `leader('.')`. Other commonly used characters are: "-" (dash) and "_" (underscore).

How to Reset the Footnotes Counter

There are cases where you want to reset the footnote counter.

For example, if you need to reset it at the beginning of each chapter, add the following rules to your customization CSS *(on page 981)*:

```
*[class ~= "bookmap/chapter"],
*[class ~= "topic/topic"][is-chapter] {
 counter-reset: footnote 1;
}
```

Or you can mark any element with an `@outputclass` value, match that value, and reset the counter at any point in your counter:

```
<p outputclass="reset-footnotes"/>
```

```
*[outputclass ~= "reset-footnotes"] {
 counter-reset: footnote 1;
}
```

Page Breaks

The page breaks can be controlled in multiple ways:

1. By creating an `@page` and assigning it to an element will create a page break between this element and the sibling elements that have a different page.
2. Using the CSS properties: `page-break-before`, `page-break-after`, or `page-break-avoid`.
3. In your DITA topic, set the `@outputclass` attribute on the topic root (or any element) to contain one of the `page-break-before`, `page-break-after`, or `page-break-avoid` values. If you want to control the page breaking from the DITA map, use the `@outputclass` attribute on the `<topicref>`, with any of the values mentioned above.

Related Information:

Double Side Pagination *(on page 1047)*

Oxygen PDF Chemistry: Controlling Page Breaks

Page Breaks - Built-in CSS

Page break properties are used in: `[PLUGIN_DIR]css/print/p-page-breaks.css`.

How to Avoid Page Breaks in Lists and Tables

To avoid splitting elements over two pages, you can use the `page-break-inside` CSS property. For example, if you want to impose this on tables and lists, then add the following rules to your customization CSS *(on page 981)*:

```
*[class ~= "topic/table"] {
    page-break-inside:avoid;
}
*[class ~= "topic/ol"] {
    page-break-inside:avoid;
}
*[class ~= "topic/ul"] {
    page-break-inside:avoid;
}
```

Note: Since the task steps are inherited from `topic/ol`, they will also not be split over two separate pages. However, if you want to allow this, add the following CSS rule:

```
*[class ~= "task/steps"] {
    page-break-inside:auto;
}
```

Note: Another way to do this is to mark the element with an `@outputclass` set to `page-break-avoid`.

How to Force a Page Break Before or After a Topic or Another Element

If you want to force a page break **before all** the second-level topics (for example, sections in chapters that are usually kept flowing one after another without page breaks), add the following in your customization CSS *(on page 981)*:

```
*[class ~= "map/map"] > *[class ~= "topic/topic"] > *[class ~= "topic/topic"] {
 page-break-before:always;
}
```

If you want to force a page break **for a specific topic**, mark the topic (or any other element you need to control page breaking for) with an `@outputclass` attribute set to one of these values:

page-break-before

Use this for a page break before the marked element.

page-break-after

Use this for a page break after the marked element.

page-break-avoid

Use this to avoid page breaks inside the marked element.

For example, to force a page break before a certain topic, use:

```
<topic outputclass="page-break-before" ... >
```

Note: You can set the output class on the `<topicref>` element from the DITA map instead of the `<topic>` element. In this way you can reuse the topic in another context where the page breaking is not necessary.

You can also control page breaking for lists, paragraphs, or any other block type elements. The following example avoids page breaks inside an ordered list:

```
<ol outputclass="page-break-avoid" ... >
```

How to Add a Blank Page After a Topic

If you want to add a new blank page after a topic, add the following rules to your customization CSS *(on page 981)*.

Style the separating blank page:

```
@page topic-separating-page{
   @top-left {
     content: "";
   }
   @top-right {
     content: "";
   }
   @top-center {
      content: "This page is blank";
   }
}
```

Associate this page to the `:after` pseudo-element of the topic:

```
*[class~="topic/topic"][outputclass~="add-separator-page"]:after {
 content: " ";
 display: block;
 page: topic-separating-page;
}
```

In the XML content, on the `<topic>` element, set the `@outputclass` to the `add-separator-page` value.

```
<topic outputclass="add-separator-page"> ... </topic>
```

The `:after` pseudo-element will be created next to the topic content and will be placed on the `topic-separating-page`.

Use the page margin box selectors to override the default content from the headers/footers.

Note: You can set the output class on the `<topicref>` element from the DITA map instead of the `<topic>` element. This allows you to reuse the topic in another context where the page breaking is not necessary.

How to Enforce a Number of Lines from Paragraphs that Continue in Next Page

In typography, an *orphan* is the first line of a paragraph that appears alone at the bottom of a page (the paragraph continues on a subsequent page), while a *widow* is the last line of a paragraph that appears alone at the top of a page. The default is 2 for each of them. You can control this number by adding the following to your customization CSS *(on page 981)*:

```
:root {
    widows:4;
    orphans:4;
}
```

How to Avoid Page Breaks Between Top-Level Topics (Chapters)

If you plan to publish a simple map with just one level of topics (such as a list of topics), then the automated page breaks between these topics might not be desired.

In this case, you can use the following CSS snippet to disable the page breaks between chapters:

```
*[class ~= "topic/topic"][is-chapter] {
  -oxy-page-group:auto;
}
```

Related Information:

Oxygen PDF Chemistry User Guide: Chapter Page Placement and Styling

Hyphenation

Hyphenation specifies how words should be hyphenated when text wraps across multiple lines.

The transformation plugin uses the capabilities of the **PDF Chemistry** processor to perform hyphenation.

Hyphenation Dictionaries

The Oxygen XML Developer Eclipse plugin provides built-in hyphenation patterns for the following languages:

Code	Language
da	Danish
de	German
de_CH	German (Switzerland)
en	English
en-GB	English (Great Britain)
es	Spanish

Code	Language
fr	French
it	Italian
nb	Norwegian Bokmål
nl	Dutch
ro	Romanian
ru	Russian
sv	Swedish
th	Thai
pt	Portuguese
da	Danish

The built-in hyphenation pattern license terms are listed in the XML files in the `[CHEMISTRY_INSTALL_DIR]/config/hyph` folder. Most of them comply with the *LaTex* distribution policy.

Installing New Hyphenation Dictionaries

Oxygen XML Developer Eclipse plugin uses the *TeX* hyphenation dictionaries converted to XML by the *OFFO* project: https://sourceforge.net/projects/offo/.

The `.xml` files allow you to access the licensing terms and you can use them as a starting point to create customized dictionaries (see How to Alter a Hyphenation Dictionary *(on page 1066)*).

The `.hyp` files are the compiled dictionaries that the Oxygen XML Developer Eclipse plugin actually uses.

One simple way to add more dictionaries:

1. Download and extract the `offo-hyphenation-compiled.zip` file. This file is a bundle of many dictionary files.
2. Copy the `fop-hyph.jar` file to the `[OXYGEN_INSTALL_DIR]/lib` directory.
3. If you just need a single dictionary, place the `.hyp` or `.xml` file in the `[OXYGEN_INSTALL_DIR]/config/hyph` directory (create that directory if it is missing).

How to Alter a Hyphenation Dictionary

You can copy the dictionaries you need to change in another directory, then use the `-hyph-dir` parameter to refer them inside your transformation.

Each file is named with the language code and has the following structure:

```
<hyphenation-info>
<hyphen-min before="2" after="3"/>
```

```
<exceptions>
o-mni-bus
...
</exceptions>
<patterns>
préémi3nent.
proémi3nent.
surémi3nent.
....
</patterns>
</hyphenation-info>
```

To change the behavior of the hyphenation, you can modify either the patterns or the exceptions sections:

exceptions

Contains the list of words that are not processed using the patterns, each on a single line. Each of the words should indicate the hyphenation points using the hyphen ("-") character. If a word does not contain this character, it will not be hyphenated.

For example, `o-mni-bus` will match the `omnibus` word and will indicate two possible hyphenation points.

Note: Compound words (like "e-mail") cannot be controlled by exception words.

patterns

Contains the list of patterns, each on a single line. A pattern is a word fragment, not a word. The numbers from the patterns indicate how desirable a hyphen is at that position.

For example, `tran3s2act` indicates that the possible hyphenation points are "tran-s-act" and the preferable point is the first one, having the higher score of "3".

How to Enable Hyphenation for Entire Map

To enable hyphenation for your entire map:

1. Make sure you set an `@xml:lang` attribute on the root of your map, or set the `default.language` parameter in the transformation.
2. In your customization CSS *(on page 981)*, add:

   ```
   :root {
       hyphens: auto;
   }
   ```
3. To except certain elements from being hyphenated, use `hyphens:none`. The following example excludes the `<keyword>` elements from being hyphenated:

   ```
   *[class ~= "topic/keyword"] {
       hyphens: none;
   ```

```
}
```

How to Enable/Disable Hyphenation for Tables

To enable hyphenation for your entire map:

1. Make sure you set an `@xml:lang` attribute on the root of your map or your table elements.
2. The built-in CSS already has the following rule defined that activates hyphenation:

```
*[class ~= "topic/table"] {
    hyphens: auto;
}
```

Note: To disable table hyphenation, add the following in your customization CSS *(on page 981)*:

```
*[class ~= "topic/table"] {
    hyphens: none;
}
```

3. To prevent certain elements from being hyphenated, use `hyphens:none`. The following example excludes the `<keyword>` elements from being hyphenated:

```
*[class ~= "topic/keyword"] {
    hyphens: none;
}
```

How to Disable Hyphenation for a Word

To disable hyphenation for a specific word, there are several possible approaches:

- If the word is a compound (like "T-shirt") and you want to keep it on the same line, you have two options:

  Manual Approach

  Wrap the word in an inline element with the `@outputclass` attribute set. In the CSS, change its style to `white-space:nowrap;`. For example:

```
.. <ph outputclass="no-hyphenation">T-shirt</ph>...
```

```
*[outputclass ~= "no-hyphenation"] {
 white-space: nowrap;
}
```

  Automatic Approach

  A better alternative to this is to write an XSLT extension that matches the text nodes and performs automatic markup (to see an example, go to How to Wrap Words in Markup *(on page 975)* in the XSLT Extensions for PDF Transformations *(on page 972)* section).
  Then match the `compound-word` class the same as in the previous example:

```
*[outputclass ~= "compound-word"] {
```

```
    white-space: nowrap;
}
```

- If the word is not a compound, you have two options:
  - Use one of the approaches listed above.
  - Alter the hyphenation dictionaries as explained in: How to Alter a Hyphenation Dictionary *(on page 1066)*.

Accessibility

By default, the PDF documents produced using this plugin are partially accessible in the sense that most of the paragraphs, tables, lists, headers, and footers are tagged automatically so a PDF reader can use this information to present the content.

Related Information:

Oxygen PDF Chemistry: Accessibility

Accessibility - Built-in CSS

Accessibility properties are defined in `[PLUGIN_DIR]css/print/p-accessibility.css`.

How to Create Fully Accessible Documents

To make your documents fully accessible (**PDF/UA1**), do the following:

1. In your customization CSS *(on page 981)*, specify fonts for all elements and for all page margin boxes. There should be no text that falls back to the basic provided fonts (Times, Courier, etc.). For instance, you can use:

   ```
   :root { font-family: Arial }
   @page {
       @top-left {font-family: Arial }
       ...
   }
   ```

2. Specify an `@alt` attribute with a description on any image, or better, use the `<alt>` DITA element.
3. Change the Chemistry invocation from the `build.xml` file, adding the argument `-pdf-ua`. There are two targets to change: `chemistry` and `chemistry.embedded`.

Fonts

Fonts are an important part of the publication. Your font selection should take into consideration both design and the targeted ranges of characters.

To use them in the customization CSS *(on page 981)*:

- You can place the font files in the same folder as your CSS and use a @font-face definition to reference them.
- You can use web fonts (for example, Google Fonts), and import the CSS snippet into your CSS.
- You can use system fonts.

All these techniques are explained in: **Oxygen PDF Chemistry User Manual: Fonts**.

How to Avoid Characters Being Rendered as

When the processor renders text with a font that does not include certain characters, those characters are replaced with the `#` symbol.

To prevent this, make sure you use the proper font.

As an example, suppose the right arrow character is used in a definition list like this:

```
<dlentry>
  <dt>&#8594;</dt>
  <dd><ph>This is the right arrow.</ph></dd>
</dlentry>
```

If the font does not include this character, the output will look something like this:

```
  #
    This is the right arrow.
```

To fix this, use a fallback font. For example, if you use *Times New Roman* for the entire publication, you could add *Symbol* as the fallback font. In your customization CSS *(on page 981)*, add:

```
*[class ~= "topic/dlentry"] {
  font-family: "Times New Roman", Symbol;
}
```

Tip: On Windows, one simple way to determine the font needed to display the text is to copy the text fragment that has rendering problems from the DITA source document and paste it into Microsoft WordPad or Word. It will automatically select a font capable of rendering the text. Simply click on the text to see the name of the font from the "Font" ribbon toolbar. Then you can use it as a fallback font in the CSS. Make sure there are no licensing restrictions on that particular font.

How to Set Fonts in Titles and Content

Suppose that in your customization CSS *(on page 981)*, you have defined your font (for example, *Roboto*) using a Google web font:

```
@import url('https://fonts.googleapis.com/css?family=Roboto');
```

You can force a font on all elements, then style the ones that need to be different. The advantage of this method is that you do not need to trace all elements that have a font family defined in the built-in CSS files, you just reset them all.

In your customization CSS *(on page 981)*, add an `!important` rule that associates a font to all the elements from the document:

```
*{
  font-family:"Roboto" !important;
}
```

Note: If you want to use the `:root` selector instead of the `*` sector, without the `!important` qualifier, the elements that have a predefined font specified in the built-in CSS will keep that font. If your content uses non-Latin glyphs, it is possible that the built-in fonts do not render them.

Next, identify the selectors for the elements that need to be styled with a different font than the one associated above. For information on how to do this, see: Debugging the CSS *(on page 982)*.

For example, if you want the titles or the pre-formatted text to have a different font from the rest, matched by the above `*` selector, you need to use more specific CSS selectors:

```
*[class~="front-page/front-page-title"],
*[class~="topic/title"] {
  font-family: Arial !important;
}
*[class~="pr-d/codeph"],
*[class~="topic/pre"] {
  font-family: monospace !important;
}
```

Related Information:

How to Change the Font of the Headers and Footers *(on page 994)*

How to Use Fonts for Asian Languages

For Asian languages, you must use a font or a sequence of fonts that cover the needed character ranges. If the characters are not found, the # symbol is used.

When you specify a sequence of fonts, if the glyphs are not found in the first font, the next font is selected until one that includes all the glyphs is found. A common font sequence for Asian languages is as follows:

```
  font-family: Calibri, SimSun, "Malgun Gothic", "Microsoft JhengHei";
```

To apply this font sequence, see: How to Set Fonts in Titles and Content *(on page 1070)*.

Some of the Asian fonts do not have italic, bold, or bold-italic variants. In this case, you may use the regular font file with multiple font face definitions to simulate (synthesize) the missing variants. You need to use the `-oxy-simulate-style:yes` CSS property in the font face definition as explained in: Using Simulated/Synthetic Styles in Oxygen Chemistry

How to Add Asian Fonts in Linux

For Asian languages on Linux distributions, you need to set a custom font inside your customization CSS *(on page 981)*. The following example uses the **Noto CJK** font.

Note: As a prerequisite, the `NotoSansCJKsc-hinted.zip` font was downloaded from Noto CJK and unzipped in a `resources` folder.

The font declaration and the font usage must be declared in the customization CSS *(on page 981)* file:

```
/* Font Declaration */
@font-face {
    font-family: "Noto Sans CJK";
    font-style: normal;
    font-weight: 100;
    src: url(resources/NotoSansCJKsc-Thin.otf);
}
@font-face {
    font-family: "Noto Sans CJK";
    font-style: normal;
    font-weight: 300;
    src: url(resources/NotoSansCJKsc-Light.otf);
}
@font-face {
    font-family: "Noto Sans CJK";
    font-style: normal;
    font-weight: 400;
    src: url(resources/NotoSansCJKsc-Regular.otf);
}
@font-face {
    font-family: "Noto Sans CJK";
    font-style: normal;
    font-weight: 700;
    src: url(resources/NotoSansCJKsc-Bold.otf);
}
/* Font usage */
* {
    font-family: "Noto Sans CJK", sans-serif;
}
.pre,
.filepath,
.codeph,
.codeblock {
    font-size: 10pt;
    font-family: "Noto Sans CJK", monospace;
```

```
}
```

Localization

The **DITA Map PDF - based on HTML5 & CSS** transformation type supports a large number of languages for customizing localization. In this type of transformation, the intermediary merged DITA map is transformed to HTML5 using the DITA-OT built-in support, so the customization of the internationalization (the labels for chapters, parts, figures, tables, notes, etc.) can be achieved by following the DITA-OT recommendations: Customizing Generated Text.

It is recommended that you do this customization on a DITA-OT distribution deployed outside of the **Oxygen** installation. Otherwise, you will lose the customization when upgrading **Oxygen**. You can contact the *Oxygen support team* to ask for the **Oxygen Publishing Engine** package.

Supported Languages (HTML5 and CSS)

There are over 20 supported languages. For the full list of supported languages (and their codes), see https://www.dita-ot.org/3.2/topics/globalization-languages.html.

How to Select a Language (HTML5 and CSS)

There are two ways to switch the labels to a specific language:

- Alter the DITA maps and/or topics to have the `@xml:lang` attribute set on the root element with one of the supported values (i.e. "fr", "de", "zh", "ru").
- If changing the files is not an option, you can alter the value of the `default.language` transformation parameter to the desired language code.

How to Add Support for a New Language

If the list of supported languages does not include the language you need, you can create a custom dictionary.

The DITA-OT documentation offers a good insight on how to do it: https://www.dita-ot.org/3.2/topics/plugin-addgeneratedtext.html

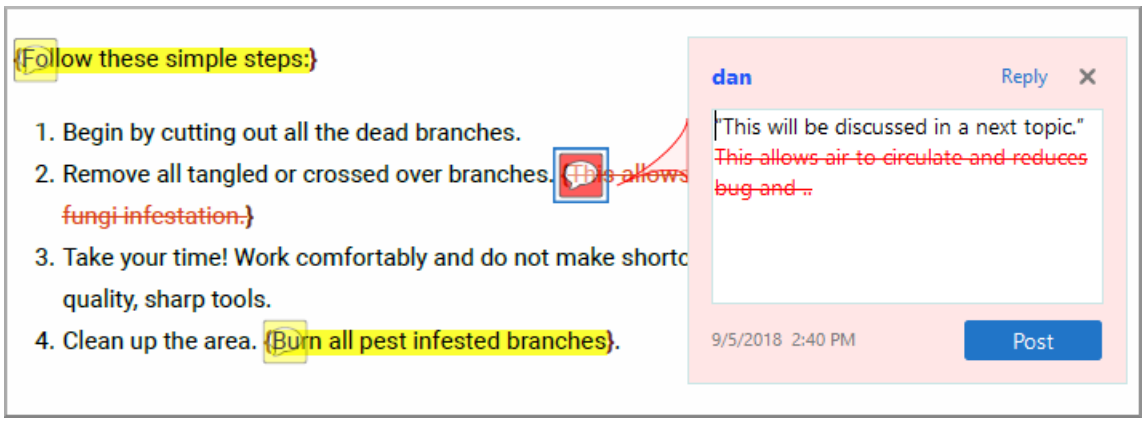
In addition to that, you need to alter CSS files. Some of the labels come from CSS files located in the `[PLUGIN_DIR]/css/print/i18n/` directory.

You should use one of the files as a template, copy its rules to your customization CSS *(on page 981)*, and then translate the labels. Make sure you also change the `:lang` selector to match your language code.

Comments, Highlights, and Tracked Changes

The comments and tracked changes can be made visible in the PDF output by setting the `show.changes.and.comments` transformation parameter to `yes`.

Figure 314. Chemistry Annotations in Acrobat Reader



By default, they are shown as PDF text annotations (sticky notes). These are graphical markers in the document content and are also listed in the **Comments** section when opening the output file in Acrobat Reader.

Note: Comments with the **Mark as Done** flag selected appear with a checkmark in the **Comments** section and with a **Completed** label (John Doe Completed).

To avoid rendering the elements as PDF annotations and show them as footnotes instead, you can use the `show.changes.and.comments.as.pdf.sticky.notes` transformation parameter set to `no`.

The comments and changes are included in the merged map file *(on page 982)* either as XML elements (`<oxy-insert>`, `<oxy-delete>`, `<oxy-comment>`, `<oxy-attributes>`) in the case of the XML merged map, or as HTML elements with similar classes (`oxy-insert`, `oxy-delete`, `oxy-comment`, `oxy-attributes`) in the case of the HTML merged map. Sub-elements contain meta-information about each change.

Tip: These elements are automatically recognized and transformed in PDF annotations when using Chemistry as PDF processor.

Note: The inserted text, deleted text, and deleted markup are included in the sticky notes, you can change this behavior by using the `show.changed.text.in.pdf.sticky.notes.content` parameter *(on page 960)*.

Related Information:

Transformation Parameters *(on page 956)*

Debugging the CSS *(on page 982)*

Comments and Tracked Changes - XML Fragment

This section contains information about how each type of tracked change is structured in the merged map XML file *(on page 982)*.

Insertions

For an insertion type of tracked change, the structure that defines the insertion details is inside the *range* (`<oxy-range-start>` to `<oxy-range-end>`), the inserted text is highlighted by an `<oxy-insert-hl>` element, and the details are stored in the `<oxy-insert>` element.

```
<oxy:oxy-range-start id="sc_1" hr_id="1"/>
   <oxy:oxy-insert href="#sc_1" hr_id="1">
       <oxy:oxy-author>dan</oxy:oxy-author>
       <oxy:oxy-content>insert</oxy:oxy-content>
       <oxy:oxy-date>2018/03/15</oxy:oxy-date>
       <oxy:oxy-hour>09:38:29</oxy:oxy-hour>
       <oxy:oxy-tz>+02:00</oxy:oxy-tz>
   </oxy:oxy-insert>
   <oxy:oxy-insert-hl>This is an insert!!</oxy:oxy-insert-hl>
<oxy:oxy-range-end hr_id="1"/>
```

Comments

Similar to insertions, comments are defined in a *range* (`<oxy-range-start>` to `<oxy-range-end>`), the comment details is in the `<oxy-comment>` element, and the highlighted content is wrapped in the `<oxy-comment-hl>` element.

```
<oxy:oxy-range-start id="sc_1" hr_id="1"/>

    <oxy:oxy-comment href="#sc_1" hr_id="1">
          <oxy:oxy-author>dan</oxy:oxy-author>
          <oxy:oxy-comment-text>This is a comment.</oxy:oxy-comment-text>
          <oxy:oxy-date>2018/03/15</oxy:oxy-date>
          <oxy:oxy-hour>09:56:59</oxy:oxy-hour>
          <oxy:oxy-tz>+02:00</oxy:oxy-tz>
    </oxy:oxy-comment>

    <oxy:oxy-comment-hl>Context</oxy:oxy-comment-hl>

<oxy:oxy-range-end hr_id="1"/>
```

Note: Comments that are marked as done have a `flag="done"` attribute:

```
<oxy:oxy-comment href="#sc_6" hr_id="6" flag="done">
```

Attribute changes

The attribute changes are more complex. The *range* is empty, and is directly above the affected element (the one that has modified attributes). The `<oxy-attributes>` element contains details about multiple attribute changes, each stored in the `<oxy-attributed-change>` element.

```
<element>
  <oxy:oxy-range-start id="sc_3" hr_id="3"/>
```

```
  <oxy:oxy-range-end hr_id="3"/>
  <oxy:oxy-attributes href="#sc_3" hr_id="3">
     <oxy:oxy-attribute-change type="inserted" name="platform">
          <oxy:oxy-author>dan</oxy:oxy-author>
          <oxy:oxy-current-value>windows</oxy:oxy-current-value>
          <oxy:oxy-date>2018/03/15</oxy:oxy-date>
          <oxy:oxy-hour>10:05:04</oxy:oxy-hour>
          <oxy:oxy-tz>+02:00</oxy:oxy-tz>
     </oxy:oxy-attribute-change>
    ....
     <oxy:oxy-attribute-change type="removed" name="audience">
        ....
     </oxy:oxy-attribute-change>
  </oxy:oxy-attributes>
...
</element>
```

Deletions

For a deletion, there are some elements that define the start and end of the deletion, and the highlighted text is wrapped in the `<oxy-delete-hl>` element.

```
  <oxy:oxy-range-start id="sc_2" hr_id="2"/>
  <oxy:oxy-delete-hl> This is a deleted text. </oxy:oxy-delete-hl>
  <oxy:oxy-range-end hr_id="2"/>
```

There is a structure that offers details about the deletion change, using the `<oxy-delete>` element. This is linked to the above deletion range by the same ID value:

```
  <oxy:oxy-delete href="#sc_2" hr_id="2">
      <oxy:oxy-author>dan</oxy:oxy-author>
      <oxy:oxy-content><image href="../img/ex.gif"></oxy:oxy-content>
      <oxy:oxy-date>2018/03/14</oxy:oxy-date>
      <oxy:oxy-hour>11:38:06</oxy:oxy-hour>
      <oxy:oxy-tz>+02:00</oxy:oxy-tz>
  </oxy:oxy-delete>
```

Colored Highlights

To show some text as highlighted with a background color:

```
  <oxy:oxy-color-hl color="rgba(140,255,140,50)">Some colored text.</oxy:oxy-color-hl>
```

Comments and Tracked Changes - HTML Fragment

This section contains information about how each type of tracked change is structured in the merged map HTML file *(on page 982)*.

Insertions

For an insertion type of tracked change, the structure that defines the insertion details is inside a *range* (`oxy-range-start` to `oxy-range-end`), the inserted text is highlighted by a `<span>` element with the class `oxy-insert-hl`, and the details are stored in a `<span>` element with the `oxy-insert` class.

```
<span class="oxy-range-start" id="sc_1" hr_id="1"/>
   <span class="oxy-insert" href="#sc_1" hr_id="1">
       <span class="oxy-author">dan</span>
       <span class="oxy-content">insert</span>
       <span class="oxy-date">2018/03/15</span>
       <span class="oxy-hour">09:38:29</span>
       <span class="oxy-tz">+02:00</span>
   </span>
   <span class="oxy-insert-hl">This is an insert!!</span>
<span class="oxy-range-end" hr_id="1"/>
```

Comments

Similar to insertions, comments are defined in a *range* (`oxy-range-start` to `oxy-range-end`), the comment details in an element with the class `oxy-comment`, and the highlighted content is wrapped in the `oxy-comment-hl` element.

```
<span class="oxy-range-start" id="sc_1" hr_id="1"/>

    <span class="oxy-comment" href="#sc_1" hr_id="1">
          <span class="oxy-author">dan</span>
          <span class="oxy-comment-text">This is a comment.</span>
          <span class="oxy-date">2018/03/15</span>
          <span class="oxy-hour">09:56:59</span>
          <span class="oxy-tz">+02:00</span>
    </span>

    <span class="oxy-comment-hl">The commented text.</span>

<span class="oxy-range-end" hr_id="1"/>
```

Note: Comments that are marked as done have a `flag="done"` attribute:

```
<span class="oxy-comment" href="#sc_6" hr_id="6" flag="done">
```

Attribute changes

The attribute changes are more complex. The range is empty, and is directly above the affected element (the one that has modified attributes). The element with the class `oxy-attributes` contains details about multiple attribute changes, each stored in an element with the class `oxy-attribute-change`.

```
<element>
```

```
  <span class="oxy-range-start" id="sc_3" hr_id="3"/>
  <span class="oxy-range-end" hr_id="3"/>
  <span class="oxy-attributes" href="#sc_3" hr_id="3">
     <span class="oxy-attribute-change" type="inserted" name="platform">
          <span class="oxy-author">dan</span>
          <span class="oxy-current-value">windows</span>
          <span class="oxy-date">2018/03/15</span>
          <span class="oxy-hour">10:05:04</span>
          <span class="oxy-tz">+02:00</span>
     </span>
    ....
     <span class="oxy-attribute-change" type="removed" name="audience">
        ....
     </span>
  </span>
...
</element>
```

Deletions

For a deletion, there are some elements that define the start and end of the deletion, and the highlighted text is wrapped in an element with the class `oxy-delete-hl`.

```
   <span class="oxy-range-start" id="sc_2" hr_id="2"/>
   <span class="oxy-delete-hl"> This is a deleted text. </span>
   <span class="oxy-range-end" hr_id="2"/>
```

There is a structure that offers details about the deletion change, using the element with the class `oxy-delete`. This is linked to the above deletion range by the same ID value:

```
   <span class="oxy-delete" href="#sc_2" hr_id="2">
       <span class="oxy-author">dan</span>
       <span class="oxy-content"><image href="../img/ex.gif"></span>
       <span class="oxy-date">2018/03/14</span>
       <span class="oxy-hour">11:38:06</span>
       <span class="oxy-tz">+02:00</span>
   </span>
```

Colored Highlights

To show some text as highlighted with a background color:

```
  <span class="oxy-color-hl" color="rgba(140,255,140,50)">Some colored text.</span>
```

Comments and Tracked Changes - Built-in CSS

The built-in CSS that controls the way tracked changes and comments are displayed is found in: `[PLUGIN_DIR]css/print/p-side-notes.css`.

How to Style Changed or Commented Text

To style the highlighted text from the document content, use the `<oxy-comment-hl>` element (or `<oxy-delete-hl>`, `<oxy-insert-hl>`, respectively, by local name or class name):

```
oxy-comment-hl,
.oxy-comment-hl {
  color:magenta;
}
```

Note: The class name selector is useful when using the **DITA Map PDF - based on HTML5 & CSS** or **DITA PDF - based on HTML5 & CSS** transformation scenarios.

If you want to change the small labels that define the affected start and end ranges (they are a number formatted like: "[n] .. [/n]" where n is the number of the change), you can use the following selectors:

```
oxy-range-start:before,
.oxy-range-start:before {
  display:inline;
  content:'[' attr(hr_id) ']';
  color:red;
}
oxy-range-end:before,
.oxy-range-end:before {
  display:inline;
  content:'[/' attr(hr_id) ']';
  color:red;
}
```

Use `display:none` if you want to hide them.

If you want to change the background color of the comment footnote (assuming you set the `show.changes.and.comments.as.pdf.sticky.notes` transformation to `no`), add the following snippet in your customization CSS *(on page 981)*:

```
oxy-comment,
.oxy-comment {
  background-color:inherit;
  border: 2pt solid yellow;
}
```

Similarly, you can style the other footnotes for `<oxy-attributes>`, `<oxy-delete>`, and `<oxy-insert>` elements.

How to Hide Tracked Change Footnotes

This topic is relevant if you have set the `show.changes.and.comments.as.pdf.sticky.notes` transformation parameter to `no`, and therefore the changes are shown as footnotes instead of PDF annotations.

In some cases, when your document contains a lot of tracked changes, you may need to hide the footnotes associated with the insertions, deletions, or attribute changes. Using the following example, the deletions and insertions are hidden, but the comments remain visible. In your customization CSS *(on page 981)*, add:

```
oxy-attributes,
oxy-delete,
oxy-insert{
  float:none;
  display:none;
}
```

Draft Watermarks

A *watermark* is an image displayed as the background of a printed document and it is faded enough to keep the publication text readable. *Draft watermarks* are used to indicate that a document is under construction or has not yet been approved.

How to Add a Draft Watermark on All Pages

To add a draft watermark to all of your publication pages, you can use the following page selector in your customization CSS *(on page 981)*:

```
@page {
    background-image: url("draft.svg");
    background-position:center;
    background-repeat:no-repeat;
    background-color:inherit;
}
```

If you have already set a background image for other pages (for example, the `front-page` or `table-of-contents`), the above selector won't change them, as they are more specific.

The best practice is to use a different `draft.css` CSS file that imports the customization CSS where the rest of the style changes reside. If you need to publish the content as a draft, use the `draft.css` in your transformation scenario, otherwise directly reference the customization CSS *(on page 981)*.

Related Information:

Images and Figures *(on page 1088)*

How to Add a Draft Watermark Depending on Metadata

Suppose you want to apply a *Draft watermark* until your DITA bookmap is approved and the map is approved when an `<approved>` element has been added to the metadata section (for example, in the **bookmeta/bookchangehistory** element).

```
<bookmeta>
   <author>John</author>
```

```
   <critdates>
      <created date="1/1/2015"/>
      <revised modified="3/4/2016"/>
      <revised modified="3/5/2016"/>
   </critdates>
   <bookchangehistory>
       <approved/>
   </bookchangehistory>
...
```

Use `oxy_xpath` every time you need to probe the value from an element other than the one matched by the CSS selector, and test the expression on the merged HTML file using the **Oxygen XPath Builder** view.

You can either use a page selector that imposes the draft watermark on the entire page surface (recommended):

```
@page {
    background-image: url(oxy_xpath('if  (//bookmeta/bookchangehistory/approved) then ""
else  "../../images/draft-watermark.png" '));
    background-position:center;
    background-repeat:no-repeat;
    background-color:inherit;
}
```

or use an element selector that restricts the watermark image only to the page area covered by that element:

```
:root, body{
  ... /* same as properties above */
}
```

The `body` selector is used here in case you want to apply the **DITA Map PDF - based on HTML5 & CSS** transformation type. You can use another element selector to target a specific part of your publication (for example, marking only the tables as drafts).

Related Information:

Metadata *(on page 1008)*

How to Debug XPath Expressions *(on page 986)*

Flagging Content

In DITA, you can mark certain content to flag it or draw attention to it. This is done by defining a flag in a DITAVAL file.

You can attach the DITAVAL file to the DITA map using the `<ditavalref>` element in the map, or by specifying it in the `args.filter` transformation parameter.

In the following example, all the elements that have the attribute `@product` set to `YourProd` is flagged to have a purple background:

```
<val>
...
   <prop action="flag" att="product" val="YourProd" backcolor="purple"/>
...
</val>
```

Related Information:

- Change Bars
- DITAVAL Elements

How to Flag Content Using Change Bars

As an example, to add a *change bar* (revision mark) for particular content, you can use the following in the DITAVAL file:

```
<val>
  <revprop action="flag"
    changebar="color:blue;style:solid;width:2pt;offset:1.25mm;placement:start" val="new"/>
</val>
```

This would result in any content that is marked with `@rev="new"` having a blue change bar.

How to Flag Content Using Images

You can mark the elements that match a specific profiling condition using images (one for the start, one for the end). The image references are relative to the DITAVAL file.

```
<val>
   <prop action="flag"
          att="product"  val="MyProd"
          backcolor="blue"
          color="yellow" >
      <startflag imageref="startflag.jpg">
        <alt-text>This is the start of my product info</alt-text>
      </startflag>
      <endflag imageref="endflag.jpg">
        <alt-text>This is the end of my product info</alt-text>
      </endflag>
   </prop>
</val>
```

Styling the Content

If you need to change the styles of the elements from the topic contents, you should create a customization CSS *(on page 981)* and then add CSS rules. To create the CSS rules, you can use the development tools described in Debugging the CSS *(on page 982)*.

Reusing the Styling for WebHelp and PDF Output

If you are using the `pdf-css-html5` transformation type, then the generated HTML5 document that is later converted to PDF is very similar to the generated HTML5 pages from the WebHelp Responsive output.

This is an output example from the WebHelp transformation:

```
<h1 class="title topictitle1" id="ariaid-title2">Care and Preparation</h1>
<div class="body">
    <p class="shortdesc">When caring ...</p>
    <p class="p">When caring for your flower garden you want ... </p>
```

And the same example from the PDF transformation (note the additional emphasized class values):

```
<h1 class="- topic/title title topictitle1" id="ariaid-title2">Care andPreparation</h1>
<div class="- topic/body body">
   <p class="- topic/shortdesc shortdesc">When caring ... </p>
   <p class="- topic/p p">When caring for your flower garden you want ...  </p>
```

It makes sense to reuse the same CSS rules you developed for one transformation type to the other. The main rule is to use the short class names instead of the long ones. For example, to style the short descriptions with italic font, use:

```
.shortdesc {
  font-style: italic;
}
```

The rule of thumb is that if you have a CSS rule that successfully styles an element in WebHelp, it should apply without any modification in the PDF output.

Titles

Titles can be styled by matching the `topic/title` class attribute.

```
*[class~="topic/title"] {
  color:navy;
}
```

How to Avoid Wrapping the Title Text Without Indentation

By default, the chapter/section number is on the same line as the title text. If the title is too long, the text wraps to the next line without any indentation.

```
4.5.5 This is a long title
```

```
text that wraps.
```

If you want the wrapped text to include indentation, you need to set the value of the `args.css.param.title.layout` transformation parameter to **table**. This results in the chapter/section number being placed in one cell while the rest of the title content is in another cell with wrapped text and it is displayed with an indent:

```
4.5.5 This is a long title
      text that wraps.
```

Equations

This processor supports MathML equations.

How to Change the Font of MathML Equations

Suppose that you need to change the font of MathML equations from the documentation, and also add some padding. The MathML fragments are wrapped in elements that have the class `equation-d/equation-block` or `equation-d/equation-inline`, so you can match them with:

```
*[class ~="equation-d/equation-block"],
*[class ~="equation-d/equation-inline"]{
 font-family:"courier new";
 font-size:1.5em;
 padding:1em;
}
```

Note: Some of the fonts may not be supported. In that case, a default serif font is used.

Lists

You can style the list item markers or other properties.

Note: The lists are treated differently than ordinary block elements in the sense that their margins are not collapsed with the margins of the neighboring blocks or lists. This is also visible for nested lists.

How to Style the List Markers

For the media print, the default CSS rules for the list items use the `:marker` pseudo-class.

```
@media print {
    *[class~="topic/li"]::marker {
      text-align: center;
      font-weight:bold;
    }
    *[class~="topic/li"] {
      margin-left: 0.25in;
    }
```

```
}
```

To eliminate the bold font weight for the items form ordered lists then add the following rules to your customization CSS *(on page 981)*:

```
*[class~="topic/ol"] > *[class~="topic/li"]::marker {
      font-weight:normal;
}
```

For the unordered lists:

```
*[class~="topic/ul"] > *[class~="topic/li"]::marker {
      font-weight:normal;
}
```

How to Change the Numbering System of Ordered Lists

It is possible to change all lists to have a different numbering system and there are several methods that can be used to achieve this.

Use the `list-style-type` CSS Property.

The **Chemistry** engine supports the following types: `decimal`, `decimal-leading-zero`, `lower-roman`, `upper-roman`, `lower-latin`, `upper-latin`, `lower-alpha`, `upper-alpha`.

```
*[class ~= "topic/ol"] {
  list-style-type: lower-roman;
}
```

Change the Content of the `:marker` CSS Pseudo-Element.

The following example emulates the Cyrillic numbering for the list items for an ordered list that has the `@outputclass` attribute set to `cyrillic`:

Important: This example will work only for lists up to 28 items. You will have to extend it for longer lists!

```
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class ~= "topic/li"]:marker {
  width:3em;
}
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(1):marker{ content:"а" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(2):marker{ content:"б" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(3):marker{ content:"в" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(4):marker{ content:"г" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(5):marker{ content:"д" }
```

```
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(6):marker{ content:"е" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(7):marker{ content:"ж" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(8):marker{ content:"з" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(9):marker{ content:"и" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(10):marker{ content:"к" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(11):marker{ content:"л" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(12):marker{ content:"м" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(13):marker{ content:"н" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(14):marker{ content:"о" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(15):marker{ content:"п" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(16):marker{ content:"р" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(17):marker{ content:"с" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(18):marker{ content:"т" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(19):marker{ content:"у" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(20):marker{ content:"ф" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(21):marker{ content:"х" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(22):marker{ content:"ц" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(23):marker{ content:"ч" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(24):marker{ content:"ш" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(25):marker{ content:"щ" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(26):marker{ content:"э" }
```

```
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(27):marker{ content:"Ю" }
*[class ~= "topic/ol"][outputclass ~= "cyrillic"] > *[class
 ~= "topic/li"]:nth-of-type(28):marker{ content:"Я" }
```

Related Information:

Oxygen PDF Chemistry User Guide: Lists

Links

Links allow the users to navigate through the documentation.

How to Remove 'on page NNN' Link Label

For printed material, it is usually desirable for the links to display a label after the text content (such as "on page 54"). This makes it easier the user to identify the target page. However, if the produced PDF is not printed and is intended only for electronic use, this label may create clutter and make the document harder to read. To eliminate this label, add the following in your customization CSS *(on page 981)*:

```
*[class ~= "topic/xref"][href]:after,
*[class ~= "topic/link"][href]:after {
    content: none !important;
}
```

Another use-case is to remove the labels only from links shown in tables cells, and leave the others as they are. For this, you could use a more specific selector:

```
*[class ~= "topic/entry"] *[class ~= "topic/xref"][href]:after{
    content: none !important;
}
```

How to Change Link Styles

Suppose you want the links to be bold and with an underline. In your customization CSS *(on page 981)*, add this snippet:

```
*[class ~= "topic/xref"][href]:after,
*[class ~= "topic/link"][href]:after {
    font-weight: bold;
    text-decoration: underline;
}
```

How to Hide Descriptions in Related Links Sections

The link descriptions that come from DITA relationship tables or related link elements within topics, are structured in the merged map *(on page 982)* like this:

```
<related-links class="- topic/related-links ">
   <linkpool class="- topic/linkpool ">
        <link class="- topic/link "
            ...
               role="friend" scope="local" type="topic">
                <linktext class="- topic/linktext ">Salvia</linktext>
                <desc class="- topic/desc ">The salvia plant</desc>
          </link>
    </linkpool>
    ...
</related-links>
```

If you need to hide these descriptions, add the following code in your customization CSS *(on page 981)*:

```
*[class ~= "topic/link"] > *[class ~= "topic/desc"] {
  display: none;
}
```

Images and Figures

Images are an important part of a publication.

Note: You can use raster image formats (such as PNG or JPEG), but it is best to use vector images (such as SVG or PDF). They scale very well and produce better results when printed. In addition, the text from these images is searchable and can be selected (if the glyphs have not been converted to shapes) in the PDF viewer.

Images - Built-in CSS

Image properties are defined in `[PLUGIN_DIR]css/print/p-figures-images.css`.

```
  *[class ~= "topic/image"] {
        prince-image-resolution: 120dpi;
        -ah-image-resolution: 120dpi;
        image-resolution: 120dpi;
        /* The US-letter page size minus page margins. See p-page-size.css for the current page
 size. */
        max-width: 6.5in;
    }
```

How to Fix Image Bleeding - Control Image Size

Sometimes the images may be too big for the page. The built-in CSS rules specify a maximum size for images, limiting to the width of the parent block. But if the parent block is itself too wide and bleeds out of page, you might consider specifying a length.

In your customization CSS *(on page 981)*, add the following snippet:

```
 *[class ~= "topic/image"] {
   ...
       /* The US-letter page size minus page margins. See p-page-size.css for the current page
size. */
       max-width: 6.5in;
   }
```

How to Change Image Resolution

This is a good technique to change the size of all **raster** images from your documentation. This will not work for vector images, such as PDF or SVG. If the images (PNG, for example) contain the image resolution in their metadata, this will be taken into account. For all other images, the default resolution is 120dpi.

Note: Instead of using `@width` and `@height`, it is recommended that you use images that fit the page dimensions. Then you could fine-tune the images in the output using the `image-resolution` property (see below). This will help to prevent image rendering issues (for example, a multiple format output or a replaced image in the source file).

If the default resolution is not good (suppose you need a higher pixel density of 300dpi), you can change it by adding the following in your customization CSS *(on page 981)*:

```
*[class ~= "topic/image"] {
        prince-image-resolution: 300dpi;
        -ah-image-resolution: 300dpi;
        image-resolution: 300dpi;
}
```

How to Place Big Images on Rotated Pages

Very wide images may bleed out of the page. One solution for this is to use landscape pages for these wide images.

In your customization CSS *(on page 981)*, add:

```
*[class~="topic/image"][outputclass='land'] {
    page: landscape-page;
}
```

Setting the attribute `@outputclass` = 'land' on the table element will force the table on a new landscape page.

Another solution is to set an `@outputclass` attribute on the image, then create a rule that matches it, and associate a landscape page for it.

How to Place a Text and Image Side by Side

If you need to align text and an image side by side, you can use the following technique:

1. Organize your text and image under a `<div>` element like this:

```
...
 <div outputclass="side-by-side">
      <p> This will be in the left side, the next figure in the right. </p>
      <fig>
        <image href="cactus.jpeg"/>
      </fig>
 </div>
...
```

Note: You can use the `@outputclass` attribute to mark the `<div>` elements that have this special layout.

2. In your customization CSS *(on page 981)*, add:

```
*[outputclass ~= "side-by-side"] > *[class ~= "topic/p"] {
  display:inline-block;
  width: 45%;
}
*[outputclass ~= "side-by-side"] > *[class ~= "topic/fig"] {
  display:inline-block;
  width: 45%;
}
```

The image should fill the entire width of the parent `<fig>` element:

```
*[outputclass ~= "side-by-side"] > *[class ~= "topic/fig"]  > *[class ~= "topic/image"] {
  width:100%;
}
```

By default, the bottom of the image is on the same line as the text baseline. If you want the text and the image to be aligned at the top, add these lines:

```
*[outputclass ~= "side-by-side"] > *[class ~= "topic/p"] {
  vertical-align:top;
}
*[outputclass ~= "side-by-side"] > *[class ~= "topic/fig"] {
  vertical-align:top;
  font-size:0pt;
}
```

Note: The `font-size:0pt` is needed to remove the font ascent and descent around the image rectangle.

How to Control the Image Size in Complex Static Content

It is common to have text and images mixed together in a `:before` or `:after` pseudo-element. For example, for notes you may have both artwork and text:

```
*[class ~= "topic/note"]::before {
    content: url('note.png') "Some text";
}
```

If you want to change the size of the image, you have two options:

- Use the `image-resolution` CSS property:

  ```
  *[class ~= "topic/note"] {
      image-resolution:300dpi;
  }
  ```

- Separate the image from the text and apply the width and height CSS properties only on the image, using the width and height properties. You could use multiple `:before` pseudo-elements for that, considering that the farthest content presented before the actual content of an element is matched by the `:before` with the highest number in the brackets:

```
*[class ~= "topic/note"]:before(2) {
    content: url('note.png') ;
    width:0.5in;
}
*[class ~= "topic/note"]:before(1) {
    content: "Some text";
}
```

How to Center Images

DITA defines a `@placement` attribute for the `<image>` elements. The implicit value is `inline`. Suppose that you need to center the images that have the placement set to `break` (for example, they are not on the same line with other content and the images from the `<fig>` element).

In your customization CSS *(on page 981)*, add:

```
*[class ~= "topic/fig"] {
  text-align:center;
}
/* Other images, with break placement. */
*[class ~= "topic/image"][placement='break']{
  display:block;
  text-align:center;
}
/*
 Scaled images are getting a computed width attribute, so we can use the auto margins.
```

```
 Auto margins function only if the block they apply to has a width.
*/
*[class ~= "topic/image"][placement='break'][width] {
  margin-left:auto;
  margin-right:auto;
  border: 2pt solid red;
}
```

How to Change/Reset the Figure Numbering

Note: This topic is applicable for the *DITA Map PDF - based on HTML5 & CSS* *DITA PDF - based on HTML5 & CSS* transformation types.

There are cases when you need to change the aspect of the figure counter that is shown before the figure titles. By default, the figure titles are formatted like this:

```
Figure NN. Lore Ipsum Title
```

NN is the number of the figure that starts being counted from the beginning of the publication.

One use-case is to have the NN counter be incremented only within one chapter (for example, the first chapter contains "Figure 1" and "Figure 2", and the second chapter starts over with "Figure 1" instead of incrementing to "Figure 3").

You should reset the figure counter on each topic marked as chapter, then hide the label from the figure `<figcaption>` (this is an HTML element generated by the XSL transformation), and create another label using a `:before` selector on the `<figcaption>`.

```
*[class ~= "topic/topic"][is-chapter] {
  counter-reset: figcount;
}
.fig--title-label{
  display:none;
}
*[class ~= "topic/fig"] > .figcap:before{
  /* Add more styling or change the content if needed */
  content: "Figure " counter(figcount) ". ";
  counter-increment: figcount;
}
```

How to Fix Missing Images

If your images are not accessible, you may receive an error message in the transformation console like this:

```
Image not found. URI:file:/path/to/my/image
```

This is usually because they are in a folder that is not in the folder subtree of the transformed map or topic.

To solve this, you can set the following transformation parameter: `fix.external.refs.com.oxygenxml=true`.

Tables

Tables are widely used in technical documentation. This section contains information about the CSS rules that are used to style them and how to fix some problems.

Tables - Built-in CSS

There is a combination of CSS files that address tables:

- `[PLUGIN_DIR]/css/core/-table-html-cals.css`
- `[PLUGIN_DIR]/css/print/p-tables.css`

How to Deal With Wide Tables - Page Rotation

Some of the tables can have a large number of columns. In this case, the table may bleed out of the page. One solution is to use landscape pages for these tables.

Setting the attribute `@orient` = 'land' on the table element will force the table on a new landscape page.

Another solution is to use automatic detection of wide tables (5 or more columns):

```
*[class~="topic/table"] > *[class~="topic/tgroup"][cols='5'],
*[class~="topic/table"] > *[class~="topic/tgroup"][cols='6'],
*[class~="topic/table"] > *[class~="topic/tgroup"][cols='7'],
*[class~="topic/table"] > *[class~="topic/tgroup"][cols='8'] {
    page: landscape-page;
    max-width: 100%;
    page-break-before: avoid;
}
```

Note: The `landscape-page` page layout is defined in the `[PLUGIN_DIR]css/print/p-pages-and-headers.css`.

If you want to rotate the entire topic that contains the big table, use:

```
*[class~="topic/table"] > *[class~="topic/tgroup"][cols='5'],
*[class~="topic/table"] > *[class~="topic/tgroup"][cols='6'],
*[class~="topic/table"] > *[class~="topic/tgroup"][cols='7'],
*[class~="topic/table"] > *[class~="topic/tgroup"][cols='8'] {
    max-width: 100%;
    table-layout:auto;
}
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *[class~="topic/table"] >
 *[class~="topic/tgroup"][cols='5']),
```

```
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *[class~="topic/table"] >
 *[class~="topic/tgroup"][cols='6']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *[class~="topic/table"] >
 *[class~="topic/tgroup"][cols='7']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *[class~="topic/table"] >
 *[class~="topic/tgroup"][cols='8']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *:not([class~="topic/topic"]) >
 *[class~="topic/table"] > *[class~="topic/tgroup"][cols='5']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *:not([class~="topic/topic"]) >
 *[class~="topic/table"] > *[class~="topic/tgroup"][cols='6']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *:not([class~="topic/topic"]) >
 *[class~="topic/table"] > *[class~="topic/tgroup"][cols='7']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *:not([class~="topic/topic"]) >
 *[class~="topic/table"] > *[class~="topic/tgroup"][cols='8']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *:not([class~="topic/topic"]) >
 *:not([class~="topic/topic"]) > *[class~="topic/table"] > *[class~="topic/tgroup"][cols='5']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *:not([class~="topic/topic"]) >
 *:not([class~="topic/topic"]) > *[class~="topic/table"] > *[class~="topic/tgroup"][cols='6']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *:not([class~="topic/topic"]) >
 *:not([class~="topic/topic"]) > *[class~="topic/table"] > *[class~="topic/tgroup"][cols='7']),
*[class~="topic/topic"]:has(*:not([class~="topic/topic"]) > *:not([class~="topic/topic"]) >
 *:not([class~="topic/topic"]) > *[class~="topic/table"] > *[class~="topic/tgroup"][cols='8']){
      page: landscape-page;
}
```

How to Fix Text Bleeding From Table Cells

Slim tables or tables that have many columns make the text from the cells be confined to a small horizontal space. Sometimes this causes long words to bleed outside the cell boundaries.

By default, the built-in CSS automatically activates the hyphenation for the text inside tables as long as your topics have the language specified.

In case the text is still bleeding outside the boundaries, you can also use the `overflow-wrap` property to force the word to break:

```
*[class ~= "topic/table"] {
  overflow-wrap: break-word;
}
```

Related Information:

Hyphenation *(on page 1065)*

How to Enable/Disable Hyphenation for Tables *(on page 1068)*

How to Avoid a Table Exceeding the Page Width

The DITA specification indicates that tables should have a fixed layout. This can be done in two different ways:

1. **Using proportional or relative measures** - It includes percent values and shares values (i.e. "3*" or "12*").
2. **Using fixed measures** - It includes all the values followed by units (i.e. *in*, *pt*, *px*, and others).

Important: Although the specification allows you to combine these values, it is **highly recommend** that you only use one method at a time. Combining both methods could lead to a table exceeding the page width and will make the content unreadable.

How to Enable the Automatic Table Layout

It is possible to automatically arrange the table layout directly from the customization CSS *(on page 981)* by simply adding this:

```
*[class~="topic/tgroup"] {
    table-layout:auto !important;
}
```

This will help you to obtain a more optimal arrangement of the cells inside your table.

If you want to control which table will use this layout, you can set the `@outputclass` attribute on the `<table>` element:

```
<table outputclass='auto_tbl'> ... </table>
```

Then, in the CSS, use a rule that matches the `@outputclass`:

```
*[class~="topic/table"][outputclass='auto_tbl'] > *[class~="topic/tgroup"] {
    table-layout:auto !important;
}
```

Important: Make sure the tables have no column width specified.

How to Rotate Content from a Table Cell

There are cases where you want to style the first column as a kind of table header, with vertical text.

There is an important thing to remember: you can rotate an element from a table cell, but not the cell itself. So, your DITA table cell should contain a `<div>` or a `<p>` element that will be rotated. The cell has to be marked somehow so that it can be matched from the CSS. One way is to set an `@outputclass` attribute on it, another will be to mark the table and then match the first entries from it.

```
<row>
    ...
    <entry outputclass="rotated">
        <p>Rotated</p>
```

```
      </entry>
      ...
 </row>
```

In your customization CSS *(on page 981)*, use the following rule that matches the child of the entry:

```
*[outputclass ~= "rotated"] > * {
  display: block;
  transform: rotate(-90deg) !important;
  text-align: left !important;
  padding:0;
  margin:0;

  white-space: nowrap;
  width:150px;
  height:80px;
}
```

The padding and margins are set to zero to clear any space that may come from other rules. The width is required - it will become the height of the cell. If this is not specified, the vertical rotated text will bleed out of the cell. The height is optional.

How to Add Horizontal Lines to a Choice Table

To add horizontal lines that separate the options within a `<choicetable>`, you can use borders set on each of the rows. The following CSS styles the top header and the first column with some background colors. In a choice table, the first column represents the choice labels.

```
*[class~="task/choptionhd"],
*[class~="task/choptionhd"],
*[class~="task/chdeschd"],
*[class~="task/choption"] {
 background-color: #EEEEEE;
 text-align: left;
}
*[class~="task/choicetable"] {
 border: 2pt solid #EEEEEE;
}
*[class~="task/choicetable"] *[class~="task/chrow"],
*[class~="task/choicetable"] *[class~="task/chhead"]{
 border-bottom: 2pt solid #EEEEEE;
}
*[class~="task/choicetable"] *[class~="topic/stentry"] {
 border-bottom: none;
 border-right: none;
```

```
}
```

Note: Using the frame attribute on the choice table will make these selectors apply partially. Please make sure you are designing your customization CSS taking into account all possible values for the frame attribute.

How to Remove the Table NN Label

For the **DITA Map PDF - based on HTML5 & CSS** transformation scenario, the label for a table's title is wrapped in a span element with the class: `table--title-label`.

```
<table ... >
...
<caption class="- topic/title title tablecap">
    <span class="table--title-label">Table
  <span class="table--title-label-number">1. </span></span>
    <span class="table--title">The title of the table</span>
</caption>
...
```

To hide it, set its display to none:

```
.table--title-label {
 display:none;
}
```

For the direct transformation, use:

```
*[class ~= "topic/table"] > *[class ~= "topic/title"]:before {
  content: none;
}
```

How to Center Tables

You can center the tables by using margins `auto`, while the table caption (title) can be centered using the `text-align` property:

```
*[class ~= "topic/table"] {
    margin-left:auto;
    margin-right:auto;
    width: 50%;
    border: 1pt solid blue;
}
*[class ~= "topic/table"] *[class ~= "topic/title"]{
    text-align:center;
}
```

Code Blocks

Code blocks are used to render section of programming code.

How to Enable Code Syntax Highlighting

This topic refers only to the **DITA Map PDF - based on HTML5 & CSS** transformation type.

You can use syntax highlighting to make it easier to read your code snippets by displaying each type of code in different colors and fonts. In the DITA topics, set the `@outputclass` attribute on the `<codeblock>` elements to one of these values:

- language-bourne
- language-c
- language-cpp
- language-csharp
- language-css
- language-ini
- language-java
- language-javascript
- language-json
- language-lua
- language-perl
- language-php
- language-python
- language-ruby
- language-sql
- language-xml
- language-xquery

For example, for a java snippet:

```
<codeblock outputclass="language-java">
   for (int i=0; i <100; i++) {
     // do something
   }
</codeblock>
```

The resulting HTML fragment in the merged HTML5 document is:

```
 <pre class="+ topic/pre pr-d/codeblock pre codeblock language-java"
    xml:space="preserve">
   <strong class="hl-keyword" style="color:#7f0055">for</strong>
       (<strong class="hl-keyword" style="color:#7f0055">int</strong>
         i=<span class="hl-number">0</span>; i
            <<span class="hl-number">100</span>; i++) {
     <em class="hl-comment" style="color:#006400">// do something</em>
```

```
   }
</pre>
```

And in the output, it will be rendered as:

```
for (int i=0; i <100; i++) {
  // do something
}
```

Changing the Colors for the Syntax Highlighting

As you can see in the above example, the HTML elements `<span>` and `<strong>` are used to color the content. Since they have a `@style` attribute set, the overriding properties need to be marked with `!important`.

Suppose you want to color the keywords in red and the comments in blue. To do so, add the following to your customization CSS *(on page 981)*:

```
.hl-keyword {
  color: red !important;
}
.hl-comment {
  color: blue !important;
}
```

How to Disable Line Wrapping in Code Blocks

By default, code blocks have the content wrapped to avoid the bleeding of long lines out of the page. To avoid wrapping, add the following in your customization CSS *(on page 981)*:

```
*[class~="pr-d/codeblock"] {
   white-space: pre;
}
```

For the **DITA Map PDF - based on HTML5 & CSS** transformation type, the best solution to distinguish between lines is to leave them wrapped, but color each line with a different background (zebra coloring). An example is provided here: XSLT Extensions for PDF Transformations *(on page 972)*.

How to Deal with Unwanted Returns in Code Blocks

There are cases where the source file contains long lines of code that need to continue onto the next line in the rendered PDF (to wrap visually).

When the user copies the block from the PDF reader, they get two separated lines. This means that the command fails when users copy it from the PDF to the command-line terminal (because it comes in as two commands).

For example, the command:

```
$gist = ls -l * | count -n | some more
```

May be rendered in the PDF on two lines:

```
$gist = ls -l * | count -n
| some more
```

And this is invalid when used in the terminal.

There is no CSS workaround for this, but to manually format the command line, add a line continuation character like this:

```
$gist = ls -l * | count -n \
| some more
```

Note: For Linux/Mac OSX, the continuation character is the backslash `\`. For Windows, this is the shift character `^`.

The command-line processor will now recognize that the first line is continuing on to the next one.

Notes

Notes contain an additional piece of information that calls attention to particular content. They may have various types (tip, caution, danger, restriction, important, warning).

For information on how to add and manage mixed content before the note icons and labels, see How to Control the Image Size in Complex Static Content *(on page 1090)*.

How to Change Note Icons

The recommended icon format is SVG.

To change the default icons for notes that do not have a `@type` attribute, add the following rule to your customization CSS *(on page 981)*:

```
div.note > span.note__title{
    background-image:url("../img/note.svg") !important;
}
```

For a note with a `@type` attribute set to *warning*, *caution*, or *trouble*, add the following corresponding CSS rule:

```
div.warning > span.note__title{
    background-image:url("../img/warning.svg") !important;
}
div.caution > span.note__title{
    background-image:url("../img/caution.svg") !important;
}
div.trouble > span.note__title{
    background-image:url("../img/troubleshooting.svg") !important;
}
```

Abbreviated Forms

When using the `<abbreviated-form>` element in your content, it is possible to style the subsequent occurrences differently than the first occurrence. To achieve this, add something similar to the following rule in your customization CSS *(on page 981)*:

```
a:has(dfn[class ~= "abbreviated-form"]) {
    color: oxy_xpath("let $cdf:= dfn return if (preceding::dfn[@keyref = $cdf/@keyref]) then
 'black' else 'red'");
    text-decoration: oxy_xpath("let $cdf:= dfn return if (preceding::dfn[@keyref = $cdf/@keyref])
 then 'none' else 'underline'");
}
```

This example would render the first occurrence with a red color and an underline, while the subsequent occurrences would be rendered with a black color and no underline.

Trademarks

Trademarks are used to specify legally registered words and they are often used in technical documentation. To specify a trademark, your DITA content could use a structure similar to this:

```
<tm tmtype="tm">My Product Name</tm>
```

Depending on the value of the `@tmtype` attribute, a different symbol is appended to the text: (®, ™ , or #).

The structure of the merged HTML document the CSS will apply to is:

```
<span class="- topic/tm tm" tmtype="tm">My Product Name<span
                          class="- topic/tmmark tmmark ">™</span></span>
```

How to Style the Trademark Element Text

To change the style of the entire trademark text, you can match the `topic/tm` class like this:

```
*[class ~= "topic/tm"] {
 font-weight:bold;
}
```

How to Style the Trademark Symbol

To change the aspect of the trademark symbol, you can use the `topic/tmmark` class. Usually, common fonts already render these symbols smaller and with superscript by default. The following example does it from the CSS:

```
*[class ~= "topic/tmmark"] {
 vertical-align: super;
 font-size: smaller;
}
```

Styling Through Custom Parameters

You can activate parts of your CSS by using custom transformation parameters that start with the `args.css.param.` prefix.

These parameters are recognized by the publishing pipeline and are forwarded as synthetic attributes on the root element of the merged map. The last part of the parameter name will become the attribute name, while the value of the parameter will become the attribute value. The namespace of these synthetic attributes is: `http://www.oxygenxml.com/extensions/publishing/dita/css/params`.

When using the **DITA Map PDF - based on HTML5 & CSS** or the **DITA PDF - based on HTML5 & CSS** transformations, the generated attribute will be in no namespace.

Notes:

- Make sure the name of your custom parameter does not conflict with an attribute name that may already exist on the root element.
- Use only Latin alphanumeric characters for parameter names.
- You can set multiple styling parameters at the same time.

How to Limit the Depth of the TOC Using a Parameter

In the following example, a custom parameter is used to switch from a full depth table of contents to a flat one that shows only the titles of the first-level topics (such as chapters, preface, notices).

The custom parameter is:

```
args.css.param.only-chapters-in-toc="yes"
```

The CSS that hides the *topicrefs* at level 2 or more:

```
:root[only-chapters-in-toc='yes'] *[class ~= "toc/toc"]
        > *[class ~= "map/topicref"]> *[class ~= "map/topicref"] {
  display:none;
}
```

The `:root[a|only-chapters-in-toc='yes']` selector makes the rule activate only when the attribute is set.

How to Change the Page Size Using a Parameter

In the following example, a custom parameter is used to modify the page size. The parameter is defined in the transformation scenario as:

```
args.css.param.page-size="A4"
```

Then in the CSS, the attribute value is extracted and used as follows:

```
@page {
    size: oxy_xpath('/*/@*[local-name()="page-size"]');
}
```

How to Change the Front Page Background Using a Parameter

Suppose you want to customize the front page of your publication depending on an "internal-use" marker. You should define the parameter in the transformation scenario as:

```
args.css.param.internal-use="yes"
```

Then in the CSS, the attribute value is extracted and used as follows:

```
:root[internal-use='yes'] *[class~='front-page/front-page'] {
    page: front-page-internal-use;
}
@page front-page-internal-use {
 background-color: yellow; /* or use an SVG as background-image */
}
```

Controlling the Publication Content

Using a plain DITA map, the transformation will produce a publication with a front page, a table of contents, chapters with content, and an index at the end. This is appropriate for most cases, but there are use cases where some adjustments are necessary. For example, if you want to do one of the following:

- Remove the TOC or index.
- Add a glossary.
- Change the position of the TOC or the index relative to the sibling topics.
- Add a preface, frontmatter, or backmatter with copyright notices, abstracts, list of tables, list of figures, etc.

All of these can be achieved using a DITA **bookmap**.

A bookmap has a more elaborate structure than a regular map. You should start by defining the title structure, with a main title and alternative title:

```
<!DOCTYPE bookmap PUBLIC "-//OASIS//DTD DITA BookMap//EN"  "bookmap.dtd">
<bookmap id="taskbook">
  <booktitle>
    <mainbooktitle>Publication Title</mainbooktitle>
    <booktitlealt>A very short description of the publication</booktitlealt>
  </booktitle>
```

Then you may define a frontmatter. For this, you can link the topics that need to appear before the main content. You can also define the location where the table of contents will be placed. In the example below, it appears between the `abstract.dita` and `foreword.dita` topics:

```
  <frontmatter>
    <topicref href="topics/abstract.dita"/>
    <booklists>
      <toc/>
```

```
  </booklists>
  <topicref href="topics/foreword.dita"/>
</frontmatter>
```

Note: To remove the TOC from the publication, just omit the `<toc>` element from the `<booklists>` element.

Next, the topics are grouped into chapters:

```
...
<chapter href="topics/installation.dita" />
...
```

At the end, you could define the structure of the backmatter. Just like for the frontmatter, you can include some topics and some generated content (such as the index). In the example below, the glossary is defined to come after the index, followed by a list of figures and list of tables. At the very end, there is a topic with some thank you notes.

```
<backmatter>
  <topicref href="topics/conclusion.dita"/>
  <booklists>
    <indexlist/>
    <glossarylist>
      <topicref href="topics/xp.dita" keys="xp" print="yes" />
      <topicref href="topics/anti_lock_braking_system.dita" keys="abs" print="yes" />
    </glossarylist>
    <figurelist/>
    <tablelist/>
  </booklists>
  <topicref href="topics/thanks.dita"/>
</backmatter>
```

As you can see, the bookmap offers much better control over the final content of the publication. It also offers more options in controlling the metadata that will go into the PDF (see the Metadata *(on page 1008)* topic).

How to Omit the Front Page, TOC, Glossary, Index for a Plain DITA Map

For a plain DITA map, there are no elements that allow you to control if and where to place the generated content such as the title page, table of contents, list of tables, glossary, or index. For the most common use-case, when you want to hide them all and just keep the content, you can use the transformation parameter `hide.frontpage.toc.index.glossary`. See: Transformation Parameters *(on page 956)*.

Related Information:

- How to Remove Entries from the TOC *(on page 1040)*
- How to Hide the TOC *(on page 1040)*

How to Make Chapters Look Like Individual Publications

Note: This topic is only applicable for the **DITA Map PDF - based on HTML5 & CSS** transformation scenario.

Sometimes you want to make each chapter independent (i.e. it can be read separately, as a separate part of your publication). For this, you need the page counter, figure, and table counters to restart at each chapter. You can control this by using the `args.css.param.numbering` *(on page 1027)* command-line parameter.

In addition to numbering, you can force the creation of a chapter TOC *(on page 1041)*.

Troubleshooting

This section contains information about fixing various publishing problems.

The format-date() XPath Function Does not Respect the Specified Locale

The XPath expressions are evaluated using the Saxon HE processor. This processor does not support languages other than English (https://sourceforge.net/p/saxon/mailman/message/26849522/).

Formatting a date using another language code:

```
title:before {
   content: oxy_xpath('format-date(current-date(), "[Mn] [Y]", "ru", (), ())');
}
```

results in an output like: `[Language: en]september 2019`, with the date being formatted in English.

As a solution, you can either switch to a more language neutral format that avoids the months names:

```
title:before{
   content: oxy_xpath('format-date( current-date(), "[M] [Y]", "en", (), ())');
}
```

or you can use a more complex XPath expression, like this:

```
title:before{


            content: oxy_xpath("let $cm:= format-date(current-date(), '[MNn]') \
return  concat( \
if ($cm= 'January') then  'JAN' else \
if ($cm= 'February') then  'FEB' else \
if ($cm= 'March') then  'MAR' else \
if ($cm= 'April') then  'APR' else \
if ($cm= 'May') then  'MAY' else \
if ($cm= 'June') then  'JUNE' else \
if ($cm= 'July') then  'JUL' else \
if ($cm= 'August') then  'AUG' else \
if ($cm= 'September') then  'SEPT' else \
if ($cm= 'October') then  'OCT' else \
```

```
if ($cm= 'November') then  'NOV' else  '' \
, \
' ', \
format-date(current-date(), '[Y0001]') \
) ");
}
```

Make sure the entire expression is rendered blue in the CSS editor. Replace the capitalized month names with the translation in the desired language.

Error Parsing CSS File - Caused by a Networking Problem

Problem

My custom styles are not applied and in the transformation results console, I get an error containing one of the following: `I/O exception`, `Unknown host`, `Error parsing`.

Cause

One of the CSS files contains references to resources from another website that is currently inaccessible. These resources may include:

- Fonts
- Images
- Other CSS files

Note: If you exported one of the built-in publishing templates from the transformation scenario dialog, it is possible that the associated CSS files use an imported Google Font.

Remedy

1. Check your proxy settings (ask the system administrator for help).
2. If the server is still inaccessible from the transformation process, download the remote resources using a web browser, save them in the customization CSS file folder, and refer them directly from your CSS.

   Note: If the problem is caused by a remote font, see Using Local Fonts.

Glossary Entries Referenced Using 'glossref' are not Displayed

Problem

I have a `<glossgroup>` that contains multiple `<glossentry>` elements and all the entries are referenced using `<glossref>` elements inside my map. When I add an `<abbreviated-form>` element linked to one of my `<glossentry>` elements (using a `@keyref`), the entry is not resolved in the PDF output.

Solution

Make sure every `<glossentry>` has an `@id`. Then, for each `<glossentry>`, declare a `<glossref>` element like this:

```
<glossref href="concepts/glossary.dita#flowers.genus" print="yes" keys="genus"/>
```

Important: For bookmaps, the `<glossref>` elements should be declared in a separate ditamap.

XSL FO-based DITA to PDF Customization

Oxygen XML Developer Eclipse plugin comes bundled with the DITA Open Toolkit that provides a mechanism for converting *DITA maps (on page 1324)* to PDF output. Oxygen XML Developer Eclipse plugin includes a built-in **DITA Map PDF - based on XSL-FO** transformation scenario *(on page 681)* that converts DITA maps to PDF using an *xsl:fo* processor.

There are several methods that can be used to customize DITA to PDF output:

- Create a customization directory that contains your customized files and reference that directory in the PDF transformation scenario (using the `customization.dir` parameter).
- Creating a DITA Open Toolkit plugin that adds extensions to the PDF output. More details can be found in the DITA Open Toolkit Documentation.

  Tip: Some sample plugins are available on GitHub that could help you to get started with creating a plugin:
  - Sample Plugin: DITA-OT PDF Customization Plugin for Oxygen User Manual
  - Sample Plugin: DITA-OT PDF2 - Generate Numbers Before Topic's Title

Using a Customization Directory

One way to customize the PDF output generated by the **DITA Map PDF - based on XSL-FO** transformation scenario *(on page 681)* is to create a dedicated folder to store customized files. With this approach, you will copy the contents of the built-in customization directory to a new directory where you can customize the files according to your needs and reference the new directory using the `customization.dir` parameter in the transformation scenario. The biggest advantage of this method is that the contents of your customization directory will remain unaffected when the DITA-OT is upgraded.

How to Create a Customization Directory

Follow this procedure to create a customization directory:

1. Copy all the entire *`DITA-OT-DIR`*`\plugins\org.dita.pdf2\Customization` directory to another location where you have write access.
2. Modify any of the files in whatever way necessary to achieve your specific goal. For inspiration, see Embedding a Company Logo *(on page 1108)* for a specific example of how you can modify contents of the directory to embed a logo in the output.

   Tip: For other specific examples, see DITA-OT Documentation - PDF Customization Plugin.

3. Edit the **DITA Map PDF - based on XSL-FO** transformation scenario *(on page 681)*, go to the **Parameters** tab, and set the **customization.dir** parameter to point to the location of your customization directory.

Related Information:

Automatic PDF plugin customization generator by Jarno Elovirta.

DITA-OT Documentation - PDF Customization Plugin

Embedding a Company Logo

The following procedure explains how to embed a company logo image in the front matter of the book.

1. Create a customization directory *(on page 1107)* (if you haven't already done so).
2. Create a `cfg\common\artwork` directory structure in your customization directory and copy your logo to that directory (for example, `[C:\Customization\cfg\common\artwork\logo.png`).

   Important: Make sure that your logo image is named: **logo.png**.
3. Rename `Customization\catalog.xml.orig` to: `Customization\catalog.xml`.
4. Open the `catalog.xml` in Oxygen XML Developer Eclipse 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 `Customization\fo\xsl\custom.xsl.orig` to: `C:\Customization\fo\xsl\custom.xsl`
6. Open the `custom.xsl` file in Oxygen XML Developer Eclipse plugin and create the template called **createFrontCoverContents** for DITA-OT 3.4.1.

   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`.

   Example:

   For example, 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,
                      'Configuration/OpenTopic/cfg/common/artwork/logo.png')})"
   />
 </fo:block>
</xsl:template>
</xsl:stylesheet>
```

7. Edit the **DITA Map PDF - based on XSL-FO** transformation scenario *(on page 681)*, go to the **Parameters** tab, and set the **customization.dir** parameter to point to the location of your customization directory.

Tip: For other specific examples, see DITA-OT 3.x Documentation - Customizing PDF Output.

Related Information:

Using a Customization Directory *(on page 1107)*

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.

1. Create a customization directory *(on page 1107)* (if you haven't already done so).
2. Locate the stylesheets and templates listed above in your customization directory and modify them in whatever way necessary to achieve your specific goal.

   Tip: For more information and examples, see the Oxygen PDF Customization Plugin project on GitHub.
3. Edit the **DITA Map PDF - based on XSL-FO** transformation scenario *(on page 681)*, go to the **Parameters** tab, and set the **customization.dir** parameter to point to the location of your customization directory.

Related Information:

https://github.com/oxygenxml/com.oxygenxml.pdf2.ug/wiki

Using a Customization Directory *(on page 1107)*

Adding a Watermark to PDF Output

To add a watermark to the PDF output of a **DITA Map PDF - based on XSL-FO** transformation scenario *(on page 681)*, follow this procedure:

1. Create a customization directory *(on page 1107)* (if you haven't already done so).
2. Create a `cfg\common\artwork` directory structure in your customization directory and copy your watermark image to that directory (for example, `C:\Customization\cfg\common\artwork\watermark.png`).
3. Rename the `Customization\catalog.xml.orig` file to: `Customization\catalog.xml`.

4. Open the `catalog.xml` in Oxygen XML Developer Eclipse 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: `Customization\fo\xsl\custom.xsl.orig` to: `Customization\fo\xsl\custom.xsl`.
6. Open the `Customization\fo\xsl\custom.xsl` file in Oxygen XML Developer Eclipse 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`. Override by copying the original template content in the `custom.xsl` and 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($artworkPrefix,
     'Configuration/OpenTopic/cfg/common/artwork/watermark.png')}">
             <fo:block/>
           </fo:block-container>
          <fo:block xsl:use-attribute-sets="__body__odd__header">
              <xsl:call-template name="insertVariable">
                  <xsl:with-param name="theVariableID" select="'Body odd header'"/>
                  <xsl:with-param name="theParameters">
                     <prodname>
                         <xsl:value-of select="$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 to 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. Override it by copying the original template content in the `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($artworkPrefix,
'Configuration/OpenTopic/cfg/common/artwork/watermark.png')}">
                  <fo:block/>
              </fo:block-container>
  <fo:block xsl:use-attribute-sets="__frontmatter">
       <!-- set the title -->
    <fo:block xsl:use-attribute-sets="__frontmatter__title">
     <xsl:choose>
      <xsl:when test="$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>
```

7. Edit the **DITA Map PDF - based on XSL-FO** transformation scenario *(on page 681)*, go to the **Parameters** tab, and set the **customization.dir** parameter to point to the location of your customization directory.

Adding an Edit Link in PDF Output to Launch Oxygen XML Web Author

You can embed *Edit* links in the DITA Map PDF output that will automatically launch a particular document in Oxygen XML Web Author. A reviewer can then simply click the link and they will be redirected to the Oxygen XML Web Author editing page with that particular file open and editable.

To embed an *Edit* link in the DITA Map PDF output, follow these steps:

1. Edit a **DITA Map PDF - based on XSL-FO** transformation scenario *(on page 681)* and open the **Parameters** tab.
2. Set values for the following parameters:
   - **editlink.ditamap.edit.url** - The URL of the DITA map used to publish your content. The easiest way to obtain the URL is to open the map in Web Author and copy the URL from the browser's address bar.
   - **editlink.additional.query.parameters** - Optional query parameters to be appended to each generated edit link. Each parameter must start with *&* (for example: *&tags-mode=no-tags*).
   - **editlink.remote.ditamap.url** (deprecated) - The URL of the DITA Map suitable for opening in Oxygen XML Web Author.
   - **editlink.web.author.url** (deprecated) - The URL of the Oxygen XML Web Author installation, for example `https://www.oxygenxml.com/oxygen-xml-web-author/`
3. Run the transformation scenario.

Result: In the PDF output, all topics will have an **Edit** link to the right side of the title and clicking the link will launch that particular document in Oxygen XML Web Author.

Force Page Breaks Between Two Block Elements in PDF Output

Suppose that in your DITA content you have two *block elements (on page 1323)*, 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 (on page 1327)* 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>
   ```

The source code for the plugin can be found on GitHub here: https://github.com/dita-community/org.dita-community.pdf-page-break.

Show Comments and Tracked Changes in PDF Output

To include comments and *tracked changes* (stored within your DITA topics) in the PDF output, follow these steps:

1. Edit a **DITA Map PDF - based on XSL-FO** transformation scenario.

2. In the **Parameters** tab, set the value of the **show.changes.and.comments** parameter to `yes`. If you also want to display change bars for inserted or deleted content in the PDF, set the **show.changebars** parameter to `yes`.
3. Optionally, you can configure any of these other parameters to adjust the colors of the comments and tracked changes:
    - **ct.insert.color** - Specifies the color for insertion type tracked changes, as a plain color (e.g. red, yellow, blue), or with a hexadecimal equivalent (e.g. #FFFFFF). The default value is 'blue'.
    - **ct.delete.color** - Specifies the color for deletion type tracked changes, as a plain color (e.g. red, yellow, blue), or with a hexadecimal equivalent (e.g. #FFFFFF). The default value is 'red'.
    - **ct.comment.bg.color** - Specifies the background color for comment type tracked changes, as a plain color (e.g. red, yellow, blue), or with a hexadecimal equivalent (e.g. #FFFFFF). The default value is 'yellow'.
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 (on page 1324)* 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 - Advanced Version *(on page 747)*.

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 *monospace* 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

Adding Libraries to the Built-in FO Processor (DITA-OT)

Starting with Oxygen XML Developer Eclipse plugin version 20.0, both hyphenation and PDF image support are enabled by default in the built-in DITA-OT FO processor. For older version of Oxygen XML Developer Eclipse plugin, use the following procedures to enable such support.

Adding Hyphenation Support for DITA-OT Transformation Scenarios

1. Download the pre-compiled *JAR (on page 1326)* from OFFO.
2. Edit the DITA-OT transformation scenario and switch to the **Advanced** tab.
3. 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.
3. Click the **Libraries** button and add the path to the libraries.

Adding Support for CGM Images

1. Go to the JCGM page and download the `jcgm-image-0.1.1.jar` and `jcgm-core-0.2.0.jar` libraries.
2. Edit the DITA-OT transformation scenario and switch to the **Advanced** tab.
3. Click the **Libraries** button and add the path to the libraries.

Debugging DITA PDF Transformations

To debug a DITA PDF transformation scenario, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*, 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 and create a **DITA Map PDF - based on XSL-FO** 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 *(on page 251)* 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.

   Note: For validation purposes, you need to add the main debugged stylesheet (usually `topic2fo_shell_fop.xsl`) to the Master Files folder *(on page 202)* in the **Navigator** view.
7. In the transformer drop-down menu, select the Saxon EE XSLT processor (the same processor used when the DITA-OT transformation is executed).
8. Click the **Parameters** button and set the `locale` parameter with the value `en_GB` and the `customizationDir.url` parameter 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. If your XSLT stylesheet uses Java extensions, you need to reference the extra JAR libraries by clicking the **Extensions** button and add the libraries in the resulting dialog box. For example, if you have enabled the `show.changes.and.comments` parameter, you need to add the following JAR library for the parameter to have an effect: `DITA-OT\plugins\com.oxygenxml.common\lib\oxygen-dita-publishing-xslt-extensions.jar`.
10. Apply the transformation to continue the debugging process.

Related Information:

Debugging XSLT Stylesheets and XQuery Documents *(on page 1210)*

How to Enable Debugging for FO Processor Transformations *(on page 749)*

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 `[DocBook XSL directory]/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 `[DocBook XSL directory]/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` structure 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 Eclipse plugin and set the customization layer in the **XSL URL** property of the scenario *(on page 692)*.

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 Eclipse plugin.

11.

Working with XPath Expressions

XPath is a language for addressing specific parts of a document. XPath models an XML document as a tree of nodes. An XPath expression is a mechanism for navigating through and selecting nodes from the document. An XPath expression is, in a way, analogous to an SQL query used to select records from a database.

Note: If an XPath expression is run over a JSON document, it is converted to XML and the XPath is executed over the converted XML document.

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.

Examples:

- **child::*** - Selects all children of the root node.
- **.//name** - Selects all elements having the name "name", descendants of the current node.
- **/catalog/cd[price>10.80]** - Selects all the `<cd>` elements that have a price element with a value larger than 10.80.

To find out more about XPath, see http://www.w3.org/TR/xpath.

Related Information:

Content Completion in XPath Expressions *(on page 370)*

XPath Builder View

The **XPath/XQuery Builder** view allows you to compose complex XPath expressions and execute them over the currently edited XML document. For XPath 2.0 / 3.1, you can use the `doc()` function to specify the source file that will have the expressions 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 Eclipse plugin executes the expressions over the XML document in the associated scenario.

Note: If an XPath expression is run over a JSON document, it is converted to XML and the XPath is executed over the converted XML document.

If the view is not displayed, it can be opened by selecting it 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 2.0 SA, XPath 3.1, XPath 3.1 SA, 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 3.1 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 Eclipse plugin uses Saxon to execute XPath 3.1 expressions. Since Saxon implements a part of the 3.1 functions, when using a function that is not implemented, Oxygen XML Developer Eclipse 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 *(on page 248)*.

Favorites button

Allows you to save certain expressions that you can later reuse. To add an expression as a favorite, click this 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 Eclipse 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 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 selected and you navigate through a document, the XPath expression corresponding to the XML node at the current cursor position is displayed. For JSON documents, it displays the XPath expression for the current property.

Evaluate as you type

When you select this option, the XPath expression you are composing is evaluated in real time.

Note: This 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

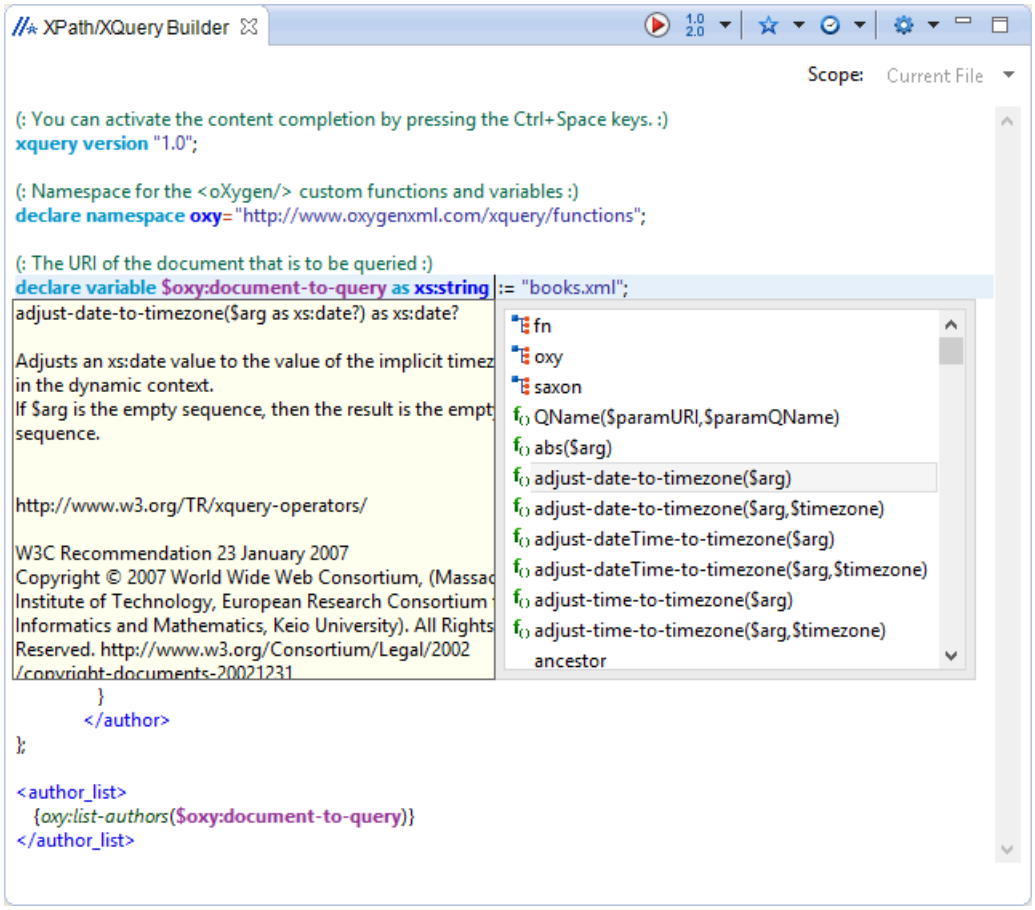
Oxygen XML Developer Eclipse plugin allows you to define a scope for the XPath operation to be executed. You can choose where the XPath expression will be executed:

- **Current file** - Currently selected file only.
- **Enclosing project** - All the files of the project that encloses the currently edited file.
- **Workspace selected files** - The files selected in the workspace. The files are collected from the last selected resource provider view (**Navigator** *(on page 192)*, **Project Explorer** or **Package Explorer**).
- **All opened files** - All files that are opened in the application.
- **Opened archive** - Files that are opened in the **Archive Browser** view *(on page 1126)*.
- **Working sets** - The selected *working sets (on page 1329)*.

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 (on page 1329)*.
- **XPath file filter** - You can filter the files from the selected scope that will have the XPath expression executed. By default, the XPath expression will be executed only on XML or JSON files, but you can also define a set of patterns that will filter out files from the current scope.

Figure 315. XPath/XQuery Builder View



When you hover your cursor over the version icon 1.0 2.0, a tooltip is displayed to let you know what engine Oxygen XML Developer Eclipse plugin currently uses.

While you edit an XPath or XQuery expression, Oxygen XML Developer Eclipse plugin assists you with the following features:

- *Content Completion Assistant (on page 1324)* - 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 Highlighting - Allows you to identify the components of an expression. To customize the colors of the components of the expression, open the **Preferences** dialog box *(on page 36)* and go to **Editor > Syntax Highlight** *(on page 101)*.
- 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 1122)*

XPath Expression Results View

When you run an XPath expression, Oxygen XML Developer Eclipse plugin displays the results of its execution in the **XPath Results** view.

This view contains the following columns:

- **Description** - The result thatOxygen XML Developer Eclipse 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 Eclipse plugin displays **Not available** in the **Location** column. Oxygen XML Developer Eclipse plugin displays the results in a valid XPath expression format.

```
- /node[value]/node[value]/node[value] -
```

XPath Results View Contextual Menu Actions

The following actions are available when the contextual menu is invoked in this view:

Select All

Extends the selection to all the messages from the view.

Copy

Copies information associated with the selected messages in case you want to paste it elsewhere.

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 *(on page 1123)*.

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 *(on page 1123)*.

Save Results as HTML

Saves the complete list of messages in a file in HTML format. For each message, the included details are the same as the ones for the **Copy** action *(on page 1123)*.

Example:

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.

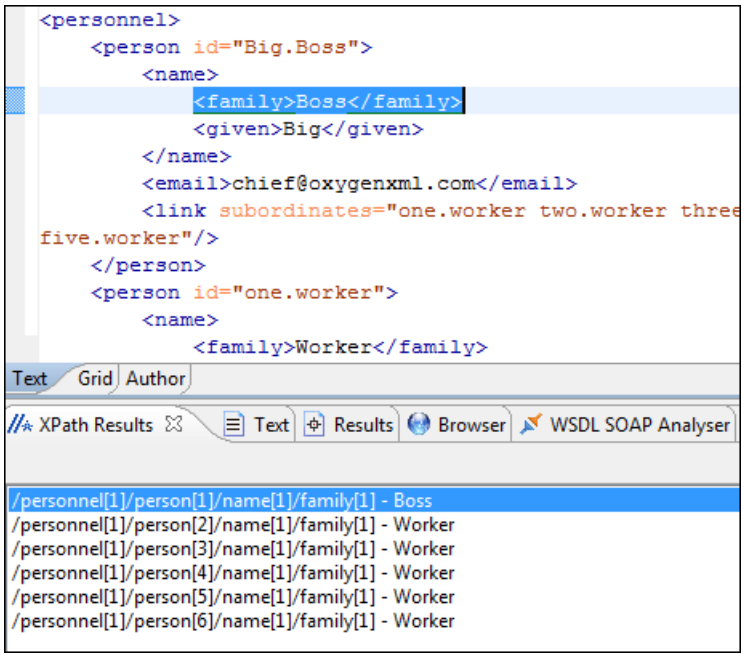
To find all `example` nodes contained in the `sect2` nodes of a DocBook XML document, use the following XPath expression: `//chapter/sect1/sect2/example`. Oxygen XML Developer Eclipse plugin adds a result in the **Results View** for each `example` node found in any `sect2` node.

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.

Figure 316. XPath Results Highlighted in Editor Panel with Character Precision



XPath and XML Catalogs

The evaluation of the XPath expression tries to resolve the locations of documents referenced in the expression through *XML Catalogs (on page 1329)*. These catalogs are configured in the **XML Catalog** preferences *(on page 121)* pages and the **XML Parser** preferences *(on page 123)*.

Example:

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 catalog-enabled parser for parsing these collection files. The class name is *ro.sync.xml.parser.CatalogEnabledXMLReader*. Specify it as it follows:

```
let $docs := collection(iri-to-uri(
   "file:///D:/temp/test/XQuery-catalog/mydocsdir?recurse=yes;select=*.xml;
   parser=ro.sync.xml.parser.CatalogEnabledXMLReader"))
```

XPath Prefix Mapping

To define default mappings between prefixes and namespace URIs go to the **XPath** preferences page *(on page 133)* 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 Eclipse 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.

12.

Working with Archives

Oxygen XML Developer Eclipse plugin includes a useful **Archive Browser** view *(on page 1126)* 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 (on page 1326)*
- Office Open XML (OOXML) files
- Open Document Format (ODF) files
- *IDML files (on page 1325)*

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.

For more information about working with an EPUB archive in Oxygen XML Developer Eclipse plugin, watch our video demonstration:

https://www.youtube.com/embed/OIGTNQwOCl8

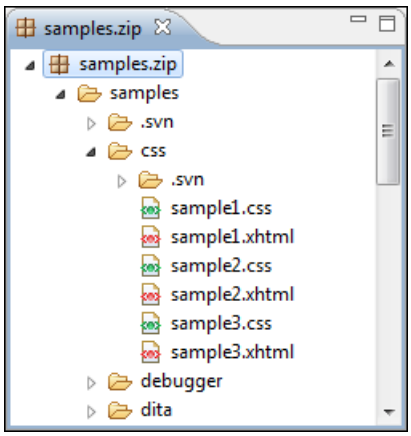
Browsing Archives

To view the contents and structure of an archive, use one of the following methods:

- Open an archive from the **Navigator** view *(on page 192)*. 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 in unmounted.

Tip: If a file is not recognized by Oxygen XML Developer Eclipse plugin as a supported archive type, you can add it from the **Archive** preferences page *(on page 37)*.

Figure 317. Browsing an Archive



Archive Contextual Menu Actions

The following additional actions are available from the contextual menu for archives:

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.

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.

For more information, watch our video demonstration about working with an EPUB in the **Archive Browser** view:

https://www.youtube.com/embed/OIGTNQwOCl8

Working with Archive Files

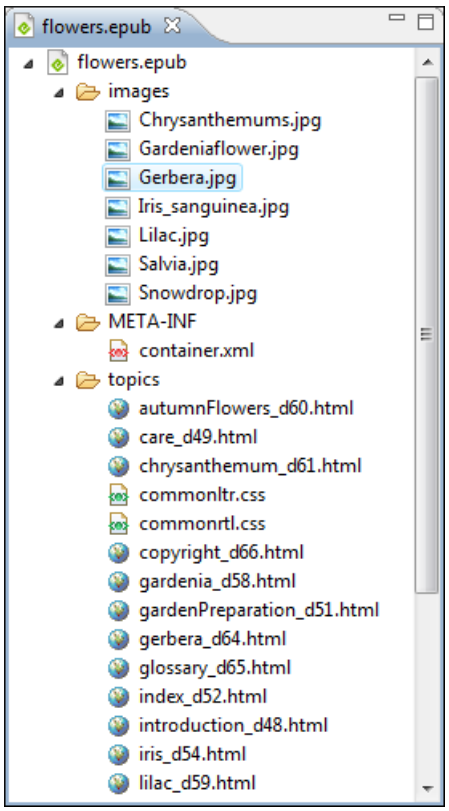
Oxygen XML Developer Eclipse plugin includes support for working with various types of archives, including the following:

- **EPUB** - An e-book file format that can be used on many types of devices, such as smart phones, tablets, e-readers, or computers.
- **OOXML** - An XML-based file format for representing spreadsheets, charts, presentations, and word processing documents.
- **ODF** - An free and open-source XML-based file format for electronic office documents, such as spreadsheets, charts, presentations, and word processing documents.

When these types of files are opened in the **Navigator** view or the main editing pane, their internal components are expanded:

- Document content (XHTML and image files).
- Packaging files.
- Container files.

Figure 318. 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 *(on page 1130)* 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 Eclipse 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.

For more information about working with an EPUB archive in Oxygen XML Developer Eclipse plugin, watch our video demonstration:

https://www.youtube.com/embed/OIGTNQwOCl8

Related Information:

The Archive Browser View *(on page 1126)*

EPUB Document Type (Framework) *(on page 665)*

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 open 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 *(on page 37)*.

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 built-in 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 Eclipse plugin also includes a built-in 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.

13.

Databases and SharePoint

Oxygen XML Developer Eclipse plugin provides support for connecting and integrating with various databases and Microsoft SharePoint. This section includes information about the database-related features in Oxygen XML Developer Eclipse plugin. It explains how to connect with the supported databases, presents the actions that are available for each type, and includes information about SharePoint 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 Eclipse 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 Eclipse plugin helps users with browsing, content editing, importing from databases, using XQuery with databases, SQL execution, and generating XML Schema from a database structure.

The types of connections that are supported in Oxygen XML Developer Eclipse plugin include:

- IBM DB2 *(on page 1136)*
- Microsoft SQL Server *(on page 1140)*
- Oracle Database *(on page 1144)*
- PostgreSQL *(on page 1149)*
- Berkeley DB XML *(on page 1153)*
- eXist *(on page 1159)*
- MarkLogic *(on page 1163)*
- MySQL *(on page 1171)*
- Generic JDBC *(on page 1174)*
- JDBC-ODBC *(on page 1175)*
- BaseX *(on page 1176)*
- WebDAV *(on page 1181)*
- Microsoft SharePoint *(on page 1193)*

Related Information:

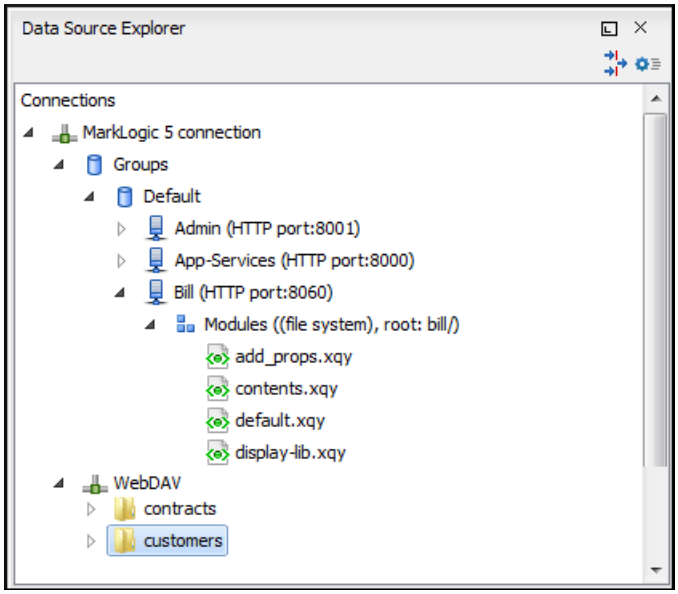
Integration with Microsoft SharePoint *(on page 1193)*

Data Source Explorer View

The **Data Source Explorer** view displays your database connections. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

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 Eclipse plugin supports multiple simultaneous database connections and the connection tree in the **Data Source Explorer** view provides an easy method for browsing them.

Figure 319. 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 *(on page 192)*, 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 *(on page 44)*, allowing you to decide which table types are displayed in the **Data Source Explorer** view.

Configure Database Sources

Opens the **Data Sources** preferences page *(on page 40)* 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 where 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.

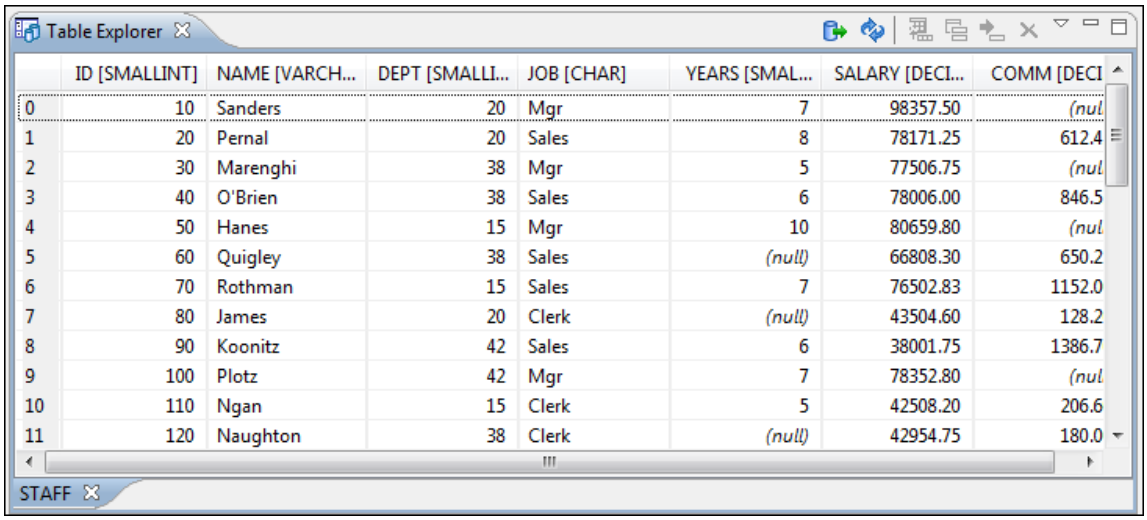
Related Information:

Data Sources Preferences *(on page 40)*

Table Explorer View

Relational databases tables in the **Data Source Explorer** view *(on page 1131)* 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 Eclipse plugin attempts to update the database with the new cell content.

Figure 320. 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 *(on page 44)*). 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 321. Duplicate Entry for Primary Key

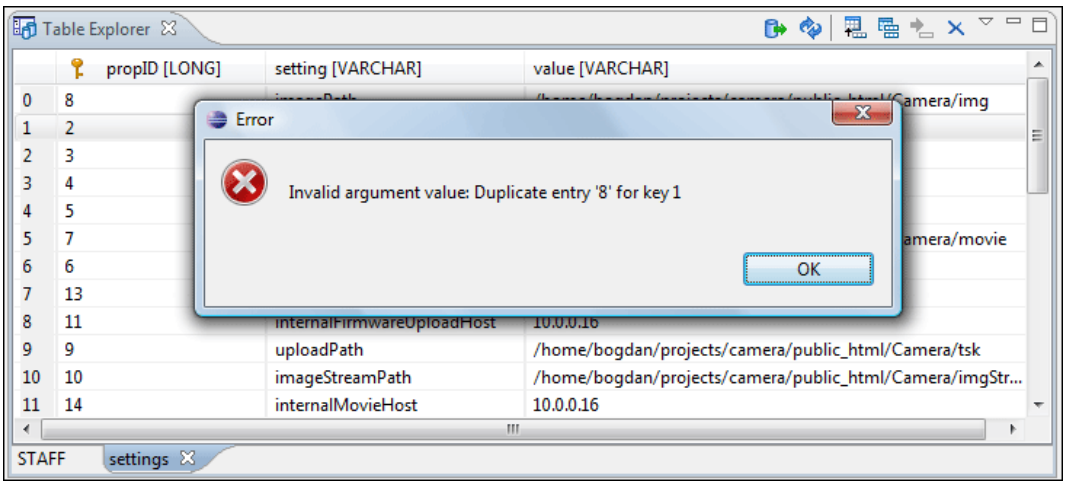


Table Explorer Contextual Menu Actions

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 not available 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 *(on page 1203)* 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 1131)*

Database Connection Support

Oxygen XML Developer Eclipse 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 *(on page 40)* and once configured, the database connections can be viewed and managed in the **Data Source Explorer** view *(on page 1131)*. Oxygen XML Developer Eclipse plugin also includes a **Database** *perspective (on page 165)* that helps you to manage databases.

The database support in Oxygen XML Developer Eclipse plugin offers a variety of capabilities, including:

- Browsing the structure of databases in the **Data Source Explorer** view *(on page 1131)*.
- Viewing relational tables in the **Table Explorer** view *(on page 1133)*.
- 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 Eclipse plugin offers support for the most commonly used relational databases, including:

- IBM DB2
- Oracle 11g
- Microsoft SQL Server
- PostgreSQL
- MySQL

Oxygen XML Developer Eclipse plugin also offers generic support (table browsing and execution of SQL queries) for any JDBC-compliant database (for example, *MariaDB*).

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 Eclipse plugin offers support for the most commonly used native XML databases, including:

- Berkeley DB XML
- eXist
- MarkLogic
- Oracle XML DB
- Base X

Related Information:

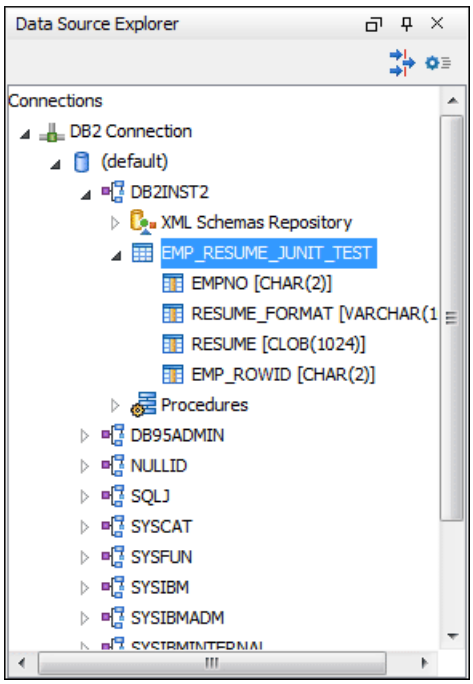
WebDAV Connections *(on page 1181)*

Integration with Microsoft SharePoint *(on page 1193)*

IBM DB2 Database Connections

Oxygen XML Developer Eclipse plugin includes support for IBM DB2 database connections. Oxygen XML Developer Eclipse plugin allows you to browse the structure of an IBM DB2 database in the **Data Source Explorer** view *(on page 1131)*, open tables in the **Table Explorer** view *(on page 1133)*, and perform various operations on the resources in the repository.

Figure 322. 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 Eclipse plugin for configuring a DB2 data source *(on page 1137)*.
2. Configure IBM DB2 *Data Source* drivers *(on page 1137)*.
3. Configure an IBM DB2 Server *Connection (on page 1138)*.
4. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

How to Configure IBM DB2 Data Source Drivers

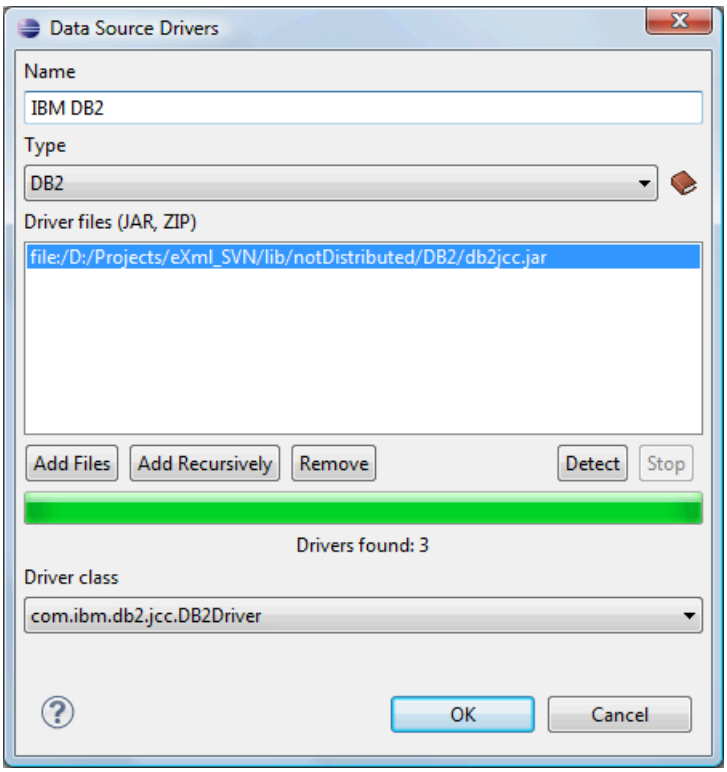
Note: Available in the Enterprise edition only.

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 *(on page 36)* 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 323. 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 *(on page 1138)*.

How to Configure an IBM DB2 Connection

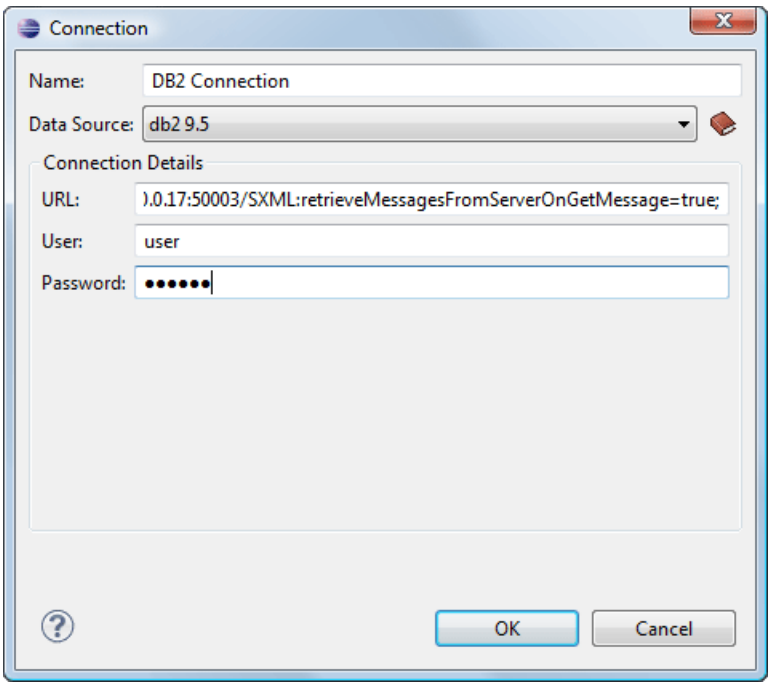
Note: The support to create an IBM DB2 connection is available in the Enterprise edition only.

To configure a connection to an IBM DB2 server, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Data Sources**.
2. Click the **+ New** button in the **Connections** panel.

   The dialog box for configuring a database connection is displayed.

Figure 324. 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 connection configuration.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

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 *(on page 1131)*, depending on the node where 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 *(on page 40)* where you can configure both data sources and connections.

Edit (available on Table nodes)

Opens the selected table in the **Table Explorer** view *(on page 1133)*.

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 *(on page 1203)* chapter).

Database-Specific Contextual Menu Actions

In addition to the general contextual menu actions in the **Data Source Explorer** view *(on page 1131)*, 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 Eclipse plugin.

Microsoft SQL Server Database Connections

Oxygen XML Developer Eclipse plugin includes support for Microsoft SQL Server database connections. Oxygen XML Developer Eclipse plugin allows you to browse the structure of a SQL Server database in the **Data Source Explorer** view *(on page 1131)*, open tables in the **Table Explorer** view *(on page 1133)*, 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: https://docs.microsoft.com/en-us/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-2017.
2. Configure MS SQL Server *Data Source* drivers *(on page 1141)*.
3. Configure a MS SQL Server *Connection (on page 1142)*.

4. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

How to Configure Microsoft SQL Server Data Source Drivers

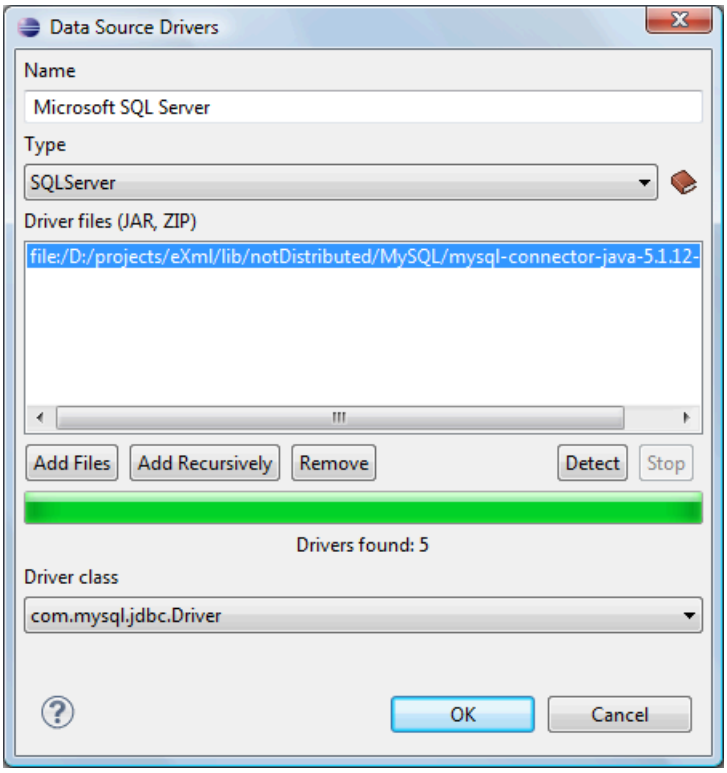
Note: Available in the Enterprise edition only.

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: https://docs.microsoft.com/en-us/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-2017.
2. Open the **Preferences** dialog box *(on page 36)* 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 325. 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 *(on page 1142)*.

How to Configure a Microsoft SQL Server Connection

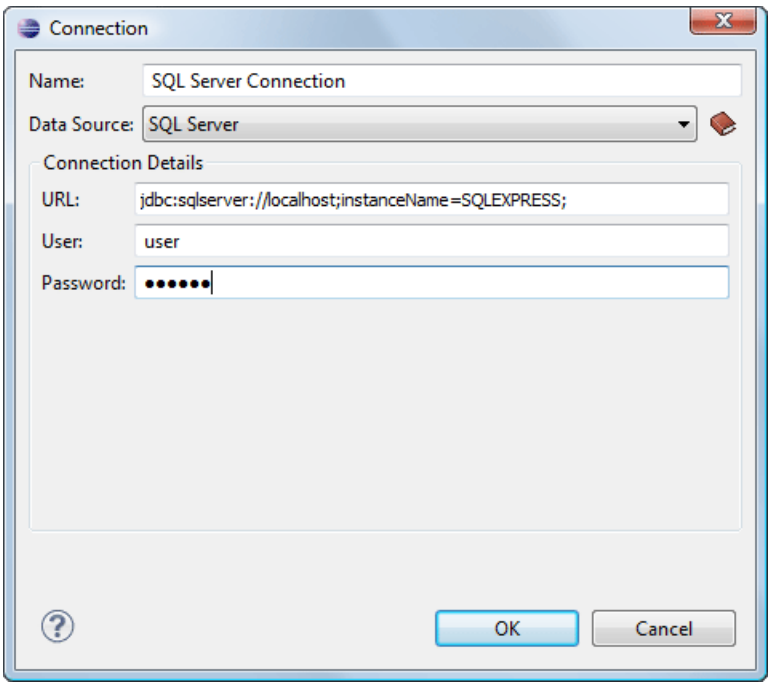
Note: The support to configure a Microsoft SQL Server connection is available in the Enterprise edition only.

To configure a connection to a Microsoft SQL Server, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.

   The dialog box for configuring a database connection is displayed.

Figure 326. 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 connection configuration.

7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

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 *(on page 1131)*, depending on the node where 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 *(on page 40)* where you can configure both data sources and connections.

Edit (available on Table nodes)

Opens the selected table in the **Table Explorer** view *(on page 1133)*.

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 *(on page 1203)* chapter).

Database-Specific Contextual Menu Actions

In addition to the general contextual menu actions in the **Data Source Explorer** view *(on page 1131)*, 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.

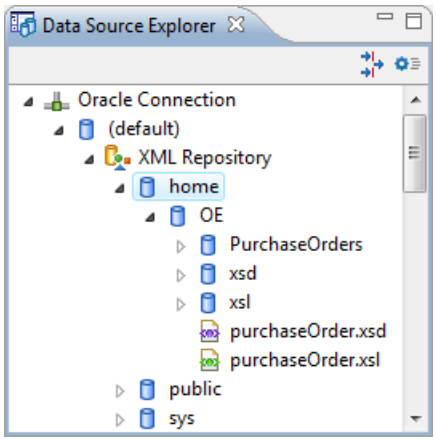
View

Opens the selected schema in Oxygen XML Developer Eclipse plugin.

Oracle Database Connections

The Oracle database is a common relational type of database system. Oxygen XML Developer Eclipse 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 Eclipse plugin allows you to browse Oracle repositories in the **Data Source Explorer** view *(on page 1131)*, open tables in the **Table Explorer** view *(on page 1133)*, and perform various operations on the resources in the repository.

Figure 327. 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 *(on page 1144)*.
3. Configure an Oracle 11g *Connection (on page 1145)*.
4. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

How to Configure Oracle 11g Data Source Drivers

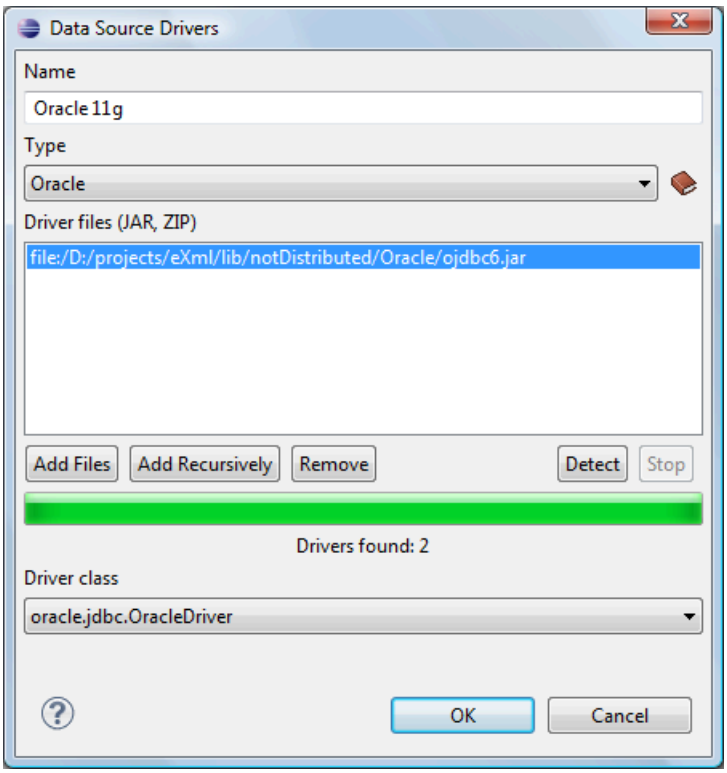
Note: Available in the Enterprise edition only.

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 *(on page 36)* 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 *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 *(on page 1145)*.

How to Configure an Oracle 11g Connection

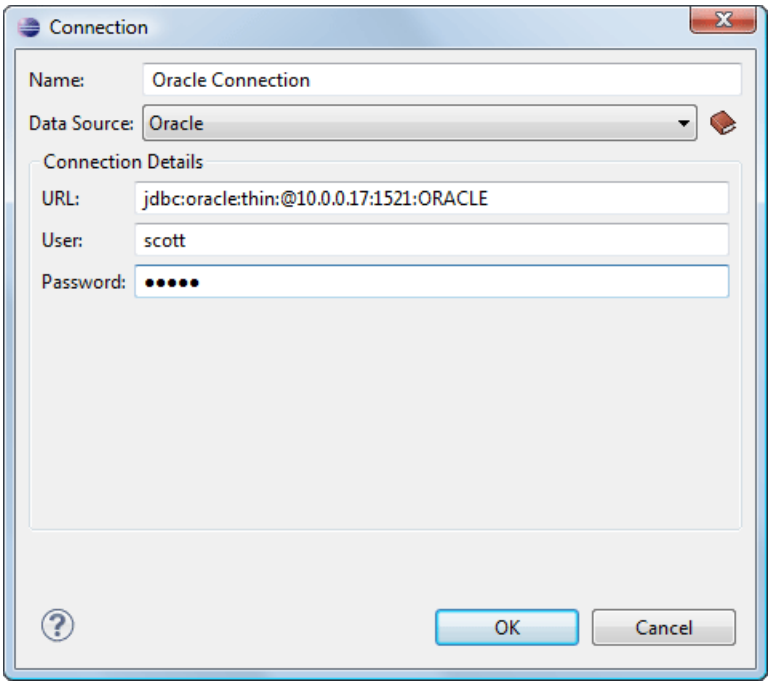
Note: Available in the Enterprise edition only.

To configure a connection to an Oracle 11g server, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Data Sources**.
2. Click the **+ New** button in the **Connections** panel.

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 *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 connection configuration.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

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 *(on page 1131)*, depending on the node where 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 *(on page 40)* where you can configure both data sources and connections.

Edit (available on Table nodes)

Opens the selected table in the **Table Explorer** view *(on page 1133)*.

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 *(on page 1203)* chapter).

Database-Specific Contextual Menu Actions

In addition to the general contextual menu actions in the **Data Source Explorer** view *(on page 1131)*, 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.

Open in System Application

When you use this action, Oxygen XML Developer Eclipse plugin downloads the selected resource to a local temporary folder and opens the selected resource in the system application that is currently set as the default application associated with that type of resource. You can then edit the resource, save it, and when you switch the focus back to the **Data Source Explorer** view, Oxygen XML Developer Eclipse plugin will detect that there was a change and will ask if you want to upload the edited resource to the server.

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.

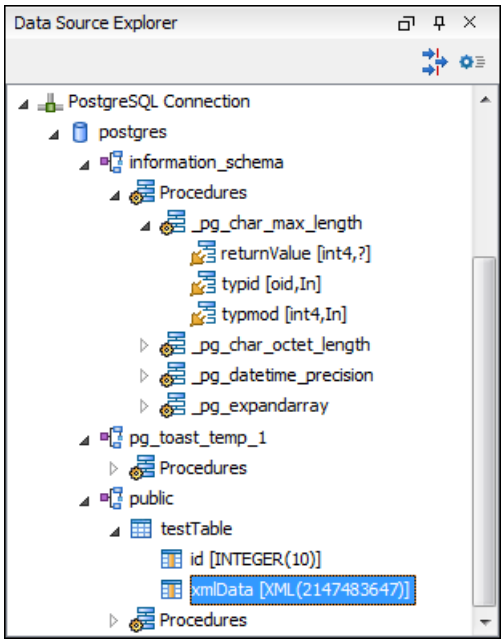
Compare

Compares two selected resources.

PostgreSQL Database Connections

Oxygen XML Developer Eclipse plugin includes support for PostgreSQL database connections. Oxygen XML Developer Eclipse plugin allows you to browse the structure of a PostgreSQL database in the **Data Source Explorer** view *(on page 1131)*, open tables in the **Table Explorer** view *(on page 1133)*, and perform various operations on the resources in the repository.

Figure 330. 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 *(on page 1150)*.
3. Configure a PostgreSQL *Connection (on page 1151)*.
4. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

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 *(on page 36)* 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

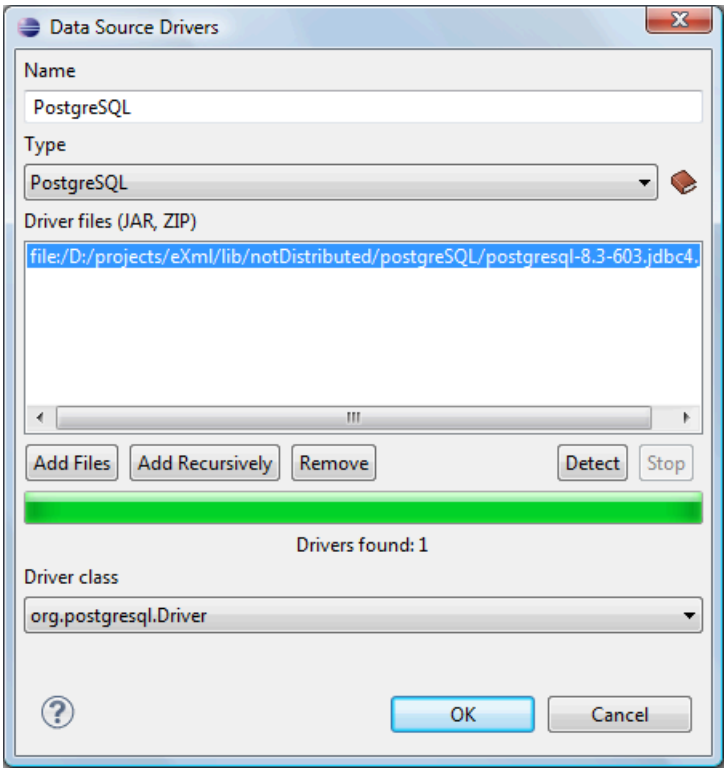
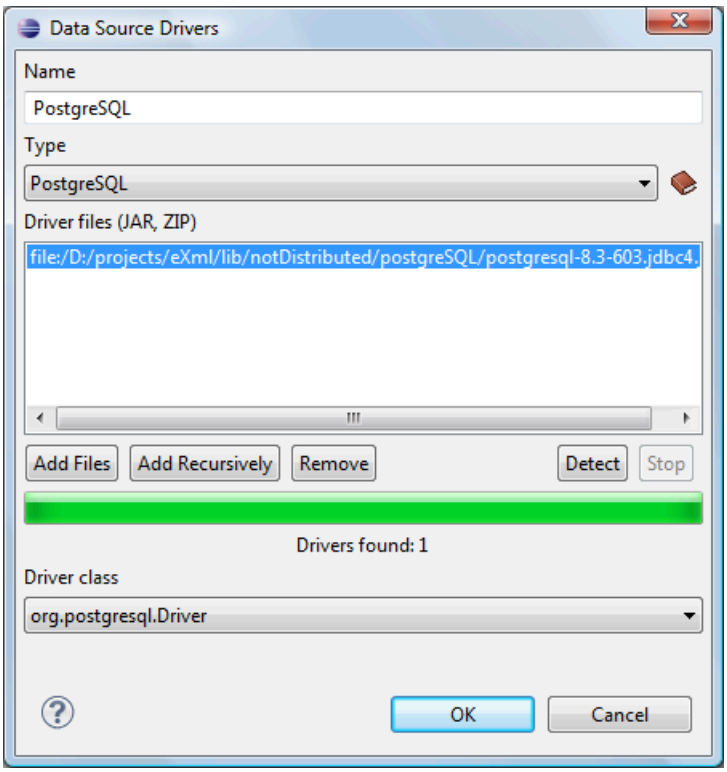


Figure 332. Data Source Drivers Configuration Dialog Box



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 *(on page 1151)*.

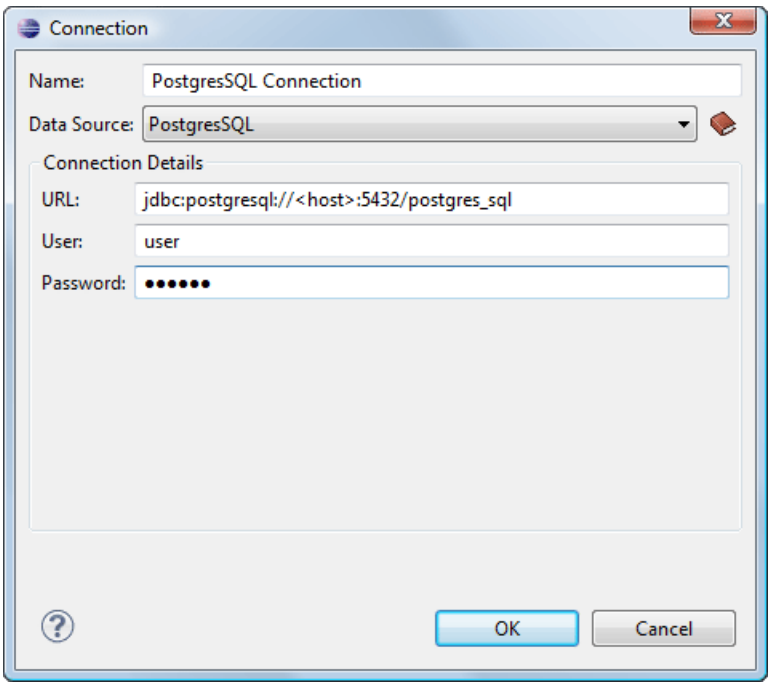
How to Configure a PostgreSQL 8.3 Connection

To configure a connection to a PostgreSQL 8.3 server, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Data Sources**.
2. Click the **+ New** button in the **Connections** panel.

   The dialog box for configuring a database connection is displayed.

Figure 333. 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 connection configuration.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

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 *(on page 1131)*, depending on the node where 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 *(on page 40)* where you can configure both data sources and connections.

Edit (available on Table nodes)

Opens the selected table in the **Table Explorer** view *(on page 1133)*.

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 *(on page 1203)* chapter).

Database-Specific Contextual Menu Actions

In addition to the general contextual menu actions in the **Data Source Explorer** view *(on page 1131)*, the resource level nodes in PostgreSQL connections include the following additional contextual menu action:

Resource Level Nodes

Compare

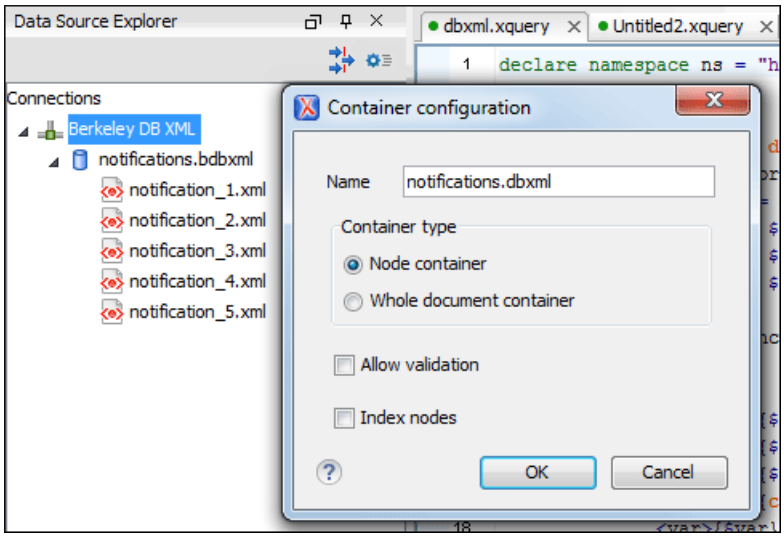
Compares two selected resources.

Berkeley DB XML Database Connections

Oxygen XML Developer Eclipse plugin includes support for Berkeley DB XML database connections. Oxygen XML Developer Eclipse plugin allows you to browse the structure of a Berkeley DB XML database in the **Data Source Explorer** view *(on page 1131)* 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 334. 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 *(on page 1154)*.
2. Configure a Berkeley DB XML *Connection (on page 1154)*.
3. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

How to Configure Berkeley DB XML Data Source Drivers

Prerequisite: For this procedure, you need to already have a Berkeley DB XML database installed on your system.

Oxygen XML Developer Eclipse 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 *(on page 36)* 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 *(on page 1154)*.

How to Configure a Berkeley DB XML Connection

Oxygen XML Developer Eclipse 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 *(on page 36)* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select a previously configured Berkeley data source 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 selected, 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.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

Berkeley DB XML Contextual Menu Actions

While browsing Berkeley DB XML connections in the **Data Source Explorer** view *(on page 1131)*, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the **Data Sources** preferences page *(on page 40)* where you can configure both data sources and connections.

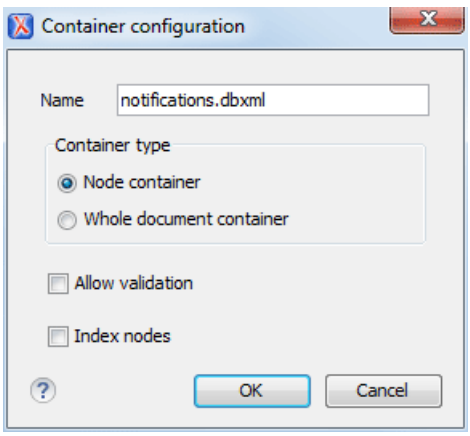
Disconnect

Stops the connection.

New Collection

Opens a **Container configuration** dialog box that allows you to adds a new container in the repository.

Figure 335. 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 selected, documents will be validated when they are loaded into the container. The default behavior is to not validate documents.
- **Index nodes** - If selected, 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.

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

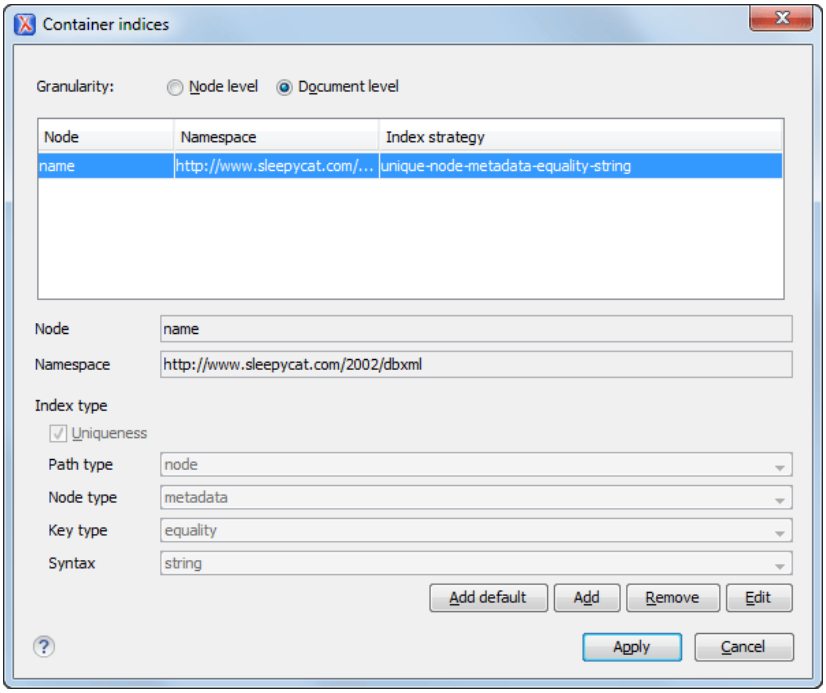
Delete

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 336. 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.

Resource Level Nodes

Open

Opens the selected resource in the editor.

Open in System Application

When you use this action, Oxygen XML Developer Eclipse plugin downloads the selected resource to a local temporary folder and opens the selected resource in the system application that is currently set as the default application associated with that type of resource. You can then edit the resource, save it, and when you switch the focus back to the **Data Source Explorer** view, Oxygen XML Developer Eclipse plugin will detect that there was a change and will ask if you want to upload the edited resource to the server.

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.

Compare

Compares two selected resources.

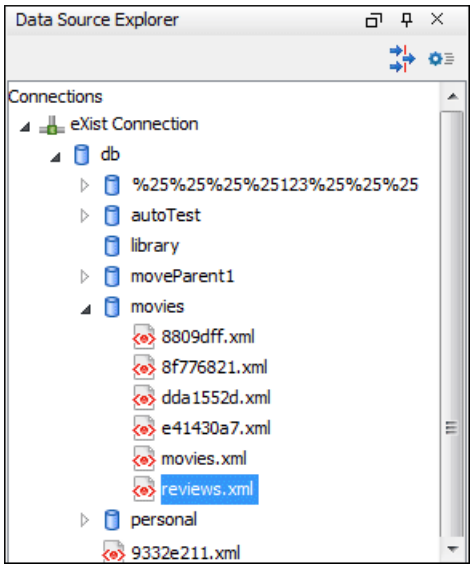
Debugging with Berkeley DB XML

The Berkeley DB XML database added a debugging interface starting with version 2.5. The current version is supported in the Oxygen XML Developer Eclipse plugin XQuery Debugger. The same restrictions and peculiarities *(on page 1169)* apply for the Berkeley debugger as for the MarkLogic debugger.

eXist Database Connections

Oxygen XML Developer Eclipse plugin includes support for *eXist* database connections. Oxygen XML Developer Eclipse plugin allows you to browse the structure of a *eXist* database in the **Data Source Explorer** view *(on page 1131)* and perform various operations on the resources in the repository.

Figure 337. eXist Database Connection



Configuring an eXist Database Connection

There are two ways to configure the support for an *eXist* database:

- Use the dedicated **Create eXist-db XML connection** wizard.
- Use the **Data Sources** preferences page to manually configure your connection.

How to Configure an eXist Connection Using the Built-in Wizard

To configure a connection for an *eXist* database using the dedicated **Create eXist-db XML connection** wizard, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)*, go to **Data Sources** and click the **Create eXist-db XML connection** link.
2. Enter your connection details in the connection wizard and click **OK**.

   Important: To create an *eXist* connection using this wizard, Oxygen XML Developer Eclipse plugin expects the `exist/webstart/exist.jnlp` path to be accessible at the provided **Host** and **Port**.
3. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

How to Configure an eXist Connection Manually

Attention: For this manual procedure, you need to already have an *eXist* database server installed.

Tip: There is an easier way to configure an *eXist* database connection using a built-in wizard. For more information, see How to Configure an eXist Connection Using the Built-in Wizard *(on page 1160)*.

Step 1: Configure eXist Data Source Drivers

Oxygen XML Developer Eclipse plugin supports *eXist* database server versions up to and including version 5.0. To configure a data source for an *eXist* database, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* 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 Eclipse plugin.
   - The `exist.jar` file located in the base directory.
   - All JAR files in the `lib/core/` directory.
6. Click the **OK** button to finish the data source configuration.

Step 2: Configure an eXist Connection

To configure a connection to an *eXist* database, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Data Sources**.
2. Click the **New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select a previously configured *eXist* data source 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.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

For more information about running XQuery against an *eXist* XML database, watch our video demonstration:

https://www.youtube.com/embed/Yoc5h1zSddA

eXist Contextual Menu Actions

While browsing *eXist* database connections in the **Data Source Explorer** view *(on page 1131)*, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the **Data Sources** preferences page *(on page 40)* where you can configure both data sources and connections.

Disconnect

Stops the connection.

Refresh

Performs a refresh on the selected node.

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.

Resource Level Nodes

Open

Opens the selected resource in the editor.

Open in System Application

When you use this action, Oxygen XML Developer Eclipse plugin downloads the selected resource to a local temporary folder and opens the selected resource in the system application that is currently set as the default application associated with that type of resource. You can then edit the resource, save it, and when you switch the focus back to the **Data Source Explorer** view, Oxygen XML Developer Eclipse plugin will detect that there was a change and will ask if you want to upload the edited resource to the server.

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.

Compare

Compares two selected resources.

MarkLogic Database Connections

Oxygen XML Developer Eclipse plugin Enterprise edition includes support for MarkLogic database connections. Once you configure a MarkLogic connection *(on page 1164)*, you can use the **Data Source Explorer** view *(on page 1131)* 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 *(on page 1131)* 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/blogofile/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 to you to spot possible issues without the need of actually executing the XQuery script.

When debugging XQuery files with MarkLogic *(on page 1167)*, you can use the **Data Source Explorer** view *(on page 1131)* to open the files from the application server that is involved in the debugging process. By using the **Data Source Explorer** view *(on page 1131)*, any imported modules are better identified by the MarkLogic server. You can also use step actions and breakpoints *(on page 1169)* in the modules to help identify problems.

Modules Container

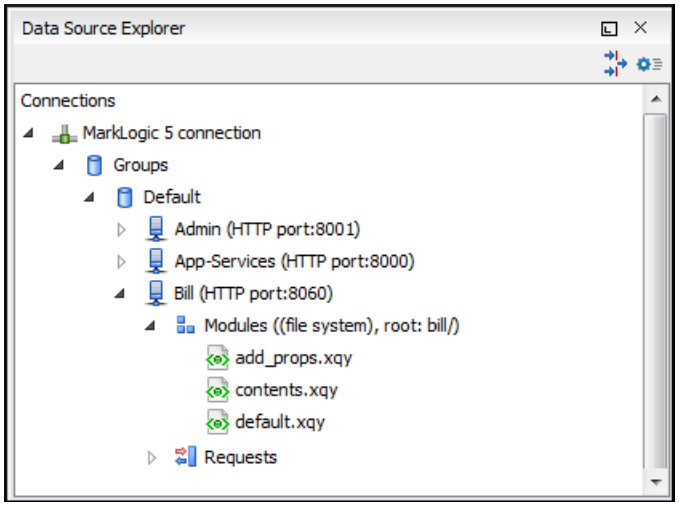
For each Application server (for example: *Bill (HTTP port:8060)*), you have access to the XQuery modules that are visible to that server. When editing, executing, or debugging XQuery it is recommended to open the XQuery files from this **Modules** container.

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 Eclipse 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 *(on page 1171)*.

Figure 338. MarkLogic Connection in Data Source Explorer



Configuring a MarkLogic Database Connection

Note that this feature is available in Oxygen XML Developer Eclipse 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 *(on page 1165)*.
3. Configure a MarkLogic *Connection (on page 1165)*.
4. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

Related Information:

MarkLogic Development in Oxygen XML Developer Eclipse plugin *(on page 1166)*

How to Configure MarkLogic Data Source Drivers

Notes:

- Available in the Enterprise edition only.
- Oxygen XML Developer Eclipse 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 *(on page 36)* 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 *(on page 1165)*.

How to Configure a MarkLogic Connection

Notes:

- Available in the Enterprise edition only.
- Oxygen XML Developer Eclipse 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 *(on page 36)* and go to **Data Sources**.
2. Click the **+ New** button in the **Connections** panel.
3. Enter a unique name for the connection.
4. Select a previously configured MarkLogic data source 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 Eclipse 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 *(on page 1170)* 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, in the **WebDAV URL** field, set the URL used for browsing the MarkLogic database in the **Data Source Explorer** view *(on page 1131)*.
The **Database** field specifies the database that will have the XQuery expressions 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.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

MarkLogic Development in Oxygen XML Developer Eclipse plugin

The Oxygen XML Developer Eclipse 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 *(on page 490)* 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 *(on page 488)* collects and displays all the functions from all imported modules. The *Content Completion Assistant (on page 1324)* 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 *(on page 1131)*. 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 Eclipse 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 Eclipse 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 Eclipse 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 *(on page 1164)*.
2. Expand the MarkLogic connection in the **Data Source Explorer** view *(on page 1131)* and open the *library* modules. The *main* module must also be opened from the **Data Source Explorer** view *(on page 1131)*.
3. Configure a validation scenario *(on page 288)* 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 (on page 1324)* and the **Outline** view *(on page 488)* should now present the functions from all the modules.

Related Information:

Debugging with MarkLogic *(on page 1167)*

Configuring a MarkLogic Database Connection *(on page 1164)*

Debugging with MarkLogic

Oxygen XML Developer Eclipse 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 *(on page 1165)* and a MarkLogic connection *(on page 1165)*.
2. Make sure that the debugging support is enabled in the MarkLogic server that Oxygen XML Developer Eclipse 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 Eclipse plugin uses to process XQuery expressions by selecting the **Use it to execute queries** action *(on page 1170)* from the contextual menu in the **Data Source Explorer** view *(on page 1131)*.

3. Open the XQuery file and start the debugging process.
   - If you want to debug an XQuery file stored on the MarkLogic server, it is recommended to use the **Data Source Explorer** view *(on page 1131)* 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 (on page 164)*. If you have a MarkLogic validation scenario configured for the XQuery file, you can choose to debug the scenario *(on page 1227)* directly.
   - Otherwise, switch to the **XQuery Debugger** *perspective (on page 1327)*, open the XQuery file in the editor, and select the MarkLogic connection in the XQuery engine selector from the debug control toolbar *(on page 1212)*.

     For general information about how a debugging session is started and controlled, see the Working with the Debugger *(on page 1227)* section.

     Note: Before starting a debugging session, it is recommended that you link the MarkLogic connection with an Eclipse project. To do this, go to the **Data Source Explorer** view *(on page 1131)* 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 (on page 1230)* 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 (on page 1229)* to help identify problems. When you add a *breakpoint (on page 1230)* on a line where the debugger never stops, Oxygen XML Developer Eclipse 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 1169)*.

Remote Debugging with MarkLogic

Oxygen XML Developer Eclipse plugin allows you to debug remote applications that use XQuery (for example, web applications that trigger XQuery executions). Oxygen XML Developer Eclipse 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 Eclipse 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.

For more information about the XQuery debugger for MarkLogic, watch our video demonstration:

https://www.youtube.com/embed/eQ4ThDZq1bk

Related Information:

MarkLogic Development in Oxygen XML Developer Eclipse plugin *(on page 1166)*

Configuring a MarkLogic Database Connection *(on page 1164)*

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 (on page 1229)* 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 open 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 *(on page 1131)*, open all the modules from the **Modules** container of the XDBC application server *(on page 1165)* that performs the debugging.
2. Set *breakpoints (on page 1230)* in the module as needed.
3. Continue debugging *(on page 1227)* the query.

If you get a warning that the *breakpoints* failed to initialize, try the following solutions:

- Check the **Breakpoints** view *(on page 1216)* 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 1163)*

MarkLogic Development in Oxygen XML Developer Eclipse plugin *(on page 1166)*

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 *(on page 1225)* and pasting it in the **XWatch** view *(on page 1218)*.

- There is no support for output to source mapping *(on page 1228)*.
- There is no support for showing the trace *(on page 1222)*.
- You can only set breakpoints *(on page 1216)* 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 *(on page 1131)*.
  - When the debugger automatically opens the modules in the Editor.
- No *breakpoints (on page 1229)* are set in modules from the same server that are not involved in the current debugging session.
- No support for profiling *(on page 1230)* when an XQuery transformation is executed in the debugger.

MarkLogic Contextual Menu Actions

While browsing MarkLogic connections in the **Data Source Explorer** view *(on page 1131)*, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the **Data Sources** preferences page *(on page 40)* 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 *(on page 1167)*.

Refresh

Performs a refresh on the selected node.

Container Level Nodes

Enable Debug Mode

Switches the server to a debugging mode. For more information, see MarkLogic debugging sessions *(on page 1167)*.

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.

Open in System Application

When you use this action, Oxygen XML Developer Eclipse plugin downloads the selected resource to a local temporary folder and opens the selected resource in the system application that is currently set as the default application associated with that type of resource. You can then edit the resource, save it, and when you switch the focus back to the **Data Source Explorer** view, Oxygen XML Developer Eclipse plugin will detect that there was a change and will ask if you want to upload the edited resource to the server.

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 1164)*

MarkLogic Development in Oxygen XML Developer Eclipse plugin *(on page 1166)*

Debugging with MarkLogic *(on page 1167)*

MySQL Database Connections

Oxygen XML Developer Eclipse plugin includes support for MySQL database connections. Oxygen XML Developer Eclipse plugin allows you to browse the structure of a SQL Server database in the **Data Source Explorer** view *(on page 1131)*, open tables in the **Table Explorer** view *(on page 1133)*, 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 *(on page 1172)*.
2. Configure a MySQL *Connection*. *(on page 1173)*
3. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

How to Configure MySQL Data Source Drivers

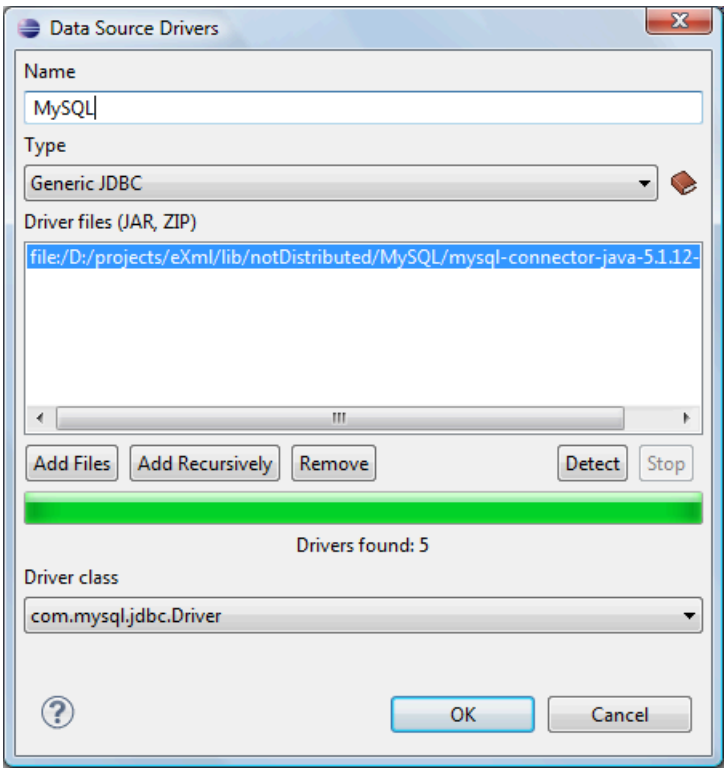
To connect to a MySQL server, you need to create a generic JDBC type data source based on the MySQL JDBC driver available on the MySQL website.

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 *(on page 36)* 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 339. 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 *(on page 1173)*.

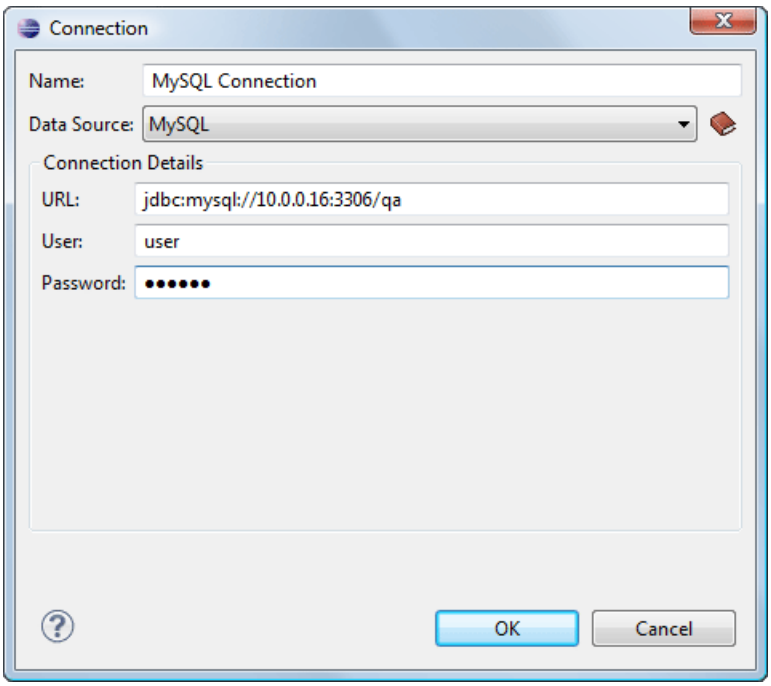
How to Configure a MySQL Connection

To configure a connection to a MySQL server, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Data Sources**.
2. Click the **+ New** button in the **Connections** panel.

   The dialog box for configuring a database connection is displayed.

Figure 340. 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 connection configuration.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

Generic JDBC Database Connections

Oxygen XML Developer Eclipse 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 *(on page 1175)*.
2. Configure a Generic JDBC *Connection (on page 1175)*.
3. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

How to Configure Generic JDBC Data Source Drivers

Starting with version 17, Oxygen XML Developer Eclipse 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 Eclipse 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 *(on page 36)* 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 *(on page 1175)*.

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 *(on page 36)* and go to **Data Sources**.
2. Click the **+ New** button in the **Connections** panel.
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 connection configuration.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

JDBC-ODBC Database Connections

Oxygen XML Developer Eclipse plugin includes support for JDBC-ODBC database connections.

How to Configure a JDBC-ODBC Connection

Starting with version 17, Oxygen XML Developer Eclipse 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

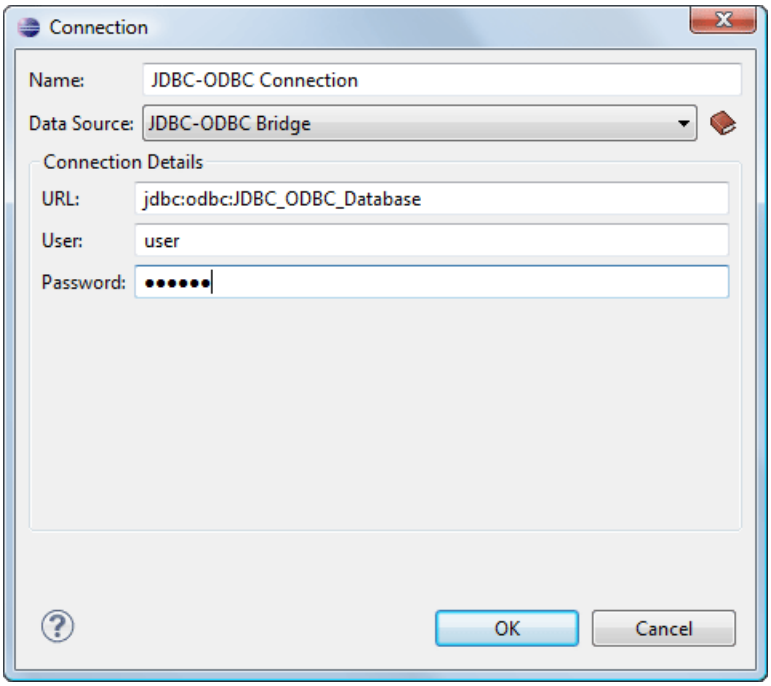
JDBC-ODBC bridge or use a platform-independent distribution of Oxygen XML Developer Eclipse plugin along with a Java VM version 7 or 6.

To configure a connection to an ODBC data source, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* and go to **Data Sources**.
2. Click the + **New** button in the **Connections** panel.

   The dialog box for configuring a database connection is displayed.

   Figure 341. 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 connection configuration.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

BaseX Database Connections

Oxygen XML Developer Eclipse 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 Eclipse plugin allows you to browse the structure of a BaseX database in the **Data Source Explorer** view *(on page 1131)* 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 Eclipse plugin as described in How to Configure a WebDAV Connection *(on page 1181)*. 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 *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

BaseX Contextual Menu Actions

While browsing BaseX connections in the **Data Source Explorer** view *(on page 1131)*, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the **Data Sources** preferences page *(on page 40)* 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.

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.

Resource Level Nodes

Open

Opens the selected resource in the editor.

Open in System Application

When you use this action, Oxygen XML Developer Eclipse plugin downloads the selected resource to a local temporary folder and opens the selected resource in the system application that is currently set as the default application associated with that type of resource. You can then edit the resource, save it, and when you switch the focus back to the **Data Source Explorer** view, Oxygen XML Developer

Eclipse plugin will detect that there was a change and will ask if you want to upload the edited resource to the server.

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.

Compare

Compares two selected resources.

Base X XQJ Connection

XQuery execution is possible in a BaseX connection through an XQJ connection.

Important: The XQJ connector is only capable of running XQuery 1.0 scrips, therefore XQuery 3.0 and 3.1 scripts are not supported.

BaseX XQJ Data Source

First of all, create an XQJ data source as described in How to Configure an XQJ Data Source *(on page 1180)*. 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 *(on page 1180)*.

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 Eclipse plugin and create an XQuery Transformation *(on page 754)*. 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.

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 Eclipse plugin after configuring the environment variables.
2. Open the **Preferences** dialog box *(on page 36)* 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. Click 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 Eclipse 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.
9. Continue on to configure the XQJ connection *(on page 1180)*.

How to Configure an XQJ Connection

The steps for configuring an XQJ connection are the following:

1. Open the **Preferences** dialog box *(on page 36)* 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 XQJ data sources *(on page 1180)* 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 to finish the connection configuration.

WebDAV Connections

Oxygen XML Developer Eclipse plugin includes support for WebDAV server connections. Oxygen XML Developer Eclipse plugin allows you to browse the structure of a WebDAV connection in the **Data Source Explorer** view *(on page 1131)* and perform various operations on the resources in the repository.

How to Configure a WebDAV Connection

By default, Oxygen XML Developer Eclipse 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 *(on page 1131)*.

To configure a WebDAV connection, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* 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 WebDAV data sources in the **Data Source** drop-down menu.
5. Enter the connection details:
   a. Set the URL to the WebDAV repository in the field **WebDAV URL**.
   b. Set the user name that is used to access the WebDAV repository in the **User** field.
   c. Set the password that is used to access the WebDAV repository in the **Password** field.
6. Click the **OK** button to finish the connection configuration.
7. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.
   For more information about the WebDAV support in Oxygen XML Developer Eclipse plugin, watch our video demonstration:

   https://www.youtube.com/embed/vDXO36CqbvM

WebDAV Contextual Menu Actions

While browsing WebDAV connections in the **Data Source Explorer** view *(on page 1131)*, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the **Data Sources** preferences page *(on page 40)* 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.

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.

Resource Level Nodes

Open

Opens the selected resource in the editor.

Open in System Application

When you use this action, Oxygen XML Developer Eclipse plugin downloads the selected resource to a local temporary folder and opens the selected resource in the system application that is currently set as the default application associated with that type of resource. You can then edit the resource, save it, and when you switch the focus back to the **Data Source Explorer** view, Oxygen XML Developer Eclipse plugin will detect that there was a change and will ask if you want to upload the edited resource to the server.

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.

Compare

Compares two selected resources.

SQL Execution Support

The database support in Oxygen XML Developer Eclipse plugin includes support for writing SQL statements, syntax highlighting, *folding (on page 1325)*, and dragging and dropping from the **Data Source Explorer** view *(on page 1131)*. It also includes transformation scenarios for executing the statements, and the results are displayed in the **Table Explorer** view *(on page 1133)*.

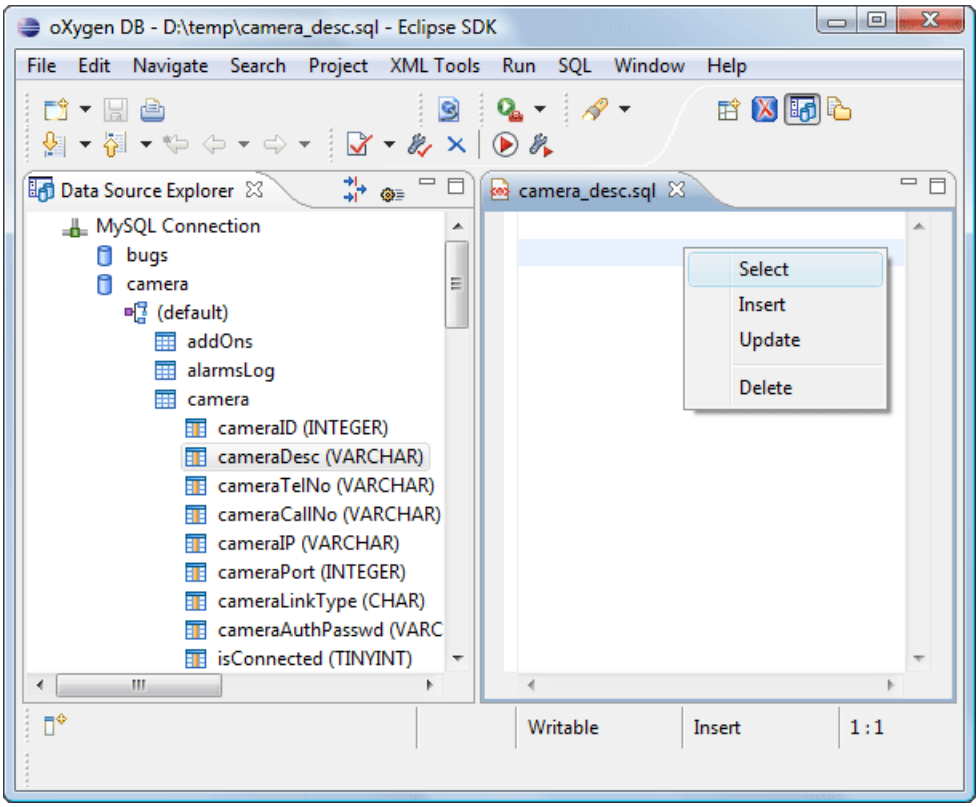
Drag and Drop from Data Source Explorer View

Dragging operations from the **Data Source Explorer** view *(on page 1131)* and dropping them in 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 *(on page 1135)* 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 342. 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 *(on page 669)* 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 inline 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 *(on page 248)* at the bottom of the Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin offer XQuery support:

- *Native XML Databases:*
  - Berkeley DB XML
  - eXist
  - MarkLogic (validation support available starting with version 5)
- *Relational Databases:*
  - IBM DB2
  - Microsoft SQL Server (validation support not available)
  - Oracle (validation support not available)

Related Information:

Editing XQuery Documents *(on page 485)*

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 *(on page 1131)* and dropping them into the editor panel.

1. Configure the data source drivers *(on page 1135)* for the particular relational database in the **Data Sources** preferences page *(on page 40)*.
2. Configure the connection *(on page 1135)* for the particular relational database in the **Data Sources** preferences page *(on page 40)*.
3. Browse the connection in the **Data Source Explorer** view *(on page 1131)*, 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 Validation When Connected to a Database

With Oxygen XML Developer Eclipse plugin, you can validate your XQuery documents when connected to a database. When you open an XQuery document from a connection that supports validation (for example, MarkLogic, or eXist), by default Oxygen XML Developer Eclipse plugin uses this connection for validation. If you open an XQuery file using a MarkLogic connection, the validation resolves imports better.

Related Information:

XQuery Validation *(on page 485)*

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 *(on page 1135)* 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**.

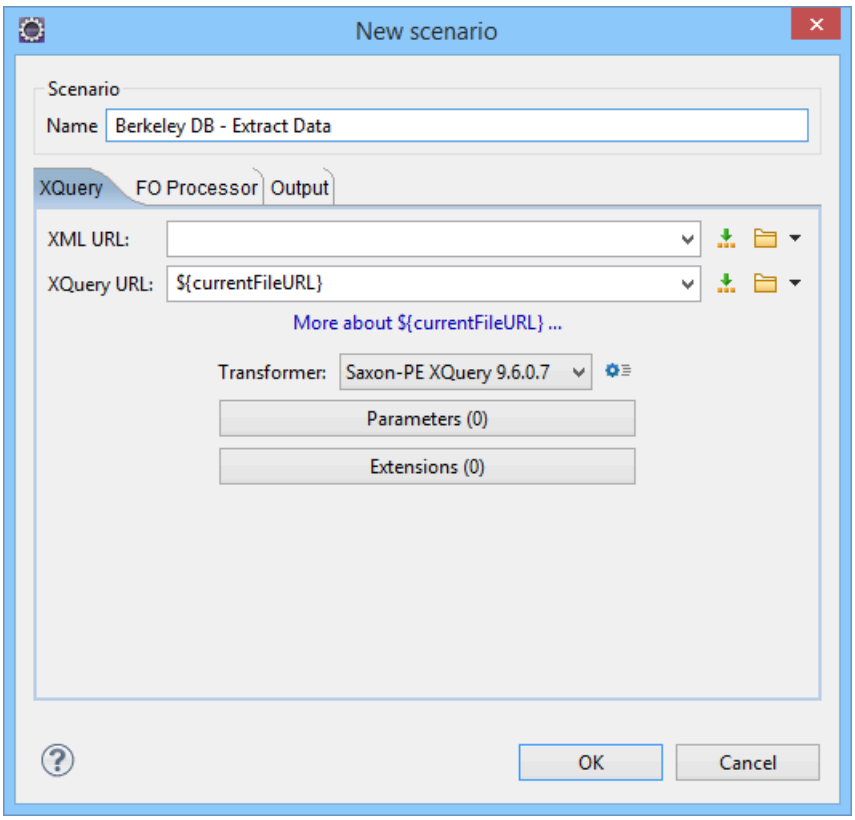
    The **Configure Transformation Scenario** dialog box *(on page 765)* is opened.

    b. Click the **New** button toward the bottom of the dialog box.

    c. Select **XML Transformation with XQUERY** *(on page 707)*.

    The **New Scenario** dialog box for configuring an XQuery scenario is opened.

Figure 343. 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 *(on page 134)* 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 (on page 1324)* 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 Eclipse 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 open XQuery editor panel of Oxygen XML Developer Eclipse plugin).

Related Information:

XML Transformation with XQuery *(on page 707)*

XQuery XQJ Transformation *(on page 1189)*

XQuery XQJ Transformation

XQuery API for Java (XQJ) refers to the common Java API for the XQuery 1.0 specification. The XQJ API enables you to execute XQuery against an XML data source.

Important: The XQJ connector is only capable of running XQuery 1.0 scrips, therefore XQuery 3.0 and 3.1 scripts are not supported.

Oxygen XML Developer Eclipse plugin supports any transformer that offers an XQJ API implementation and it be used for validating XQuery or transforming XML documents.

To configure the support for XQJ, do the following:

1. Configure an XQJ *Data Source (on page 1180)*.
2. Configure an XQJ *Connection (on page 1180)*.
3. To view your connection, go to the **Data Source Explorer** view *(on page 1131)* (if the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu) or switch to the **Database** *perspective (on page 1327)*.

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 Eclipse plugin after configuring the environment variables.
2. Open the **Preferences** dialog box *(on page 36)* 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. Click 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 Eclipse 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.
9. Continue on to configure the XQJ connection *(on page 1180)*.

How to Configure an XQJ Connection

The steps for configuring an XQJ connection are the following:

1. Open the **Preferences** dialog box *(on page 36)* 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 XQJ data sources *(on page 1180)* 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 to finish the connection configuration.

XQuery Database Debugging

Oxygen XML Developer Eclipse 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 Eclipse plugin, see Debugging XSLT Stylesheets and XQuery Documents *(on page 1210)*.

Debugging with MarkLogic

Oxygen XML Developer Eclipse 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 *(on page 1165)* and a MarkLogic connection *(on page 1165)*.
2. Make sure that the debugging support is enabled in the MarkLogic server that Oxygen XML Developer Eclipse 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 Eclipse plugin uses to process XQuery expressions by selecting the **Use it to execute queries** action *(on page 1170)* from the contextual menu in the **Data Source Explorer** view *(on page 1131)*.

3. Open the XQuery file and start the debugging process.
    - If you want to debug an XQuery file stored on the MarkLogic server, it is recommended to use the **Data Source Explorer** view *(on page 1131)* 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 (on page 164)*. If you have a MarkLogic validation scenario configured for the XQuery file, you can choose to debug the scenario *(on page 1227)* directly.
    - Otherwise, switch to the **XQuery Debugger** *perspective (on page 1327)*, open the XQuery file in the editor, and select the MarkLogic connection in the XQuery engine selector from the debug control toolbar *(on page 1212)*.

      For general information about how a debugging session is started and controlled, see the Working with the Debugger *(on page 1227)* section.

      Note: Before starting a debugging session, it is recommended that you link the MarkLogic connection with an Eclipse project. To do this, go to the **Data Source Explorer** view *(on page 1131)* 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 (on page 1230)* 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 (on page 1229)* to help identify problems. When you add a *breakpoint (on page 1230)* on a line where the debugger never stops, Oxygen XML Developer Eclipse 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 1169)*.

Remote Debugging with MarkLogic

Oxygen XML Developer Eclipse plugin allows you to debug remote applications that use XQuery (for example, web applications that trigger XQuery executions). Oxygen XML Developer Eclipse 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 Eclipse 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.

For more information about the XQuery debugger for MarkLogic, watch our video demonstration:

https://www.youtube.com/embed/eQ4ThDZq1bk

Related Information:

MarkLogic Development in Oxygen XML Developer Eclipse plugin *(on page 1166)*

Configuring a MarkLogic Database Connection *(on page 1164)*

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 (on page 1229)* 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 open 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 *(on page 1131)*, open all the modules from the **Modules** container of the XDBC application server *(on page 1165)* that performs the debugging.
2. Set *breakpoints (on page 1230)* in the module as needed.
3. Continue debugging *(on page 1227)* the query.

If you get a warning that the *breakpoints* failed to initialize, try the following solutions:

- Check the **Breakpoints** view *(on page 1216)* 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 1163)*

MarkLogic Development in Oxygen XML Developer Eclipse plugin *(on page 1166)*

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 *(on page 1225)* and pasting it in the **XWatch** view *(on page 1218)*.

- There is no support for output to source mapping *(on page 1228)*.
- There is no support for showing the trace *(on page 1222)*.
- You can only set breakpoints *(on page 1216)* 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 *(on page 1131)*.
  - When the debugger automatically opens the modules in the Editor.
- No *breakpoints (on page 1229)* are set in modules from the same server that are not involved in the current debugging session.
- No support for profiling *(on page 1230)* 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 Eclipse plugin XQuery Debugger. The same restrictions and peculiarities *(on page 1169)* apply for the Berkeley debugger as for the MarkLogic debugger.

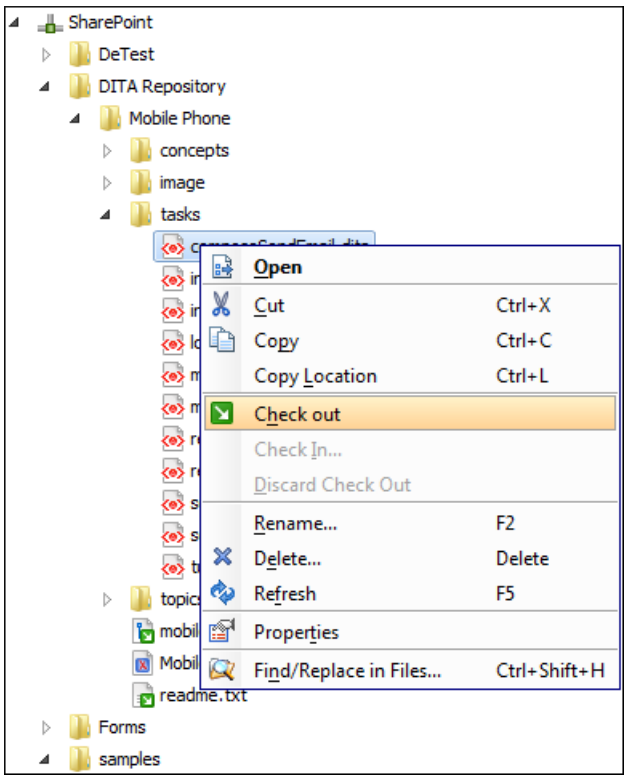
Integration with Microsoft SharePoint

Oxygen XML Developer Eclipse plugin provides support for browsing and managing SharePoint connections in the **Data Source Explorer** view *(on page 1131)*. 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 Eclipse plugin.

Figure 344. SharePoint Connection



Related Information:

Working with Databases *(on page 1131)*

How to Configure a SharePoint Connection

By default, Oxygen XML Developer Eclipse 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 *(on page 1131)*.

To configure a SharePoint connection, follow these steps:

1. Open the **Preferences** dialog box *(on page 36)* 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.

SharePoint Contextual Menu Actions

While browsing SharePoint connections in the **Data Source Explorer** view *(on page 1131)*, the various nodes include the following contextual menu actions:

Connection Level Nodes

Configure Database Sources

Opens the **Data Sources** preferences page *(on page 40)* 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.

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.

Resource Level Nodes

Open

Opens the selected resource in the editor.

Open in System Application

When you use this action, Oxygen XML Developer Eclipse plugin downloads the selected resource to a local temporary folder and opens the selected resource in the system application that is currently set as the default application associated with that type of resource. You can then edit the resource, save it, and when you switch the focus back to the **Data Source Explorer** view, Oxygen XML Developer Eclipse plugin will detect that there was a change and will ask if you want to upload the edited resource to the server.

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.

Compare

Compares two selected resources.

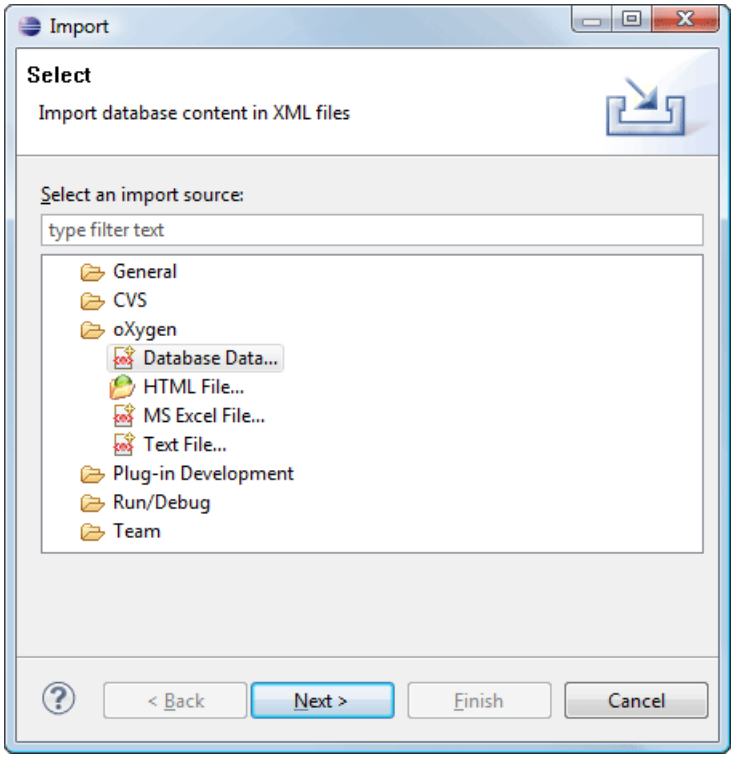
14.

Importing Data

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 Eclipse 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 *(on page 669)*.

Figure 345. Import Wizards of the Oxygen XML Developer Eclipse plugin Plugin



Import from Text Files

Oxygen XML Developer Eclipse plugin includes the possibility of importing text files (`txt` or `csv` file extensions) as XML documents.

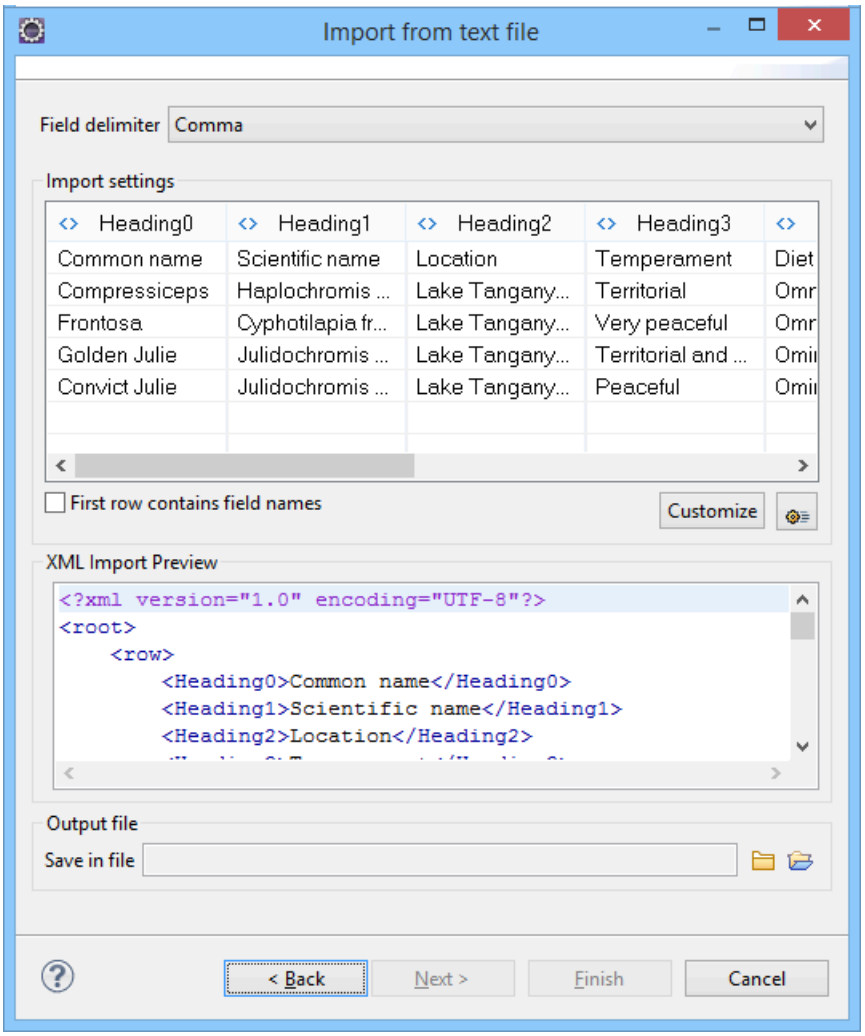
To import a text file into an XML file, follow these steps:

1. Go to **File > Import > Oxygen XML Developer Eclipse plugin > Text File** and click **Next**.
   A **Select text file** dialog box is displayed.
2. Select the URL of the text file (`txt` or `csv` file extensions).
3. Select the encoding of the text file.

4. Click the **Next** button.

   The **Import from text file** dialog box is displayed.

Figure 346. Import from Text File Dialog Box



5. Configure the settings for the conversion.
   a. Select the **Field delimiter** for the import settings. You can choose between the following: `Comma`, `Semicolon`, `Tab`, `Space`, or `Pipe`.
   b. The **Import settings** section presents the input data in a tabular form. By default, all data items are converted to element content (‹› symbol), but this can be overridden by clicking the individual column headers. Clicking a column header once causes the data from this column to be converted to attribute values (= symbol). Clicking a second time causes the column data to be ignored (× symbol) when generating the XML file. You can cycle through these three options by continuing to click the column header.
   c. **First row contains field names** - If this option is selected, 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 options in the drop-down menu: `ELEMENT`, `ATTRIBUTE`, or `SKIPPED`.

e. **Import Settings** - Clicking this button opens the **Import** preferences page *(on page 112)* 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 selected, the new XML document is saved in the specified path.

6. Click **Finish** to generate the XML document.

Import from MS Excel Files

Oxygen XML Developer Eclipse plugin provides several methods for importing MS Excel files into an XML file. You can copy data from Excel and paste it into inserted cells in **Grid** mode. If you want to import an entire Excel file, Oxygen XML Developer Eclipse plugin also offers a configurable import wizard that works with any type of XML document.

Grid Mode Method

The **Grid** mode in Oxygen XML Developer Eclipse plugin displays all content in an XML document as a structured grid of nested tables and you can work with the cells in those tables much like you would with any spreadsheet application. When importing Excel data into **Grid** mode, you first need to insert new cells in the particular nested table and then you can paste data from Excel the same as you would in any table or spreadsheet.

1. Copy the particular cells from your Excel spreadsheet that you want to import into an XML file.
2. Switch to **Grid** mode in Oxygen XML Developer Eclipse plugin.
3. Expand the particular nodes and locate the nested table where you want to insert the copied cells.
4. Right-click a particular row or column where you want to insert the data and select **Insert row** or **Insert column**, depending on the structure of the copied cells.
5. Paste the copied cells from the clipboard into the newly inserted cells in **Grid** mode.
6. You may need to make some manual adjustments. For example, if the selection of copied cells contained an empty cell, Oxygen XML Developer Eclipse plugin might ignore that cell.

Import Wizard Method

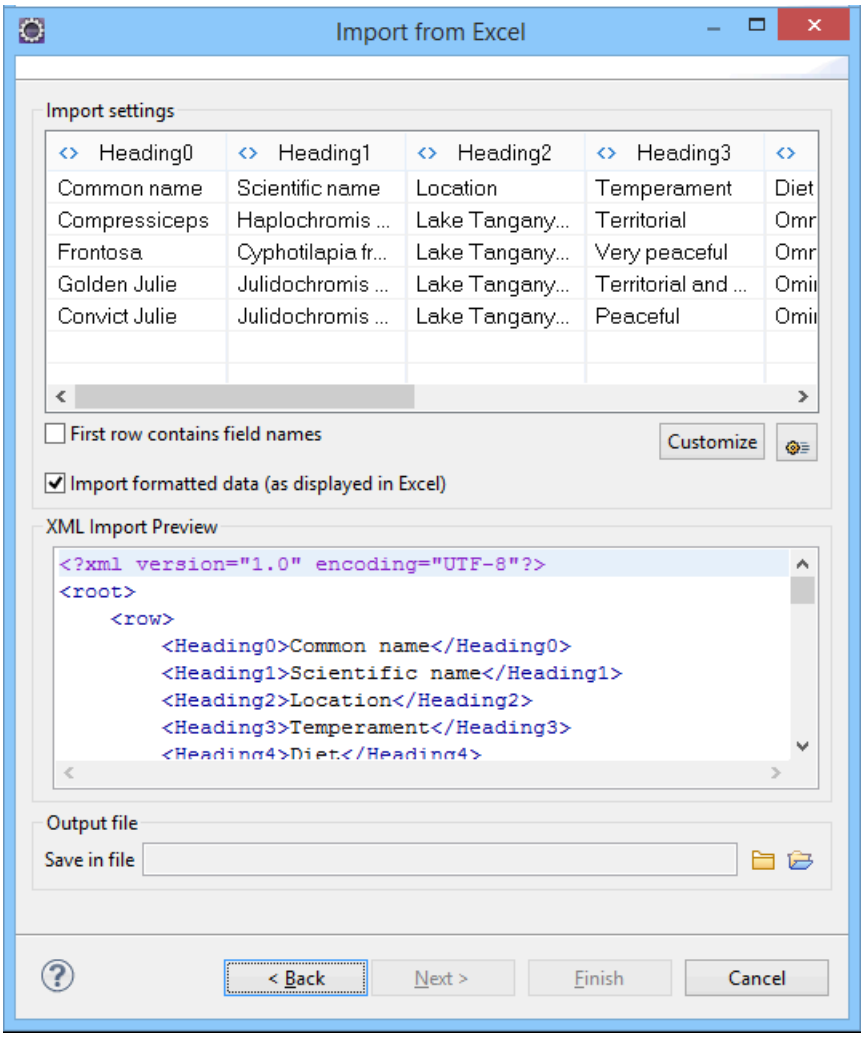
By default, this method supports importing Excel 97/2000/XP/2003 formats out-of-the-box. To import spreadsheet data from Excel 2007 or newer, additional libraries are needed before using this procedure. See Import Data from MS Excel 2007 or Newer *(on page 1202)* for instructions on adding more libraries.

To use the **Import** wizard to import an Excel file into an XML file, follow these steps:

1. Go to **File > Import > Oxygen XML Developer Eclipse 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 to proceed to the next stage of the wizard.

Figure 347. Import Wizard



4. Configure the settings for the conversion. This stage of the wizard offers the following options:

Import settings section

Presents the input data in a tabular form. By default, all data items are converted to element content (<> symbol), but this can be overridden by clicking the individual column headers. Clicking a column header once causes the data from this column to be converted to attribute values (= symbol). Clicking a second time causes the column data to be ignored (× symbol) when generating the XML file. You can cycle through these three options by continuing to click the column header.

First row contains field names

If this option is selected, 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.

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`.

Import Settings

Clicking this button opens the **Import** preferences page *(on page 112)* that allows you to configure more import options.

Import formatted data (as displayed in Excel)

If this option is selected, the imported data retains the Excel data formatting (such as the representation of numeric values or dates). If deselected, the data formatting is not imported.

XML Import Preview panel

Contains an example of what the generated XML document will look like.

Save in file

If selected, the new XML document is saved in the specified path.

5. Click **Finish** to generate the XML document.

For more information about exchanging data between Oxygen XML Developer Eclipse plugin and spreadsheet applications, watch our video demonstration:

https://www.youtube.com/embed/8VwsF58zLkU

Related Information:

Exporting XML Content to Excel *(on page 276)*

Import Data from MS Excel 2007 or Newer

To import spreadsheet data from Excel 2007 or newer (`.xlsx`), Oxygen XML Developer Eclipse plugin needs additional libraries from the release 3.10 of the Apache POI project.

Manually Add the Libraries

To manually 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 into the `plugin.xml` file of the Oxygen XML Developer Eclipse 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`

Note: The `.jar` files can be found in either the root folder where you unpacked the archive, or in the `ooxml-lib` subfolder.

Result: You can now use the **Import** wizard *(on page 1200)* to import data from Excel 2007 or newer.

Related Information:

Exporting XML Content to Excel *(on page 276)*

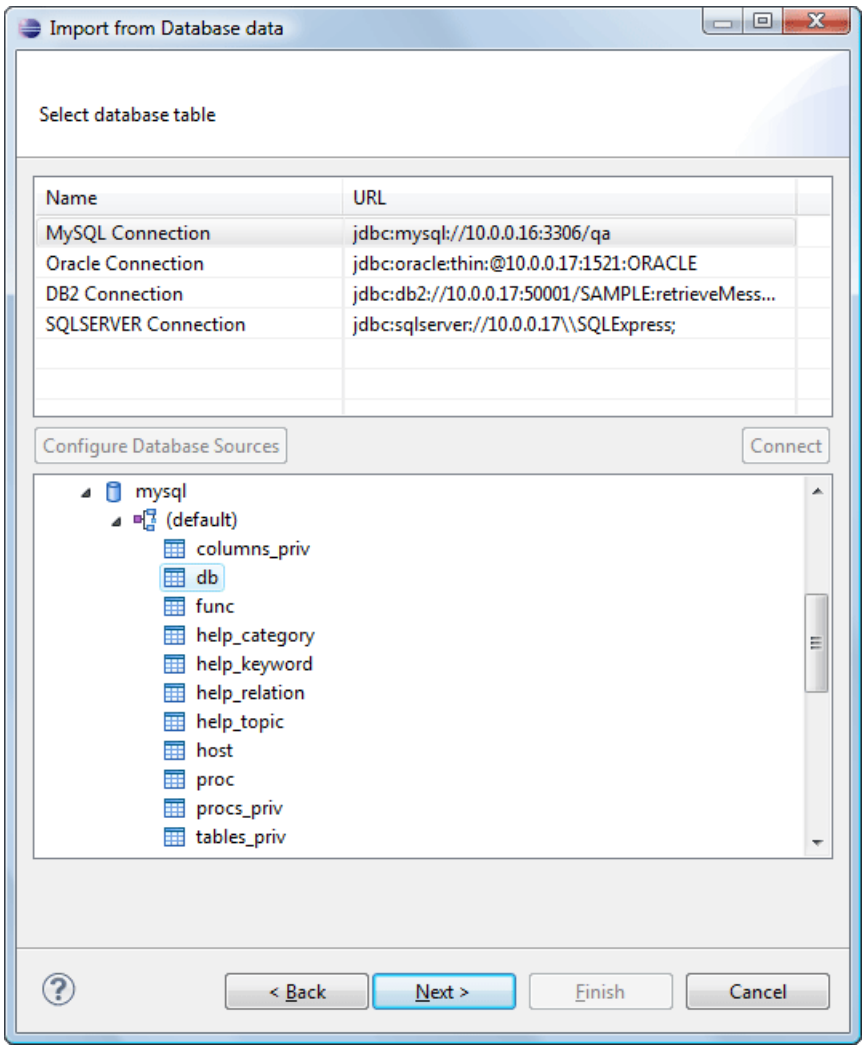
Import Database Data as an XML Document

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 348. Select Database Table Dialog Box



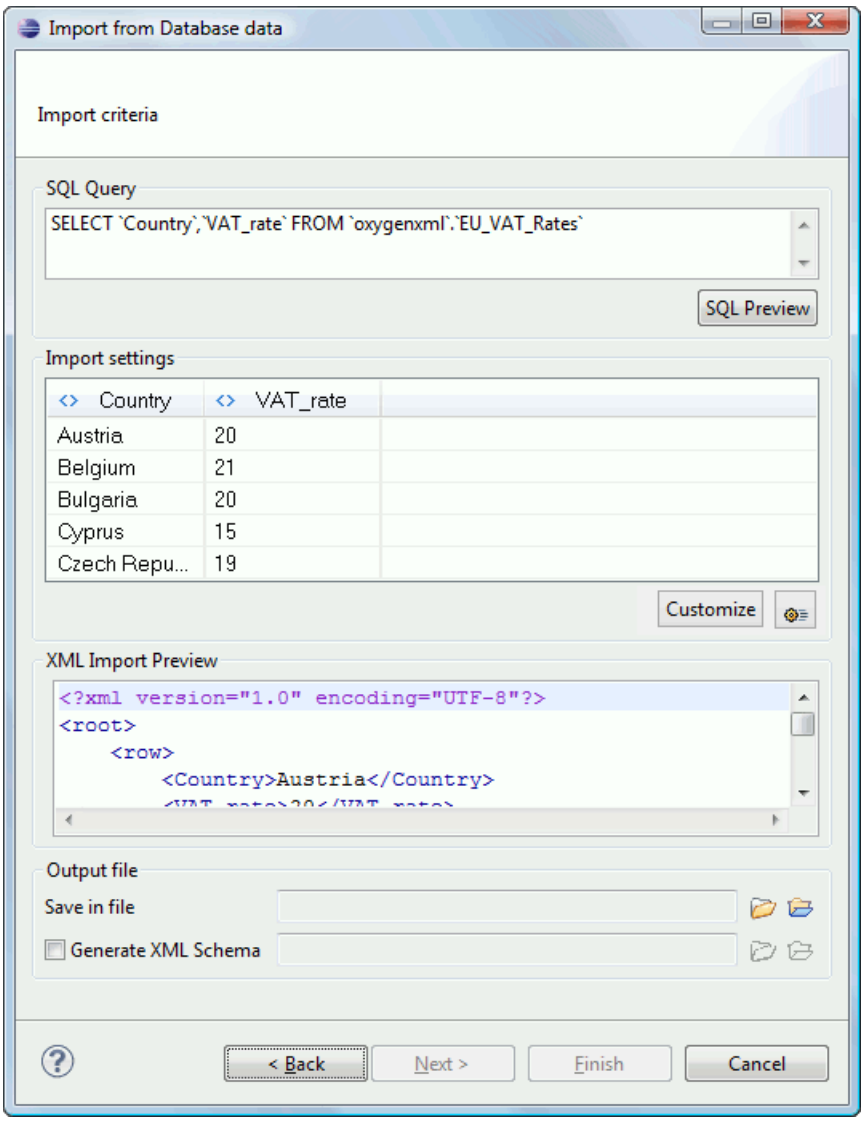
2. Select the connection to the database that contains the appropriate data.

   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 349. Import from Database Criteria Dialog Box



7. Configure the settings for the conversion.
    a. **SQL Preview** - If this button is pressed, the **Import settings** pane displays the labels that are used in the XML document and the first five lines from the database. By default, all data items are converted to element content (‹› symbol), but this can be overridden by clicking the individual column headers. Clicking a column header once causes the data from this column to be converted to attribute values (= symbol). Clicking a second time causes the column data to be ignored (× symbol) when generating the XML file. You can cycle through these three options by continuing to click the column header.
    b. **Customize** - This button opens a **Presentation Names** dialog box that allows you to edit the name, XML name, and conversion criterion for the root and row elements. The XML names can be edited by double-clicking the desired item and entering the label. The conversion criteria can also be modified by selecting one of the following option in the drop-down menu: `ELEMENT`, `ATTRIBUTE`, or `SKIPPED`.
    c. **Import Settings** - Clicking this button opens the **Import** preferences page *(on page 112)* 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 selected, 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 Eclipse plugin offers support for importing HTML files into an XML document.

Import Wizard Method

To use the **Import** wizard to import from HTML files, follow these steps:

1. Go to **File > Import > Oxygen XML Developer Eclipse 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:
   - XHTML5
   - XHTML 1.0 Transitional
   - XHTML 1.0 Strict
5. Click the **Finish** button.

Result: 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 XHTML5, XHTML Transitional, or XHTML Strict depending on the option you chose.

Import Content Dynamically

Along with the built-in support for various useful URL protocols (such as HTTP or FTP), Oxygen XML Developer Eclipse 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:my.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 (on page 1329)* 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 (on page 1324)* 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 Eclipse 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 (on page 1326)* 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 Eclipse plugin to resolve such references during a special pre-processing stage. Depending on the conversion, you may also require additional libraries to be added using the **Libaries** button in the **Advanced** tab of the transformation scenario.

Related Information:

https://github.com/oxygenxml/dita-glass

15.

Debugging XSLT Stylesheets and XQuery Documents

Oxygen XML Developer Eclipse 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 (on page 1327)* 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 (on page 1327)* 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 (on page 1327)* allows 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 9.9.1.5 XSLT engine) and XQuery 1.0 / 3.0 (using Saxon 9.9.1.5 XQuery engine).
- Stepping capabilities: step in, step over, step out, run, run to cursor, run to end, pause, stop.
- Output to source mapping between every line of output and the instruction element / source context that generated it.
- *Breakpoints (on page 1229)* 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.

For even more information, watch our video demonstration:

https://www.youtube.com/embed/m9d8c4V-LJw

Debugger Layout

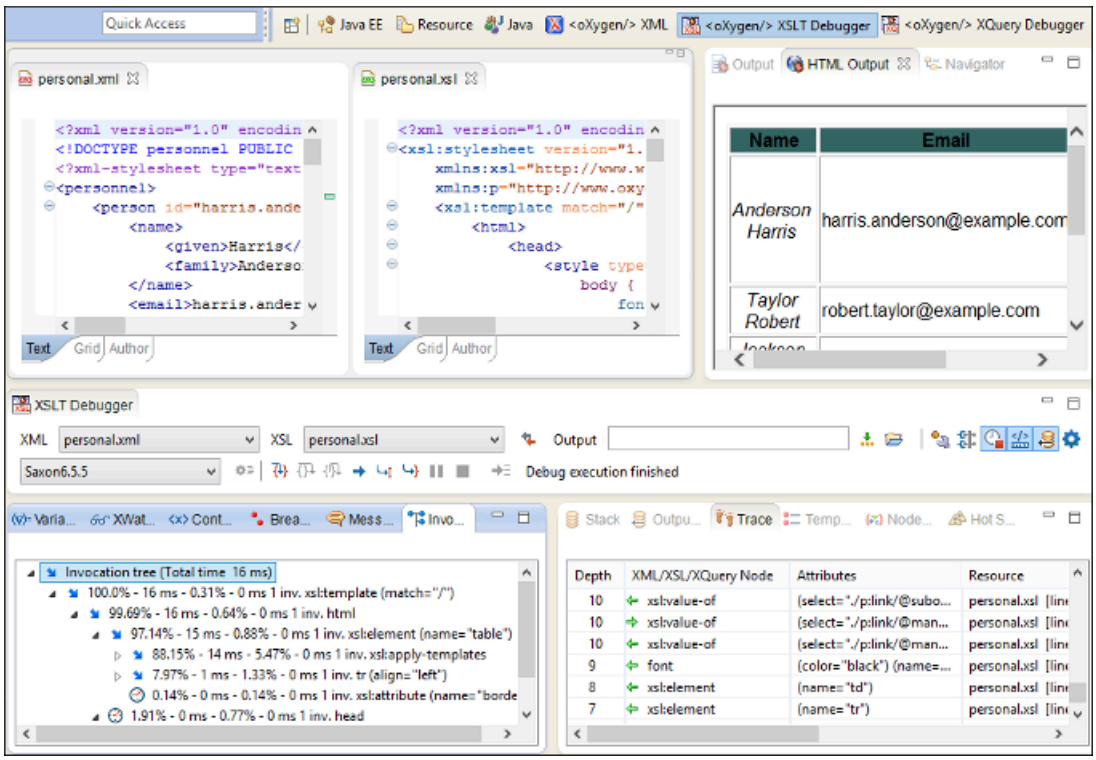
The XML and XSL files are displayed in **Text** mode *(on page 222)*. The **Grid** mode *(on page 168)* is available only in the Editor *perspective (on page 161)*.

The **XSLT/XQuery Debugger** *perspective (on page 1327)* contains the following components:

- **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 (using the **Run** button on the Control toolbar *(on page 1214)*). Several actions are available in the contextual menu for this view, including **Find/Replace**, **Copy**, and **Format and Indent**. There are two types of output views: a text-based **Output** view (with XML syntax highlights) and **HTML** view.
- **Control Toolbar** *(on page 1212)* - Contains a variety of actions to help you configure and control the debugging process.
- **Information Views** *(on page 1215)* - The information views at the bottom of the editor display various types of information to help you understand the transformation process.

Tip: The **Output** view and the various other information views are dockable *(on page 1324)* so that you can configure the workspace according to your preferences.

Figure 350. Debugger Interface



XML documents and XSL stylesheets or XQuery documents that were opened in the **Editor** *perspective (on page 1327)* 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 that you are used to in 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.

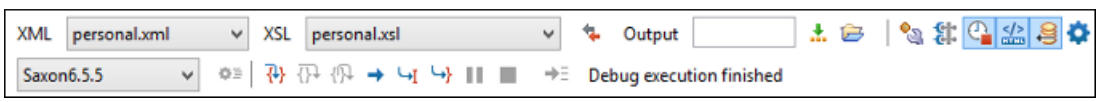
Related Information:

- Steps in a Typical Debugging Process *(on page 1227)*
- Identify the XSLT / XQuery Expression that Generated Particular Output *(on page 1228)*
- Supported Processors for XSLT / XQuery Debugging *(on page 1236)*
- Performance Profiling of XSLT Stylesheets and XQuery Documents *(on page 1230)*

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 351. Control Toolbar



XML source selector

The current selection represents the source document used as input by the transformation engine. The selection list contains all open 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 open files (XSLT / XQuery files being emphasized).

Link with editor

When selected, the XML and XSLT/XQuery selectors display the names of the files open in the central editor panels. This button is toggled off 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** *(on page 148)* 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.

Edit extensions

Allows you to add and remove the Java classes and *JARS (on page 1326)* used as XSLT extensions.

/ Turn on/off profiling

Enables / Disables current transformation profiling.

Enable XHTML output

Enables the rendering of the output in the **HTML** output view *(on page 1211)* 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 (on page 1327)* by 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 *(on page 131)*.

XSLT / XQuery engine selector

Lists the processors available for debugging XSLT and XQuery transformations *(on page 1236)*.

XSLT / XQuery engine advanced options

If Saxon HE/PE/EE is selected, you can click this button to open the Advanced Saxon Transformation Options page *(on page 695)*.

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 Shift + F5

Starts the debugging process. The execution of the process is paused when a breakpoint *(on page 1216)* 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 (on page 1229)* is reached or the execution ends.

Run to end Alt + F5

Runs the transformation until the end, without taking into account enabled *breakpoints (on page 1229)*, 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 710)*

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

- **Breakpoints** view *(on page 1216)*
- **XWatch** view *(on page 1218)*
- **Context** view *(on page 1217)*
- **Messages** view *(on page 1219)* (XSLT only)
- **Variables** view *(on page 1225)*
- **Invocation Tree** view *(on page 1232)*

Right side information views

- **Stack** view *(on page 1220)*
- **Output Mapping Stack** view *(on page 1220)*
- **Trace** view *(on page 1222)*
- **Templates** view *(on page 1223)* (XSLT only)
- **Nodes/Values Set** view *(on page 1224)*
- **Hotspots** view *(on page 1233)*

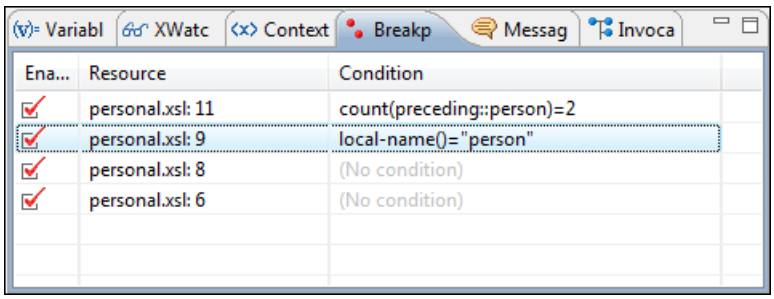
Tip: The information views are dockable *(on page 1324)* so that you can configure the workspace according to your preferences.

Breakpoints View

The **Breakpoints** view lists all *breakpoints (on page 1229)* that are set on open documents. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu. *Breakpoints* can be inserted *(on page 1230)* in the XML source document or the XSLT/XQuery document in debugging sessions.

Once you insert a *breakpoint*, it is automatically added to the list in the **Breakpoints** view and you can edit its associated *condition*. A *breakpoint* can have an associated break condition that represents an XPath expression evaluated in the current debugger context. For them 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 352. Breakpoints View



The **Breakpoints** view contains the following columns:

- **Enabled** - If selected, 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*.

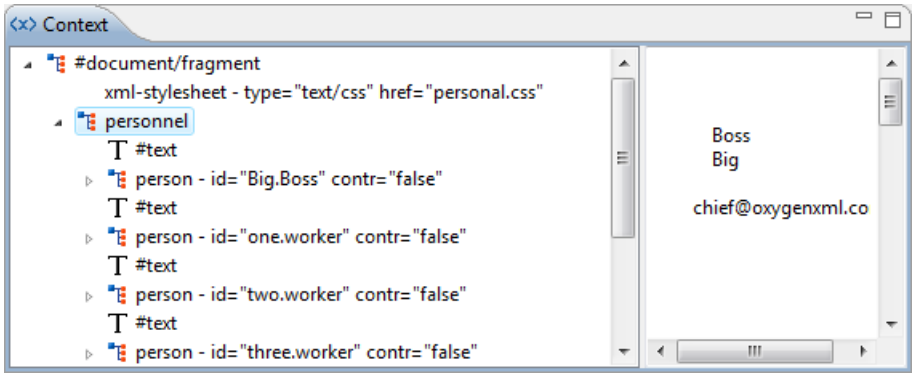
Related Information:

Using Breakpoints *(on page 1229)*

Context 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 *(on page 1218)*. The value of the context node is presented as a tree in the **Context** view. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

Figure 353. Context node view



The context nodes are 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 by selecting it 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 Eclipse plugin supports you with syntax highlight and content completion assistance *(on page 370)*.

Figure 354. XPath Watch View

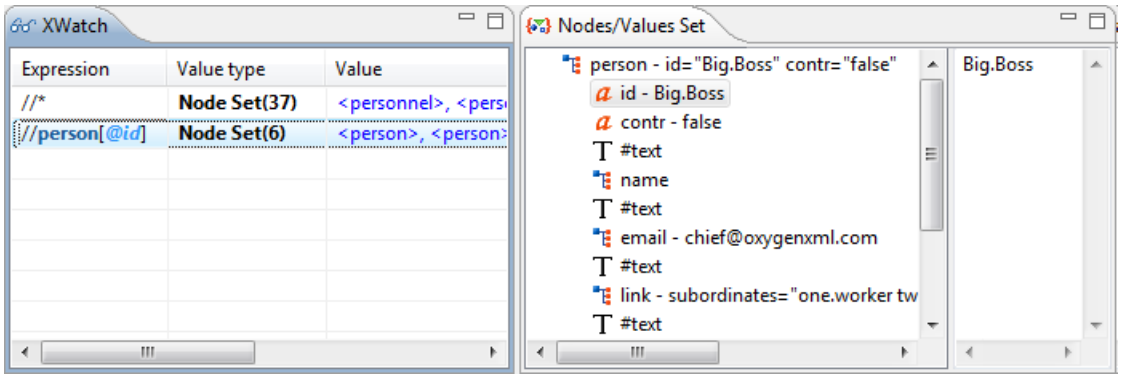


Table 36. XWatch columns

Column	Description
Expression	XPath expression to be evaluated (XPath 1.0 or 2.0 / 3.0 compliant).
Value	Result of XPath expression evaluation. Value has a type (see the possible values *(on page 1225)* in the Variables View *(on page 1225)*) section. For *Node Set* results, the number of nodes in the set is shown in parenthesis.

Important: Notes 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 *(on page 1224)*.
-

Messages View

Using an **xsl:message** instruction is 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 by selecting it from the **Window > Show View** menu.

Figure 355. Messages View

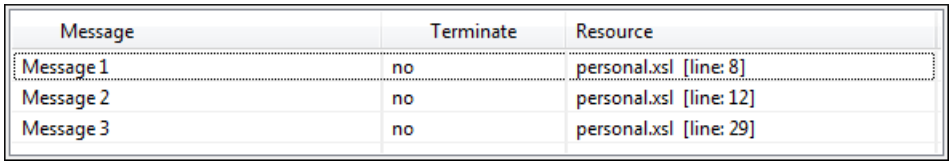


Table 37. Messages columns

Column	Description
Message	Message content.
Terminate	Signals whether or not the processor terminates the transformation once it encounters the message (yes/no respectively).
Resource	Resource file where *xsl:message* instruction is defined and the message line number.

The following actions are available in the contextual menu:

Go to

Highlight the XSL fragment that generated the message.

Copy

Copies to clipboard message details (system ID, severity info, description, start location, terminate state).

Clear all

Removes all messages from the view.

Important:

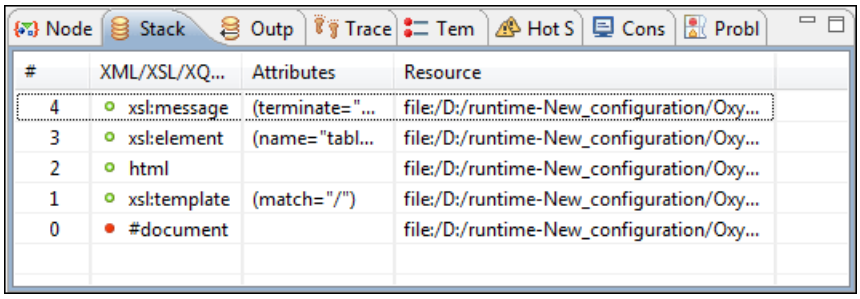
- Clicking a record from the table highlights the `xsl:message` declaration line.
- Message table values can be sorted by clicking the corresponding column header. Clicking the column header switches the sorting order between: ascending, descending, no sort.

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 by selecting it from the **Window > Show View** menu.

During the transformation, two stacks are managed. One for source nodes being processed and the other for XSLT/XQuery nodes being processed. Oxygen XML Developer Eclipse plugin shows both node types in 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.

Figure 356. 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 38. Stack Columns

Column	Description
#	Order number, represents the depth of the node (0 is the stack base).
XML/XSLT/XQuery Node	Node from source or stylesheet document currently being processed. One particular stack node is the document root, noted as **#document**.
Attributes	Attributes of the node (a list of *id="value"* pairs).
Resource	Resource file where the node is located.

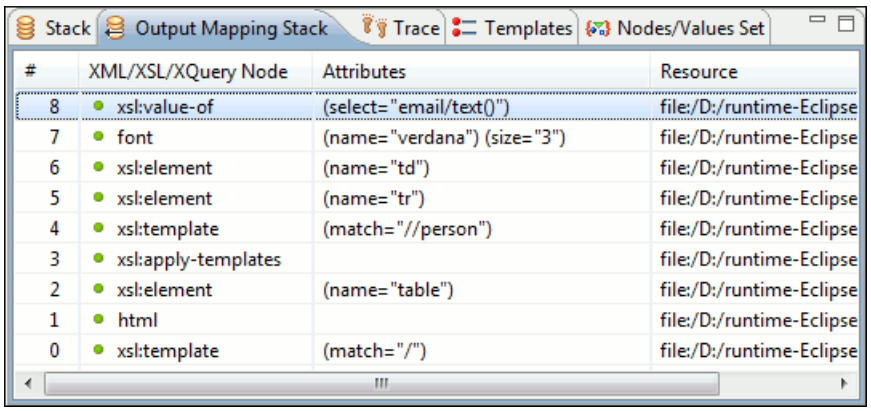
Important: Remarks:

- Clicking a record from the stack highlights that node's location inside resource.
- Using Saxon, the stylesheet elements are qualified with XSL proxy, while using Xalan you only see their names. (example: `xsl:template` using Saxon and `template` using Xalan).
- Only the Saxon processor shows element attributes.
- The Xalan processor shows also the built-in rules.

Output Mapping Stack View

The **Output Mapping Stack** view displays context data *(on page 1228)* and presents the XSLT templates/XQuery elements that generated specific areas of the output. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

Figure 357. Output Mapping Stack view



The **Go to** action of the contextual menu takes you to the line that contains the XSLT element displayed in the **Output Mapping Stack** view.

Table 39. 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 Information:

Identify the XSLT / XQuery Expression that Generated Particular Output *(on page 1228)*

Stack View *(on page 1220)*

Trace View *(on page 1222)*

Templates View *(on page 1223)*

Trace View

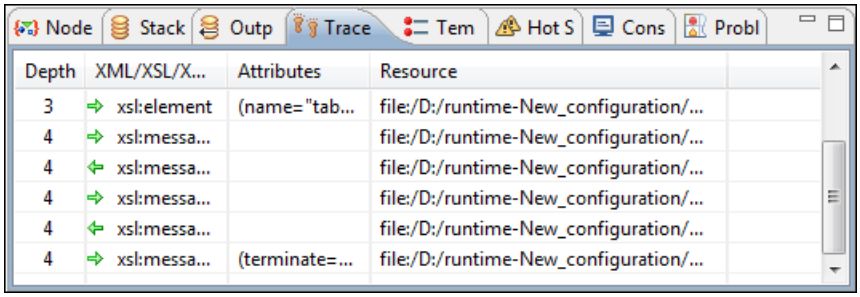
Usually, the XSLT/XQuery processors signal the following events during transformation:

- - Entering a source (XML) node.
- - Leaving a source (XML) node.
- - Entering an XSLT/XQuery node.
- - Leaving an XSLT/XQuery node.

The **Trace** view catches all of these events, so you can see how the process evolved. If the view is not displayed, it can be opened by selecting it 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 (using the **Export to XML** action in the contextual menu). Thus, you have the possibility of comparing the trace results from multiple debug sessions.

Figure 358. 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 in XML format.

Table 40. 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.

Table 40. Trace History Columns (continued)

Column	Description
XML/XSLT/XQuery Node	Represents the node from the processed source or stylesheet document. One particular node is the document root, noted as *#document*. Every node is preceded by an arrow that represents what action was performed on it (entering or leaving the node).
Attributes	Attributes of the node (a list of *id="value"* pairs).
Resource	Resource file where the node is located.

Important: Remarks:

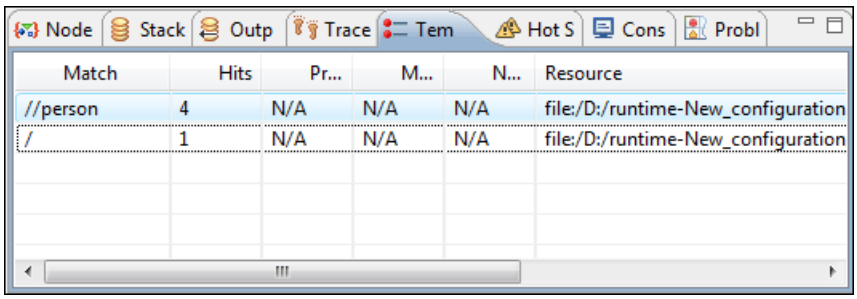
- Clicking a record highlights that node's location inside the resource.
- Only the Saxon processor shows the element attributes.
- The Xalan processor shows also the built-in rules.

Templates View

The **xsl:template** is the basic element for stylesheets transformation. 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 by selecting it 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 359. Templates view



The contextual menu contains one action: **Go to**, which moves the selection in the editor panel to the line that contains the XSLT template displayed on the selected line from the view.

Table 41. Templates columns

Column	Description
Match	The *match* attribute of the *xsl:template*.

Table 41. Templates columns (continued)

Column	Description
Hits	The number of hits for the *xsl:template*. Shows how many times the XSLT processor used this particular template.
Priority	The template priority as established by XSLT processor.
Mode	The *mode* attribute of the *xsl:template*.
Name	The *name* attribute of the *xsl:template*.
Resource	The resource file where the template is located.

Important: Remarks:

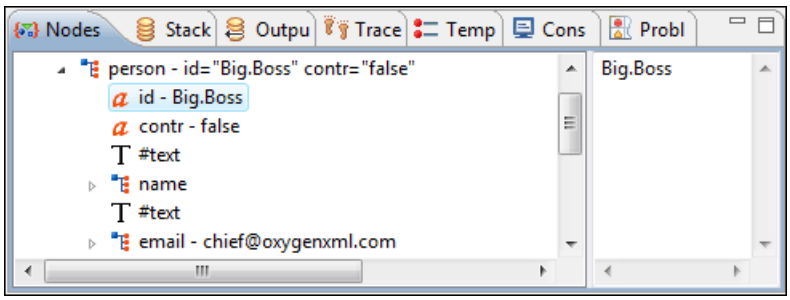
- Clicking a record highlights that template definition inside the resource.
- Saxon only shows the applied templates having at least one hit from the processor. Xalan shows all defined templates, with or without hits.
- Template table values can be sorted by clicking the corresponding column header. When clicking a column header the sorting order switches between: ascending, descending, no sort.
- Xalan shows also the built-in rules.

Nodes/Values Set View

The **Nodes/Values Set** view is always used in relation with the **Variables** view *(on page 1225)* and **XWatch** view *(on page 1218)*. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu. It shows an XSLT node set value in a tree form. This view is updated as a response to the following events:

- You click a variable that has a node set value in the **Variables** *(on page 1225)* or **XWatch** view *(on page 1218)*.
- You click a tree fragment in the **Variables** *(on page 1225)* or **XWatch** view *(on page 1218)*.
- You click an XPath expression evaluated to a node set in the **Variables** *(on page 1225)* or **XWatch** view *(on page 1218)*.

Figure 360. 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 by selecting it from the **Window > Show View** menu.

Variables and parameters play an important role during an XSLT/XQuery transformation. Oxygen XML Developer Eclipse plugin uses the following icons to differentiate variables and parameters:

- V - Global variable.
- {V} - Local variable.
- P - Global parameter.
- {P} - Local parameter.

The following value types are available:

- **Boolean**
- **String**
- **Date** - XSLT 2.0 / 3.0 only.
- **Number**
- **Set**
- **Object**
- **Fragment** - Tree fragment.
- **Any**
- **Undefined** - The value was not yet set, or it is not accessible.

Note:

When Saxon 6.5 is used, if the value is unavailable, then the following message is displayed in the **Value** field: "The variable value is unavailable".

When Saxon 9 is used:

- If the variable is not used, the **Value** field displays "The variable is declared but never used".
- If the variable value cannot be evaluated, the **Value** field displays "The variable value is unavailable".

- **Document**
- **Element**
- **Attribute**
- **ProcessingInstruction**
- **Comment**
- **Text**
- **Namespace**
- **Evaluating** - Value under evaluation.
- **Not Known** - Unknown types.

Figure 361. Variables View

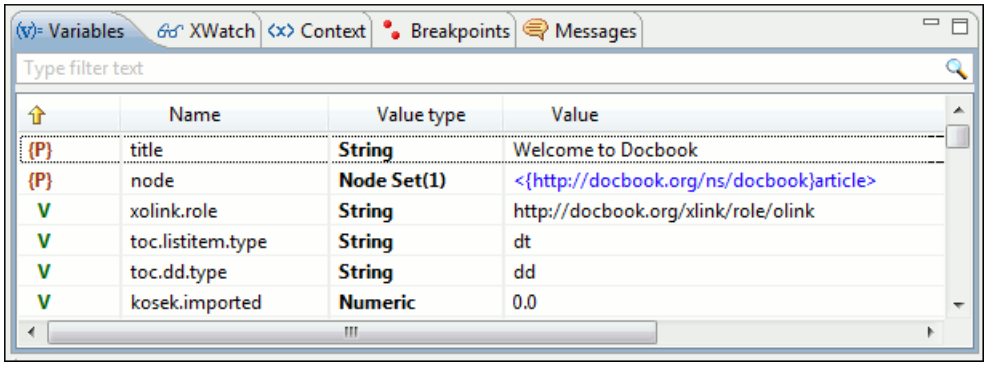


Table 42. Variables Columns

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 areas with the **Copy value** action from the contextual menu. This is useful if you have long and complex values that cannot be easily remembered just 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 *(on page 1224)* 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.

Steps in a Typical Debugging Process

Depending on your situation and needs, the debugging process might be more complex, but the following procedure is an example of a typical debugging process:

1. Open the source XML document and the XSLT/XQuery document.
2. If you are in the **Editor** *perspective (on page 1327)*, switch to the **XSLT Debugger** or **XQuery Debugger** *perspective (on page 1327)* 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 (on page 1327)* 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 *(on page 1213)*. 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 XSL/XQuery selector of the Control toolbar *(on page 1213)*.
5. Set XSLT/XQuery parameters using the **Configure parameters** button on the Control toolbar *(on page 1213)*.
6. Set one or more breakpoints *(on page 1229)*.
7. Step through the stylesheet using the following buttons available on the Control toolbar *(on page 1214)*:
    - **Step into**
    - **Step over**
    - **Step out**
    - **Run**
    - **Run to cursor**

- **Run to end**
- **Pause**
- **Stop**

8. Examine the data in the information views to find the bug in the transformation process.
   For more information about fixing bugs in the transformation, see: Identify the XSLT / XQuery Expression that Generated Particular Output *(on page 1228)*.

Related Information:

Identify the XSLT / XQuery Expression that Generated Particular Output *(on page 1228)*

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.

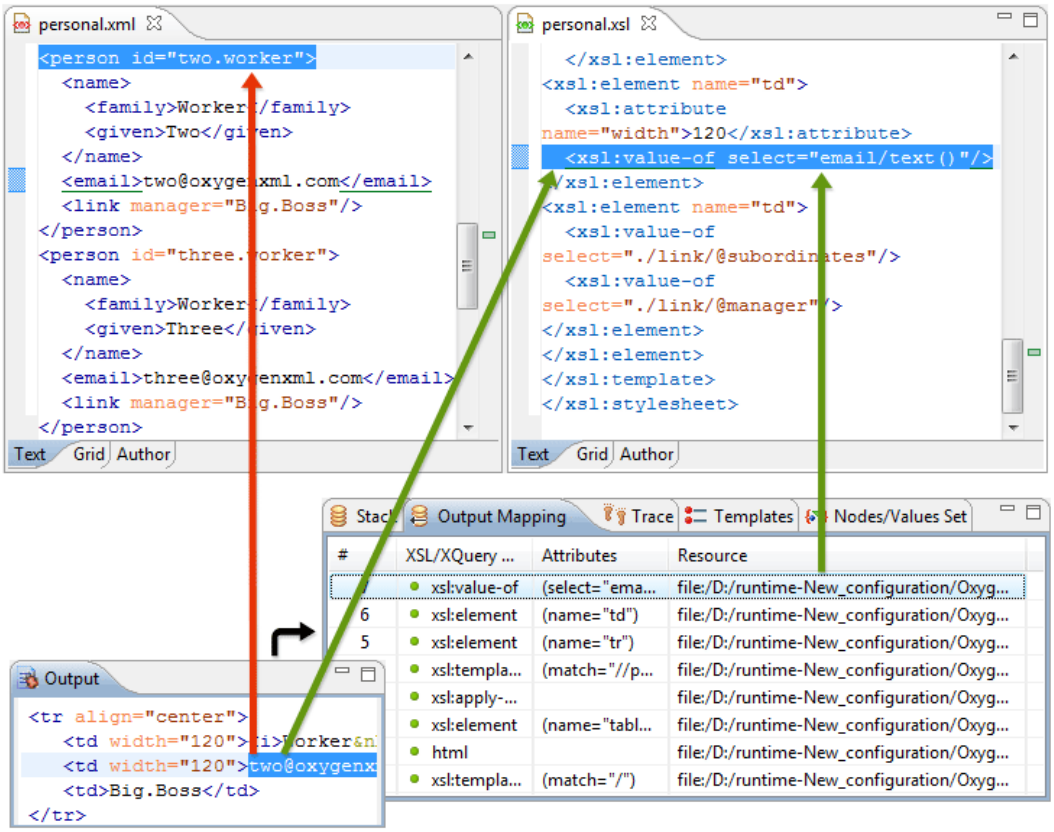
You can click particular text in the **Output** view and the editor will select the XML source context and the XSLT template/XQuery element that generated that text. Also, inspecting the whole stack of XSLT templates/XQuery elements that determined the state of the XSLT/XQuery processor at the moment of generating the specified output area speeds up the debugging process.

This is an example of a typical procedure for identifying an expression that generated particular output:

1. Switch to the **XSLT Debugger** or **XQuery Debugger** *perspective (on page 1327)* 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 (on page 1327)* with the parameters of the transformation scenario. Any modification applied to the scenario parameters (the transformer engine, XSLT parameters, transformer extensions, etc.) will be saved back in the scenario when exiting from the Debugger *perspective*.
2. Select the source XML document in the XML source selector of the Control toolbar *(on page 1213)*. In the case of XQuery debugging, if your XQuery document has no implicit source, set the source selector value to **NONE**.
3. Select the XSLT/XQuery document in the XSL/XQuery selector of the Control toolbar *(on page 1213)*.
4. Select the appropriate engine in the XSLT/XQuery engine selector of the Control toolbar *(on page 1214)*.

5. Set XSLT/XQuery parameters using the **Configure parameters** button on the Control toolbar *(on page 1213)*.
6. Apply the XSLT stylesheet or XQuery transformation using the **Run to end** button that is available on the Control toolbar *(on page 1214)*.
7. Inspect the mapping by clicking a section of the output in the **Output** view.

Figure 362. 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 *(on page 1220)*.

Related Information:

- Output Mapping Stack View *(on page 1220)*
- Trace View *(on page 1222)*
- Templates View *(on page 1223)*

Using Breakpoints

The Oxygen XML Developer Eclipse plugin XSLT/XQuery Debugger allows you to interrupt XSLT/XQuery processing to gather information about variables and processor execution at particular points. To ensure *breakpoints* are persistent between work sessions, they are saved at project level. You can set a maximum of 100 *breakpoints* per project.

Inserting Breakpoints

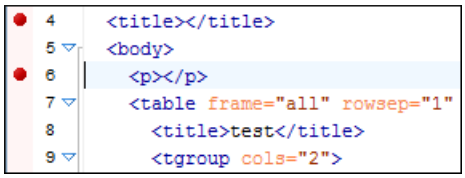
To insert a *breakpoint*, follow these steps:

1. Click the line where you want to insert the *breakpoint* in the XML source document or the XSLT/XQuery document. *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**.

Result:

Once you insert a *breakpoint*, it is automatically added to the list in the **Breakpoints** view and you can edit its associated *condition*. A *breakpoint* can have an associated break condition that represents an XPath expression evaluated in the current debugger context. For them 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 363. Example: Breakpoints



Tip: You can configure the color and how *breakpoints* are shown from the Eclipse **Annotations** preferences page (**Window ('Eclipse' on Mac OSX) > Preferences > General > Editors > Text Editors > Annotations**).

Removing Breakpoints

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**.

Related Information:

Breakpoints View *(on page 1216)*

Performance Profiling of XSLT Stylesheets and XQuery Documents

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 processor that can be used for debugging and it is available from the debugging *perspective (on page 1327)*.

Enabling the Profiler

Enabling and disabling the profiler is controlled by the / **Profiler** button from the debugger Control toolbar *(on page 1213)*. The XSLT/XQuery profiler is off by default. This option is not available during a debugger session so you need to set it before starting the transformation. For information about a common debugging procedure, see Steps in a Typical Debugging Process *(on page 1227)*.

Profiling Information Views

Immediately after enabling the profiler, two new information views are added to the current debugger information views *(on page 1215)*:

- **Invocation tree** view *(on page 1232)* on left side
- **Hotspots** view *(on page 1233)* on right side

Profiling data is available only after the transformation ends successfully.

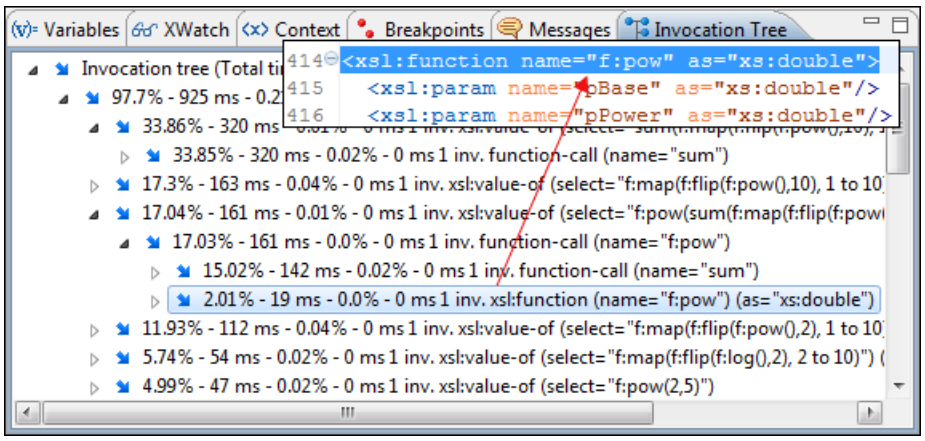
On the left side (**Invocation tree** view *(on page 1232)*), you can examine how style instructions are processed. This result view is also named call-tree, as it represents the order of style processing. The profiling result shows the duration time for each of the style-instruction including the time needed for its called children.

On the right side (**Hotspots** view *(on page 1233)*), 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).

Source Backmapping

In either the **Invocation tree** *(on page 1232)* or **Hotspots** view *(on page 1233)*, you can use the backmapping feature to find the XSLT stylesheet or XQuery expression definition. Clicking the selected item causes Oxygen XML Developer Eclipse plugin to highlight the XSLT stylesheet or XQuery expression source line where the instruction is defined.

Figure 364. Source Backmapping



Saving and Customizing Profiling Data

The profiling data can be saved (exported) into XML and HTML format. In either the **Invocation tree** *(on page 1232)* or **Hotspots** view *(on page 1233)*, right-click anywhere in the view and select **Export to XML** or **Export to HTML**. The HTML report can be customized based upon the profiling raw data. When you select **Export to HTML**, Oxygen XML Developer Eclipse plugin will save it as XML and apply an XSLT stylesheet to render the report as XML. You can customize these stylesheets to suit your needs. By default, they are located in: *[OXYGEN_INSTALL_DIR]*`/frameworks/profiler/`.

Other Profiling Notes

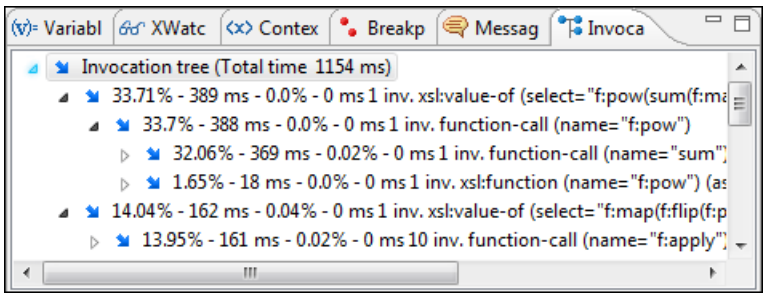
- If you want to change the XSLT/XQuery profiler settings *(on page 132)*, use the contextual menu and choose the corresponding **View settings** entry.
- Profiling exhaustive transformations may run into an **OutOfMemory** error due to the large amount of information being collected. If this is the 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.
- For more information about the XSLT/XQuery Profiler, watch our video demonstration:

  https://www.youtube.com/embed/4ftHschjLqA

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 by selecting it from the **Window > Show View** menu.

Figure 365. 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** *(on page 132)*:

- 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.

The **Invocation Tree** view also includes the following contextual menu actions:

Export to HTML

Selecting this option will save the profiling data as XML and then apply an XSLT stylesheet to render the report as HTML. These stylesheets are included in the subfolder: `[OXYGEN_INSTALL_DIR]/frameworks/profiler/`. You can use them to customize your own report based on the profiling raw data.

Export to XML

Use this option to save the profiling data as an XML file in a specified location.

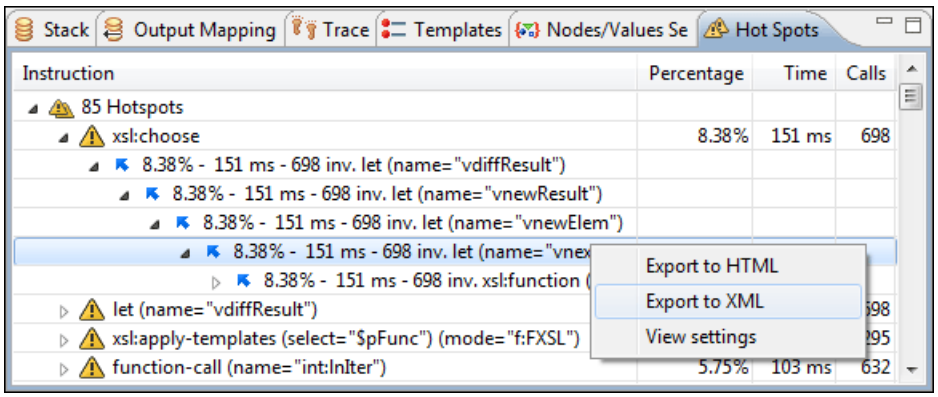
View settings

Opens the XSLT/XQuery Profiler preferences page *(on page 132)* that allows you to configure various profiling settings.

Hotspots View

The **Hotspots** view displays a list of all instruction calls that lie above the threshold defined in the **XSLT/XQuery profiler settings** *(on page 132)*. If the view is not displayed, it can be opened by selecting it from the **Window > Show View** menu.

Figure 366. 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** *(on page 132)*:

- 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.

The **Hotspots** view also includes the following contextual menu actions:

Export to HTML

Selecting this option will save the profiling data as XML and then apply an XSLT stylesheet to render the report as HTML. These stylesheets are included in the subfolder:

`[OXYGEN_INSTALL_DIR]/frameworks/profiler/`. You can use them to customize your own report based on the profiling raw data.

Export to XML

Use this option to save the profiling data as an XML file in a specified location.

View settings

Opens the XSLT/XQuery Profiler preferences page *(on page 132)* that allows you to configure various profiling settings.

Debugging XSLT that Call Java Extensions

It is possible to debug an XSLT that calls Java extensions. This is achieved through a transformation scenario where the Java extensions are specified, and the debugging can be done based upon the same scenario.

To debug XSLT with Java extensions, follow this procedure:

1. Create an XSLT transformation scenario *(on page 732)* for your XSLT document (select **Configure Transformation Scenario(s)** action from the toolbar, then click **New**, and select **XSLT transformation**).
2. In the **New scenario** dialog box, click the **Extensions** button (in the **XSLT** tab), specify the Java extensions (JAR libraries) that are needed, and click **OK**.
3. Once you are finished configuring the transformation scenario, click **OK**, then select **Save and close**.
4. Use the **Debug scenario** action on the toolbar and the debugging will be based upon the same transformation scenario you just configured and saved.

Tip: You could achieve this during a typical debugging process *(on page 1227)* by specifying the Java extensions using the **Edit extensions** button on the debugger control toolbar *(on page 1213)*.

Related Information:

Validating XSLT Stylesheets that Call Java Extensions *(on page 366)*

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 (on page 1229)* 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 1024 MB) by setting the `-Xmx` parameter in the debug configuration (for example, *"-Xmx1024m"*).

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 *(on page 1213)* of the XSLT/XQuery transformation in the debugging *perspective (on page 1327)*.

c. Set the class *ro.sync.exml.Oxygen* as the main Java class of the configuration.
The main Java class *ro.sync.exml.Oxygen* is located in the `oxygen.jar` file.

2. Start the debug configuration.
Now you can set *breakpoints* and inspect Java variables as in any Java debugging process executed in the selected IDE (Eclipse SDK, and so on.).

Supported Processors for XSLT / XQuery Debugging

The following built-in XSLT processors are integrated in the debugger and can be selected in the Control Toolbar *(on page 1212)*:

- **Saxon 9.9.1.5 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 9.9.1.5 PE (Professional Edition)** - capable of running XSLT 1.0 transformations, XSLT 2.0 basic ones and XQuery 1.0 ones, available in both the XSLT debugger and the XQuery one,
- **Saxon 9.9.1.5 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 2.7.2** - capable of running only XSLT 1.0 transformations, available only in the XSLT debugger.

16.

Extension Points for the Oxygen Eclipse Plugin

The Oxygen XML Developer Eclipse plugin includes a number of *extension (on page 1327)* points, which can be implemented by other Eclipse *plugins (on page 1327)* 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*

Contributes a folder that contains the additional XML Refactoring operation descriptor files and XQuery scripts that can be used by the batch XML refactoring actions. Its `EXSD` schema can be found in: `OXYGEN_PLUGIN_DIR/exsd-schema/xmlRefactoringContributor.exsd`.

Extension point: *workspaceAccessPlugin*

Use this extension point to be notified when Oxygen XML Developer Eclipse plugin has started.
Its EXSD schema can be found in: `OXYGEN_PLUGIN_DIR/exsd-schema/exsd-schema/workspaceAccessExtension.exsd`.

17.
Tools

Oxygen XML Developer Eclipse 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 Eclipse 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 *(on page 192)*.

Note: The built-in 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 Eclipse 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.

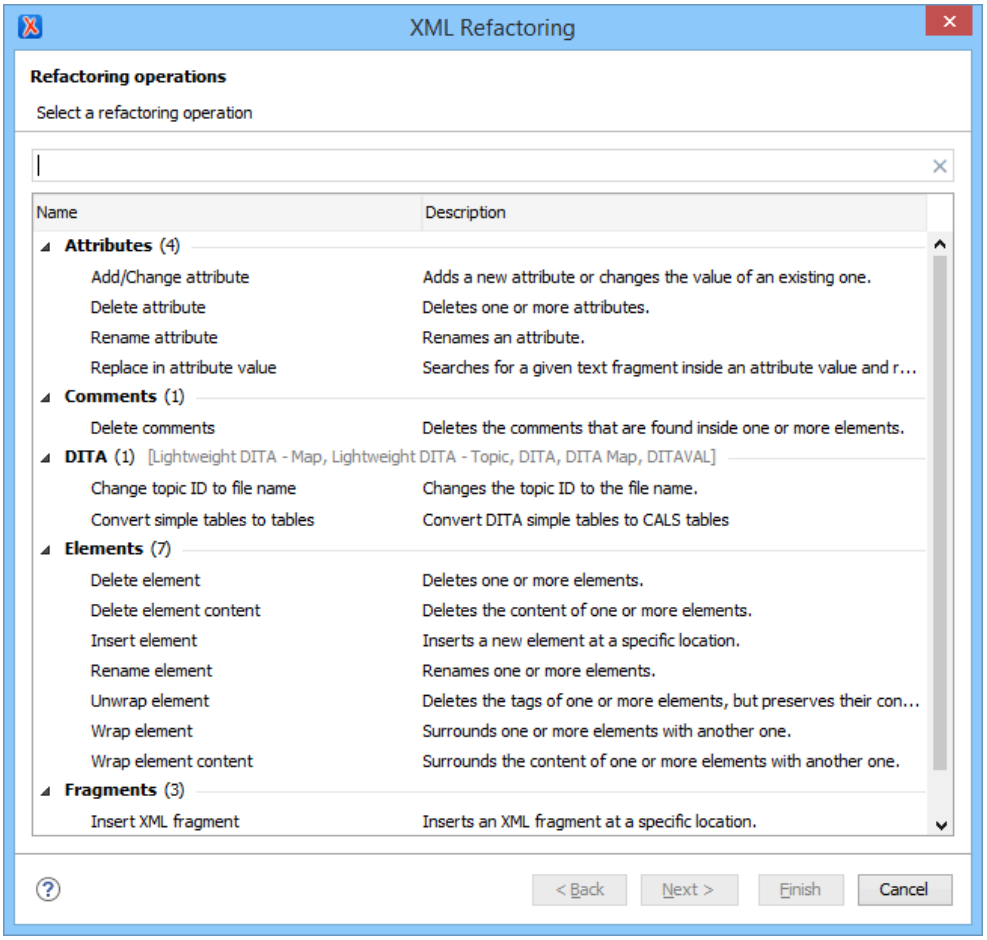
XML Refactoring Wizard

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.

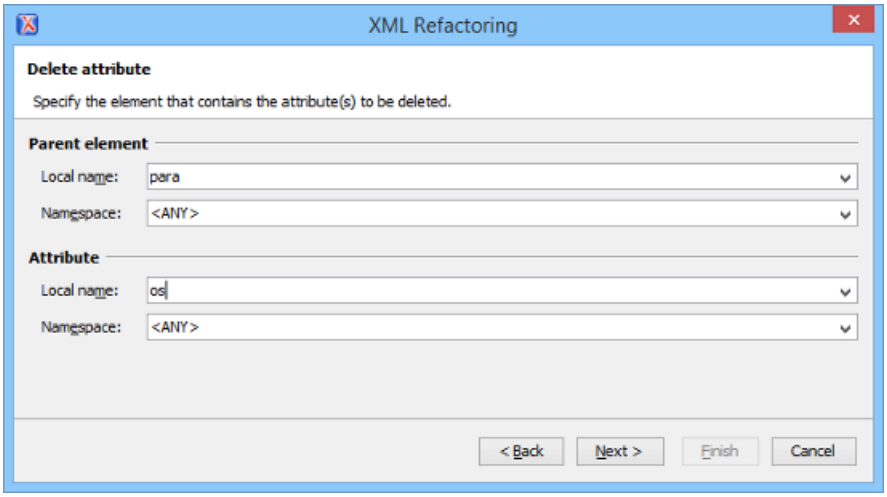
Figure 367. XML Refactoring Wizard



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.

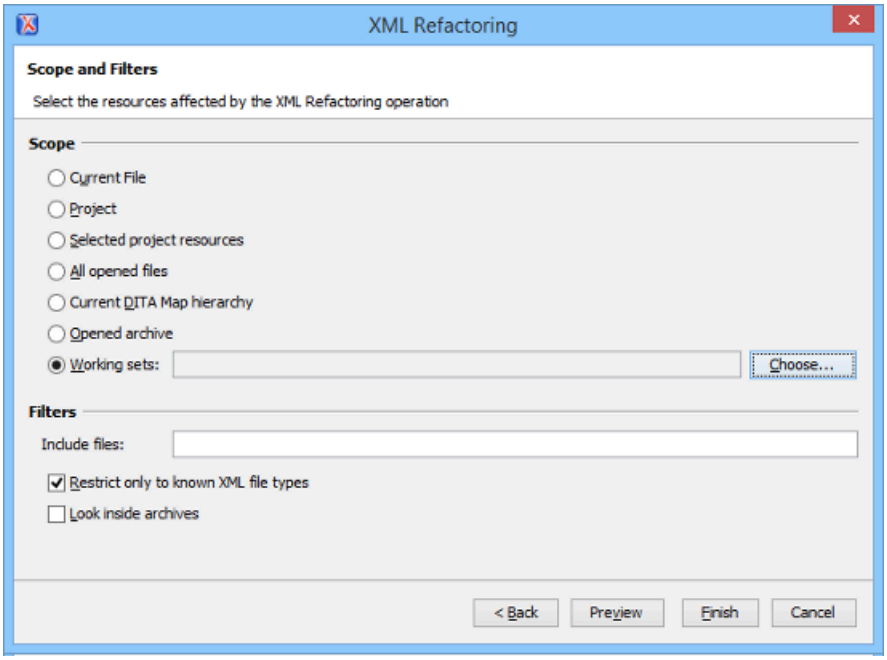
Figure 368. XML Refactoring 2nd Wizard Page (Delete Attribute Operation)



Scope and Filters

The last wizard page allows you to select the set of files that represent the input of the operation.

Figure 369. XML Refactoring - Scope and Filters Wizard Page



Scope section

In the **Scope** section, you can select from predefined resource sets (such as the current file, your whole project, the current *DITA map (on page 1324)* hierarchy for DITA projects, etc.) or you can define your own set of resources by creating a *working set (on page 1329)*.

Filters

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 selected, only resources with a known XML file type will be affected by the operation.

Preview

You can use the **Preview** button to open a comparison panel where you can review all the changes that will be made by the refactoring operation before applying the changes.

Finish

After clicking the **Finish** button, the operation will be processed and Oxygen XML Developer Eclipse plugin provides no automatic means for reverting the operations. Any **Undo** action will only revert changes on the current document.

Troubleshooting: If an operation fails, a notification will be displayed in the **Results** panel with some information about the error. For example, if the operation was invoked on a read-only resource, the error will indicate that a read-only file cannot be converted.

Tip: If an operation takes longer than expected you can use the **Stop** button in the progress bar to cancel the operation.

Restriction: XML refactoring operations cannot preserve CDATA sections. If your document contains XML CDATA sections, the refactoring operations will convert them to plain text nodes.

Built-in Refactoring Operations

The XML Refactoring tool includes a variety of built-in 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 Eclipse 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 built-in 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

Adds the attribute to all instances of the specified parent element.

Change the value in the parent elements where the attribute already exists

Replaces the value of the already existing attribute in all instance of the specified parent element.

Both

Adds the attributes to the instances where it is missing and replaces the value in instances where the attribute already exists.

Convert attribute to element

Use this operation to convert a specified attribute to an element. This operation allows you to specify the following parameters:

Parent element section

Element

The parent element of the attribute to be converted, 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.

New element section

Local name

The local name of the new element.

Namespace

The namespace of the new element.

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) that will have comments 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.

Convert CALS tables to simple tables

Use this operation to convert DITA CALS tables to simple tables.

Convert conrefs to conkeyrefs

Use this operation to convert `@conref` attributes to `@conkeyref` attributes.

Convert simple tables to CALS tables

Use this operation to convert DITA simple tables to CALS tables.

Convert Nested Topics to New Topics

Use this operation on topics that contain nested `<topic>` elements to convert each nested topic to a new topic.

Convert Sections to New Topics

Use this operation on topics that contain multiple sections to convert each section to a new topic.

Convert to Concept

Use this operation to convert a DITA topic (of any type) to a DITA Concept topic type (for example, Topic to Concept).

Convert to General Task

Use this operation to convert a DITA topic (of any type) to a DITA General Task topic type (for example, Task to General Task).

Convert to Reference

Use this operation to convert a DITA topic (of any type) to a DITA Reference topic type (for example, Topic to Reference).

Convert to Task

Use this operation to convert a DITA topic (of any type) to a DITA Task topic type (for example, Topic to Task).

Convert to Topic

Use this operation to convert a DITA topic (of any type) to a DITA Topic (for example, Task to Topic).

Convert to Troubleshooting

Use this operation to convert a DITA topic (of any type) to a DITA Troubleshooting topic type (for example, Topic to Troubleshooting).

Rename Key

Use this operation to rename a key. It also updates all references to it. Note that it does not work on DITA 1.3 key scopes.

All of these DITA refactoring actions allow you to choose a scope for the operation and some filters:

Scope

Select from a variety of options to define the scope that will have resources affected by the operation. For example, you can choose to affect all resources in the **Project**, **All opened files**, **Current DITA map hierarchy**, 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.

Add Blue DOCTYPE - NISO JATS Publishing 1.1

Use this operation to add a JATS 'Blue' 1.1 DOCTYPE declaration.

Normalize IDs

Use this operation to normalize assigned IDs and assigned IDs to elements that are missing them.

All of these JATSKit refactoring actions allow you to choose a scope for the operation and some filters:

Scope

Select from a variety of options to define the scope for the resources that 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 *Publishing Template*

These operations are for those who use *Oxygen Publishing Templates* for WebHelp Responsive output customization.

Migrate HTML Page Layout Files to v21

Use this operation to convert custom HTML page layout files *(on page 830)* that are included in a custom Publishing Template that was created in Oxygen XML Developer Eclipse plugin version 20.0 or 20.1 so that they will be compatible with Oxygen XML Developer Eclipse plugin version 21.0.

Update HTML Pages

⚠ **Attention:** This operation is only used by Oxygen XML Developer Eclipse plugin and should not be used manually.

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** *(on page 133)* 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

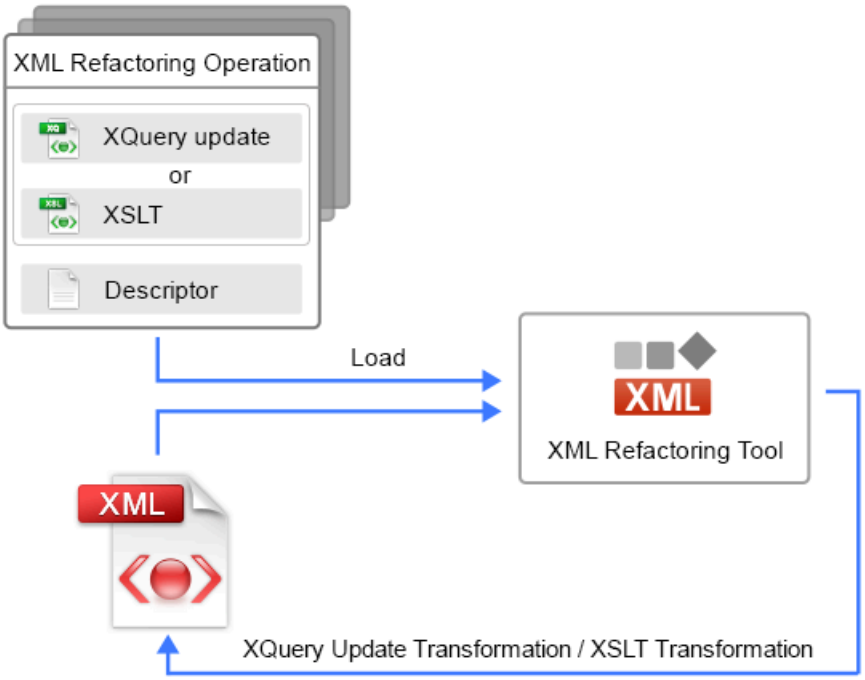
While Oxygen XML Developer Eclipse plugin includes a variety of built-in XML refactoring operations to help you accomplish particular tasks, you can also create custom operations according to your specific needs. For example, you could create a custom refactoring operation to convert an attribute to 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 Eclipse 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 370. 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 *(on page 326)*, along with the built-in operations.

After the user chooses an operation and specifies its parameters, Oxygen XML Developer Eclipse 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.

    Note: This can be changed using Saxon extension functions in XSLT *(on page 352)*.
  - 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 applied on the root element and the nodes inside it. However, as a work around to this limitation, you can use Saxon extension functions and the XSLT stylesheet method *(on page 352)* to implement the new custom XML refactoring operation.

Creating a Custom Refactoring Operation

To create a custom refactoring operation, follow these steps:

1. Create an XQuery Update script *(on page 344)* or XSLT stylesheet file *(on page 348)*.
2. Create an **XML Refactoring Operation Descriptor** file contains the path to the XQuery Update script *(on page 346)* or XSLT stylesheet *(on page 350)*.
3. Store both files in one of the locations that Oxygen XML Developer Eclipse plugin *(on page 354)* 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 *(on page 326)*.

Related Information:

Storing and Sharing Refactoring Operations *(on page 354)*

Custom Refactoring Script

The first step in creating a custom refactoring operation is to create an XQuery Update script *(on page 344)* or XSLT stylesheet *(on page 348)* 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 the XQuery method example *(on page 344)* or XSLT method example *(on page 348)* to help you with the content.

There are cases when it is necessary to add parameters in the XQuery script *(on page 344)* or XSLT stylesheet *(on page 348)*. 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 Eclipse plugin. Thus, if you want to create or edit an XQuery Update script you have to enable this mechanism by creating an XQuery transformation scenario *(on page 754)* and choose **Saxon EE** as the transformation engine. Also, you

need to make sure the **Enable XQuery update** option is selected in the Saxon processor advanced options *(on page 711)*.

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 *(on page 133)*.

The next step in creating a custom refactoring operation is to create an **XML Refactoring Operation Descriptor** file contains the path to the XQuery Update script *(on page 346)* or XSLT stylesheet *(on page 350)*.

Related Information:

XQuery Update Script for Creating a Custom Operation *(on page 344)*

XSLT Stylesheet for Creating a Custom Operation *(on page 348)*

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 *(on page 344)* or XSLT stylesheet *(on page 348)* 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 *(on page 327)* 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 *(on page 327)*.

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 Eclipse 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** *(on page 133)* 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 Eclipse 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 (on page 1328)* 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 Eclipse 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 Eclipse 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" useCurrentContext="true">
    <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>
```

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" useCurrentContext="true">
    <localName label="Name" name="attribute_localName">
        <description>The name of the attribute to be converted.</description>
    </localName>
    <namespace label="Namespace" name="attribute_namespace" allowsAny="true">
        <description>Namespace of the attribute to be converted.</description>
    </namespace>
</attributeParameter>
```

Note: All built-in operations are loaded from the *`[OXYGEN_INSTALL_DIR]/refactoring`* folder.

Related Information:

Example of an Operation Descriptor File with an XSLT Stylesheet *(on page 350)*

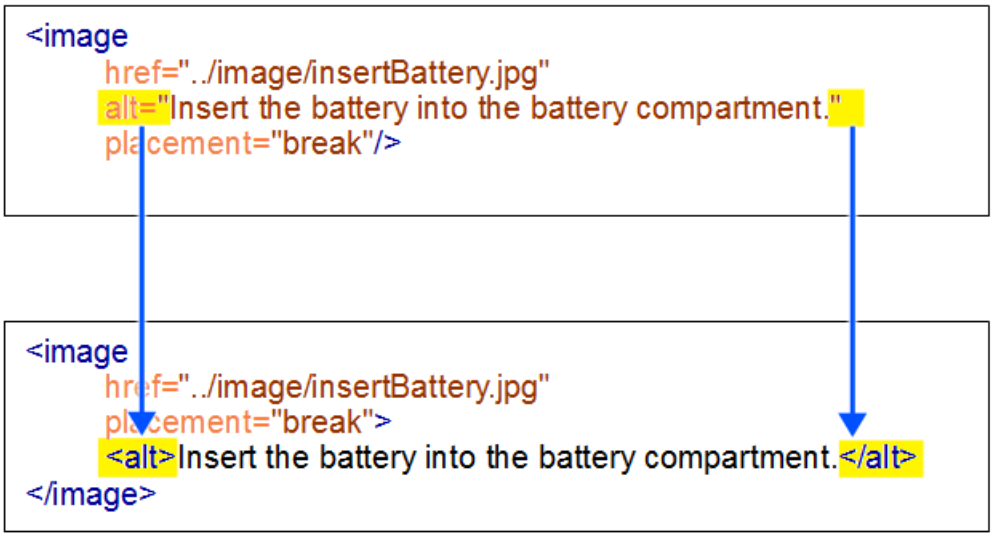
Example of an Operation Descriptor File with an XQuery Update script *(on page 346)*

XSLT Stylesheet for Creating a Custom Operation

To demonstrate creating a custom operation, suppose that you have a task where you 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, the 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 371. Example: Custom XML Refactoring Operation



An XSLT stylesheet can be used to implement the new custom XML refactoring operation. The second requirement is an XML Refactoring operation descriptor file *(on page 350)* that contains the path to the XSLT stylesheet.

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 (on page 1329)* set in the XML Refactoring *framework (on page 1325)*.

Example of an Operation Descriptor File That References the XSLT Stylesheet for Creating a Custom Operation to *Convert an Attribute to an Element*

After you have developed the 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"/>
```

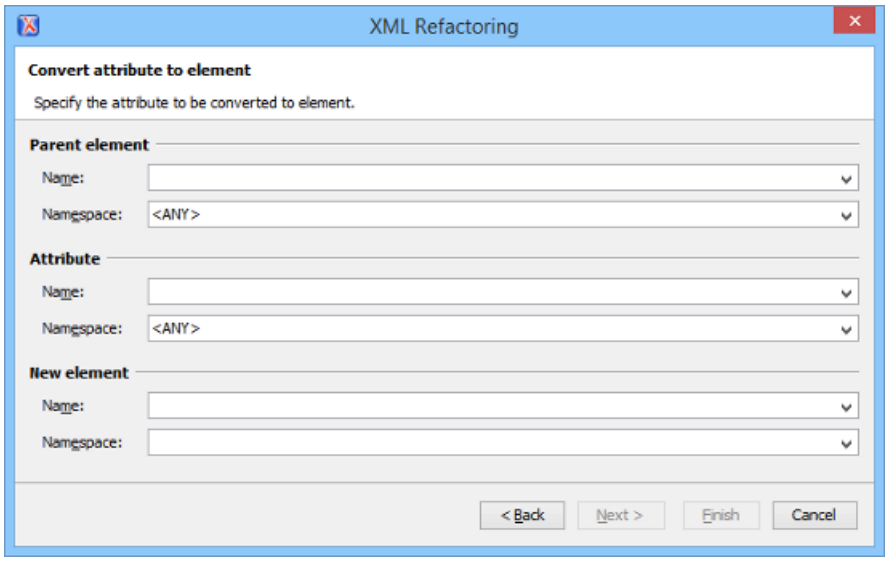
The code exemplified above and other refactoring examples can be found on the DITA Refactoring GitHub sample project.

Results

After you have created these files, copy them into a folder scanned by Oxygen XML Developer Eclipse plugin when it loads the custom operation *(on page 354)*. 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 the example of the custom operation to convert an attribute to an element:

Figure 372. Example: XML Refactoring Wizard for a Custom Operation



Using Saxon Extension Functions to Allow Custom Refactoring Operations to Read and Modify Content Outside the Root Node

One advantage to using an XSLT stylesheet is that there is limitation when using an XQuery Update script *(on page 344)* in that refactoring operations can only be performed on *comments* or *processing instructions* that are inside the root element. Thus, using the XQuery method, *comments* or *processing instructions* that are in any node before or after the root element cannot be modified by an XML Refactoring operation.

The XSLT stylesheet method offers a work-around to this limitation through the use of some Saxon extension functions.

To illustrate the use of these functions, consider the following sample XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- comment before root -->
<?pi before root ?>
<root>
    <child></child>
</root>
<!-- comment after root -->
<?pi after root ?>
```

The following Saxon extension functions can be used to read and modify content outside the root node:

Note: They belong to the `http://www.oxygenxml.com/ns/xmlRefactoring/functions` namespace.

- **get-content-after-root()** - Returns the content after root as `xs:string`.

  For the XML above, the call of this function will return the following string value:

  ```
  <!-- comment after root -->
  <?pi after root ?>
  ```

- **set-content-after-root(xs:string)** - Updates the content that will be serialized in the refactored document after the root node.

  The function call `set-content-after-root('<!-- Inserted comment -->')` will result in replacing the nodes after the root element with the comment passed as string argument. The XML document will be modified as follows:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <!-- comment before root -->
  <?pi before root ?>
  <root>
      <child></child>
  </root><!-- Inserted comment -->
  ```

- **get-content-before-root()** - Returns the content before root as `xs:string`.

  For the XML above, the call of this function will return the following string value:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <!-- comment before root -->
  <?pi before root ?>
  ```

- **set-content-before-root(xs:string)** - Updates the content that will be serialized in the refactored document after the root node.

  The function call `set-content-before-root('<!-- Inserted comment -->')` will result in replacing the nodes before the root element with the comment passed as string argument. The XML document will be modified as follows:

  ```
  <!-- Inserted comment --><root>
      <child></child>
  </root>
  <!-- comment after root -->
  <?pi after root ?>
  ```

XSLT Example:

To process content after the root node, the XSLT would look like this:

```
<?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:xrf="http://www.oxygenxml.com/ns/xmlRefactoring/functions" version="3.0">
```

```
    <xsl:template match="/">
        <!-- The comment content that will be inserted after the root element -->
        <xsl:variable name="commentAsText"><!-- COMMENT ADDED FROM XR OPERATION-->
        </xsl:variable>
        <!-- Retrieve the content after the root element as is -->
        <xsl:variable name="after-root-content" as="xs:string"
                      select="xrf:get-content-after-root()"/>
        <xsl:variable name="processedContent"
                      select="concat($after-root-content, $commentAsText)"/>

        <!-- Update the content after the root element -->
        <xsl:value-of select="xrf:set-content-after-root($processedContent)"/>
        <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="node() | @*">
        <xsl:copy>
            <xsl:apply-templates select="node() | @*"/>
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

Note: The above XSLT retrieves the nodes after the root element as string, appends a new comment, and then sets back the updated content into the XML document.

Storing and Sharing Refactoring Operations

Oxygen XML Developer Eclipse 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. To associate the parent directory to the *framework*, you need to add it in the **Classpath** tab of the **Document type** configuration dialog box *(on page 55)*.
- A folder that you specify in the **Load additional refactoring operations from** text box *(on page 127)* in the **XML Refactoring** preferences page *(on page 127)*.
- The `refactoring` folder from the Oxygen XML Developer Eclipse plugin installation directory (*`[OXYGEN_INSTALL_DIR]`*`/refactoring/`).

Sharing Custom Refactoring Operations

The purpose of Oxygen XML Developer Eclipse 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 *framework (on page 1325)* or projects.

After storing custom operations, you can share them with other users by sharing the resources.

Localizing XML Refactoring Operations

Oxygen XML Developer Eclipse 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 `possibleValues` for each `parameter`.

Translated refactoring information uses the following form:

```
${i18n(translation_key)}
```

Oxygen XML Developer Eclipse 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 *(on page 36)*, 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 Eclipse plugin installation directory (*[OXYGEN_INSTALL_DIR]/refactoring/i18n*).

Example: 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 Eclipse 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 *(on page 126)*.

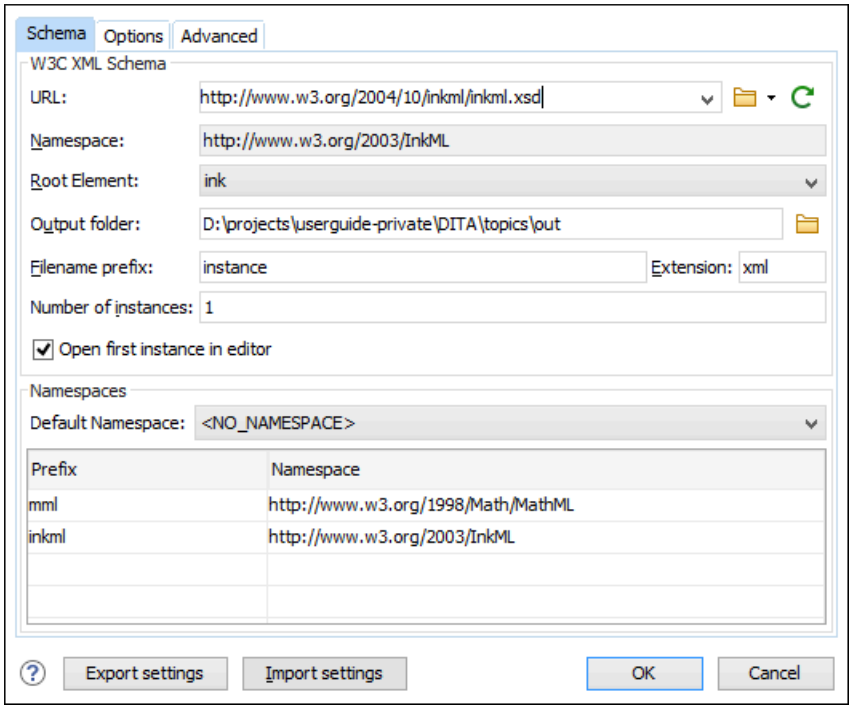
To generate sample XML files from an XML Schema, use the **Generate Sample XML Files** action from the **XML Tools** menu. This action is also available in the contextual menu of the schema Design mode *(on page 410)*. The action opens the **Generate Sample XML Files** dialog box that allows you to configure a variety of options for generating the files.

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 *(on page 119)*.

Schema Tab

The first set of options for the **Generate Sample XML Files** tool are found in the **Schema** tab.

Figure 373. 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 actions 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 selected, 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.

Export settings

Use this button to save the current settings for future use.

Import settings

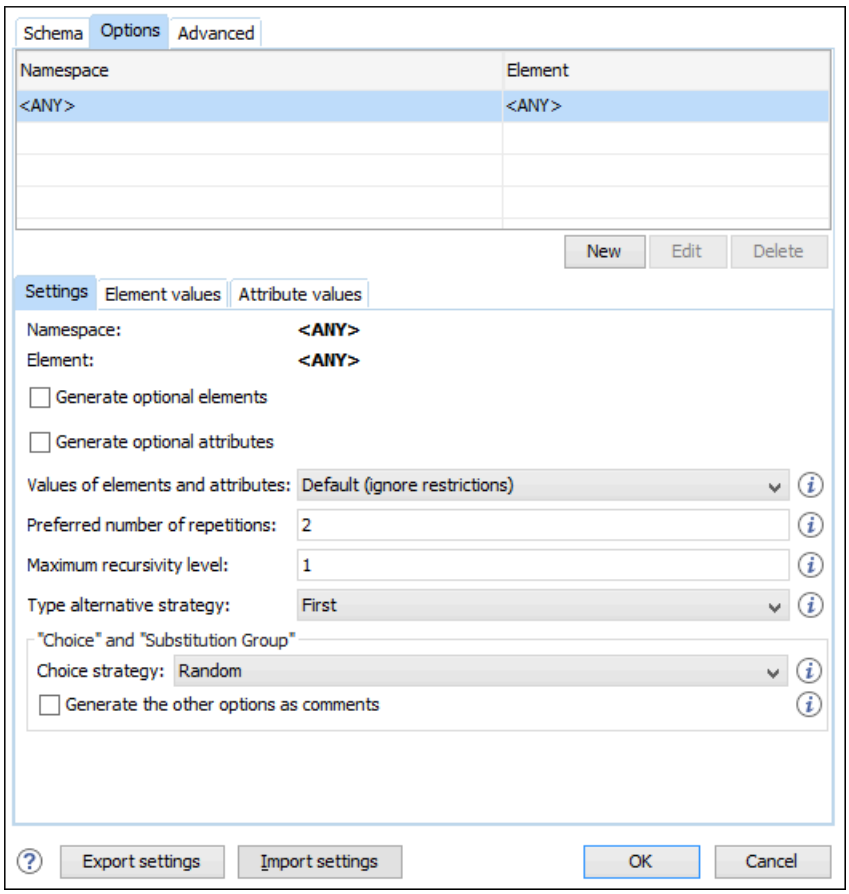
Use this button to load previously exported settings.

You can click **OK** at any point to generate the sample XML files.

Options Tab

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

Figure 374. 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 selected, all elements are generated, including the optional ones (having the `minOccurs` attribute set to 0 in the schema).

Generate optional attributes

When selected, 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 on the 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 **Sample XML Files Generator** preferences page). Note that type restrictions are ignored when this option is selected. 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 on the data type descriptor of the particular element or attribute.

  Important: If all of the following are true, the **Generate Sample XML Files** tool 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 selected, 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.

Export settings

Use this button to save the current settings for future use.

Import settings

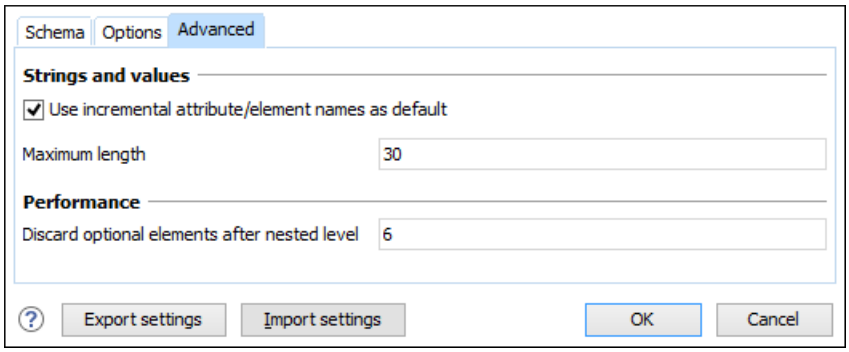
Use this button to load previously exported settings.

You can click **OK** at any point to generate the sample XML files.

Advanced Tab

The **Advanced** tab allows you to set some options regarding output values and performance.

Figure 375. Generate Sample XML Files Dialog Box (Advanced Tab)



This tab includes the following options:

Use incremental attribute / element names as default

If selected, 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 selected, 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.

Export settings

Use this button to save the current settings for future use.

Import settings

Use this button to load previously exported settings.

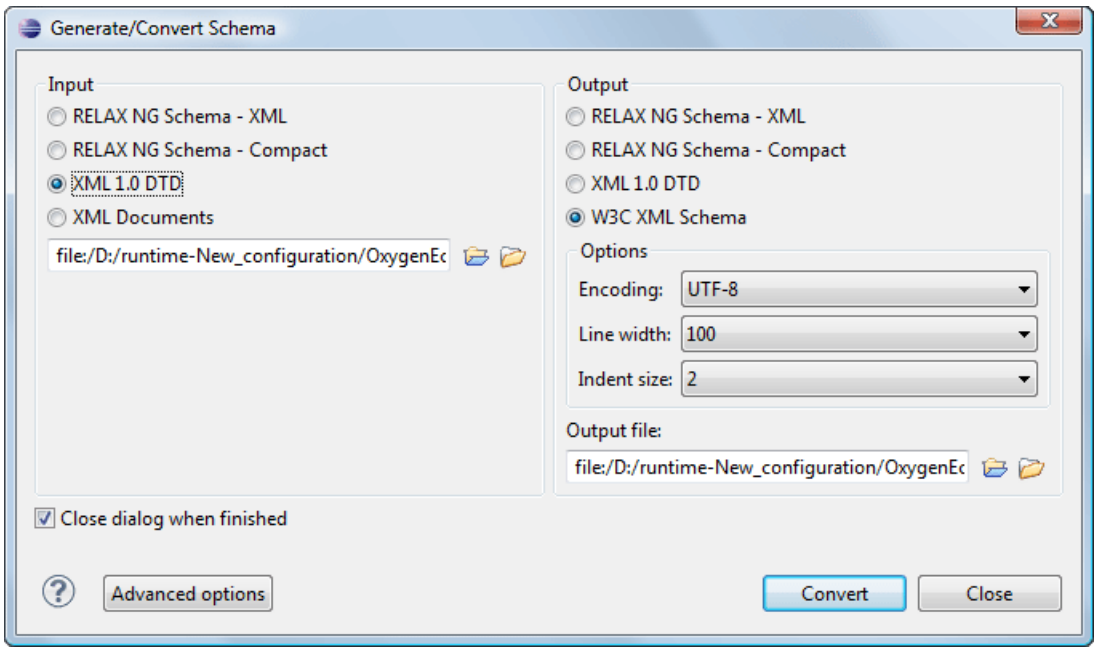
Tip: This function can be executed from an automated command-line script, but it requires an additional **Oxygen XML Scripting License**. For more details, see https://www.oxygenxml.com/oxygen_scripting.html.

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 Eclipse plugin generates an approximation of the source schema. Oxygen XML Developer Eclipse plugin uses the *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 376. 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 open 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 a `@prefix:defaultValue` annotation attribute 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 Eclipse 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 *(on page 40)* 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.

Compiling an XSL Stylesheet for Saxon

As of Saxon 9.7, it is possible to export a compiled form of a stylesheet as an XML file (called a *stylesheet export file* or SEF). Oxygen XML Developer Eclipse plugin includes a simple tool called **Compile XSL Stylesheet for Saxon** (found in the **XML Tools** menu) that does this for you.

Use-Cases for a Stylesheet Export File (SEF)

- **Use Saxon-JS to run transformations in a browser** - A *stylesheet export file* (SEF) is needed if you want to use the Saxon-JS product to run transformations in a browser, as in the following example:

```
<script type="text/javascript" src="SaxonJS/SaxonJS.min.js"></script>
<script>
    window.onload = function() {
      SaxonJS.transform({
         stylesheetLocation: "books.sef",
         sourceLocation: "books.xml"
         });
    }
</script>
```

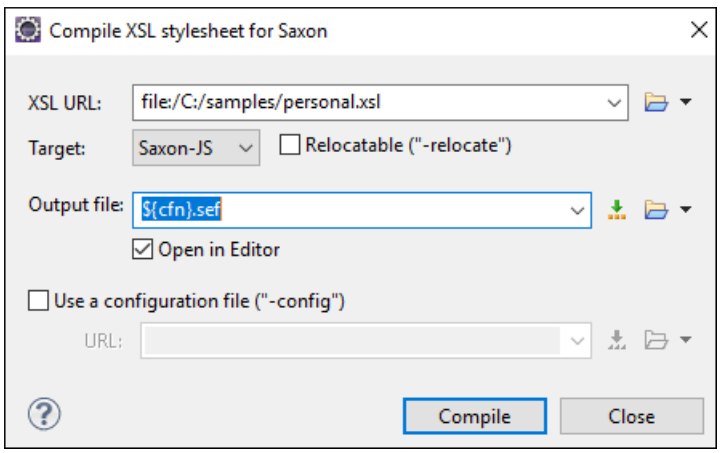
- **Use SEF to run transformations in Oxygen XML Developer Eclipse plugin** - You can also use a *stylesheet export file* (SEF) in Oxygen XML Developer Eclipse plugin to apply an XSLT transformation over an XML file. This requires **Saxon-EE** or **Saxon-PE** versions of the Saxon product and you must select one of those two versions for the **Target** when you configure the SEF file *(on page 405)*. When configuring the XSLT transformation, you will specify the SEF file in the **XSL URL** field *(on page 692)*.

Compiling an SEF File

The **Compile XSL Stylesheet for Saxon** tool can be found in the **XML Tools** menu and it compiles a specified stylesheet as an XML file (*stylesheet export file* with a file extension of `.sef`).

Selecting this tool opens the **Compile XSL Stylesheet for Saxon** dialog box that allows you to configure some options for conversion.

Figure 377. Compile XSL Stylesheet for Saxon Dialog Box



This dialog box includes the following options:

XSL URL

Allows you to select URL of the source XSL stylesheet. You can specify the URL by using the text field, the history drop-down, or the browsing actions in the **Browse** drop-down list.

Target

Allows you to select the type of Saxon product that the export file will be used with. You can choose **Saxon-JS**, **Saxon-EE**, **Saxon-PE**, or **Saxon-HE**.

Relocatable

Can be used to control the Saxon `-relocate` parameter. You can select this option to produce a *relocatable* export package (SEF) that can be deployed to a different location, with a different base URI.

Output file

You can specify the path where the output file will be saved by entering it in the text field, using the **Insert Editor Variables** button, or using the browsing actions in the **Browse** drop-down list.

Open in Editor

Select this option to open the resulting *stylesheet export file* in the main Oxygen XML Developer Eclipse plugin editing pane.

Use a configuration file ("-config")

Select this option if you want to use a Saxon 9.9.1.5 configuration file that will be executed for the XSLT transformation and validation processes. You can specify the path to the configuration file by entering it in the **URL** field, or by using the **Insert Editor Variables** button, or using the browsing actions in the **Browse** drop-down list.

Compile

Use this button to generate the *stylesheet export file* according the options selected in this dialog box.

JSON Tools

Oxygen XML Developer Eclipse plugin includes some useful tools for converting JSON to and from XML, converting XSD to JSON Schema, and generating JSON instances or a JSON Schema.

JSON to XML Converter

Online JSON to XML Converter

⚠ **Attention:**

For a simple ONLINE tool for converting a single JSON file to XML, or vice versa, go to: https://www.oxygenxml.com/xml_json_converter.html.

Converting JSON to XML in Oxygen

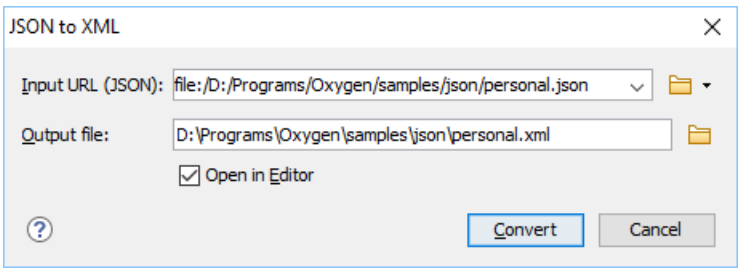
Oxygen XML Developer Eclipse plugin includes a useful and simple tool for converting JSON files to XML. The **JSON to XML** action for invoking the tool can be found in the **XML Tools > JSON Tools** menu.

To convert a JSON document to XML, follow these steps:

1. Select the **JSON to XML** action from the **XML Tools > JSON Tools** menu.

   The **JSON to XML** dialog box is displayed:

   Figure 378. JSON to XML Dialog Box

2. Choose or enter the **Input URL** of the JSON document.
3. Choose the path of the **Output file** that will contain the resulting XML document.
4. Select the **Open in Editor** option to open the resulting XML document in the main editing pane.
5. Click the **Convert** button.

Result: The original JSON document is now converted to an XML document.

Figure 379. Example: XML to JSON Operation Result

Conversion Details

- If the JSON document has more than one property on the first level, the converted XML document will have an additional root element called `<JSON>`.

  For example, the following JSON document:

```
{
   "personnel": {
    "person": [
      {"name": "Boss"},
      {"name": "Worker"}
    ]
  },
  "id":"personnel-id"
}
```

  it is converted to:

```
<?xml version="1.0" encoding="UTF-8"?>
<JSON>
  <personnel>
    <person>
      <name>Boss</name>
    </person>
```

```
    <person>
      <name>Worker</name>
    </person>
  </personnel>
  <id>personnel-id</id>
</JSON>
```

- If the JSON document is an array, the converted XML document will have a root element called `<array>` and for each item within the array, another `<array>` is created.

```
[
  {"name": "Boss"},
  {"name": "Worker"}
]
```

it is converted to:

```
<?xml version="1.0" encoding="UTF-8"?>
<array>
  <array>
    <name>Boss</name>
  </array>
  <array>
    <name>Worker</name>
  </array>
</array>
```

- If the name of a JSON property contains characters that are not valid in XML element names (for example, **$**), then the invalid characters will be escaped as its hexadecimal equivalent in the converted XML.

```
{"$id": "personnel-id"}
```

is converted to:

```
<_X24_id>personnel-id</_X24_id>
```

Related Information:

XML to JSON Converter *(on page 573)*

XML to JSON Converter

Online XML to JSON Converter

⚠ **Attention:**

For a simple ONLINE tool for converting a single XML file to JSON, or vice versa, go to: https://www.oxygenxml.com/xml_json_converter.html.

Converting XML to JSON in Oxygen

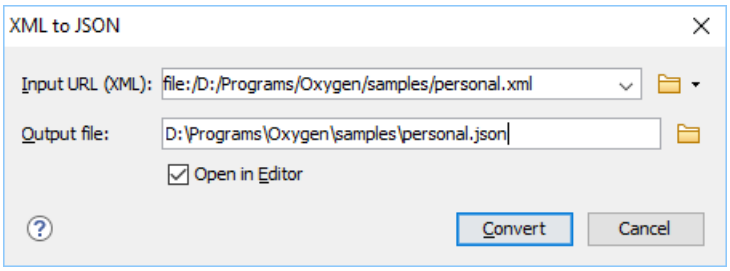
Oxygen XML Developer Eclipse plugin includes a useful and simple tool for converting XML files to JSON. The **XML to JSON** action for invoking the tool can be found in the **XML Tools > JSON Tools** menu.

To convert an XML document to JSON, follow these steps:

1. Select the **XML to JSON** action from the **XML Tools > JSON Tools** menu.

   Step Result: The **XML to JSON** dialog box is displayed:

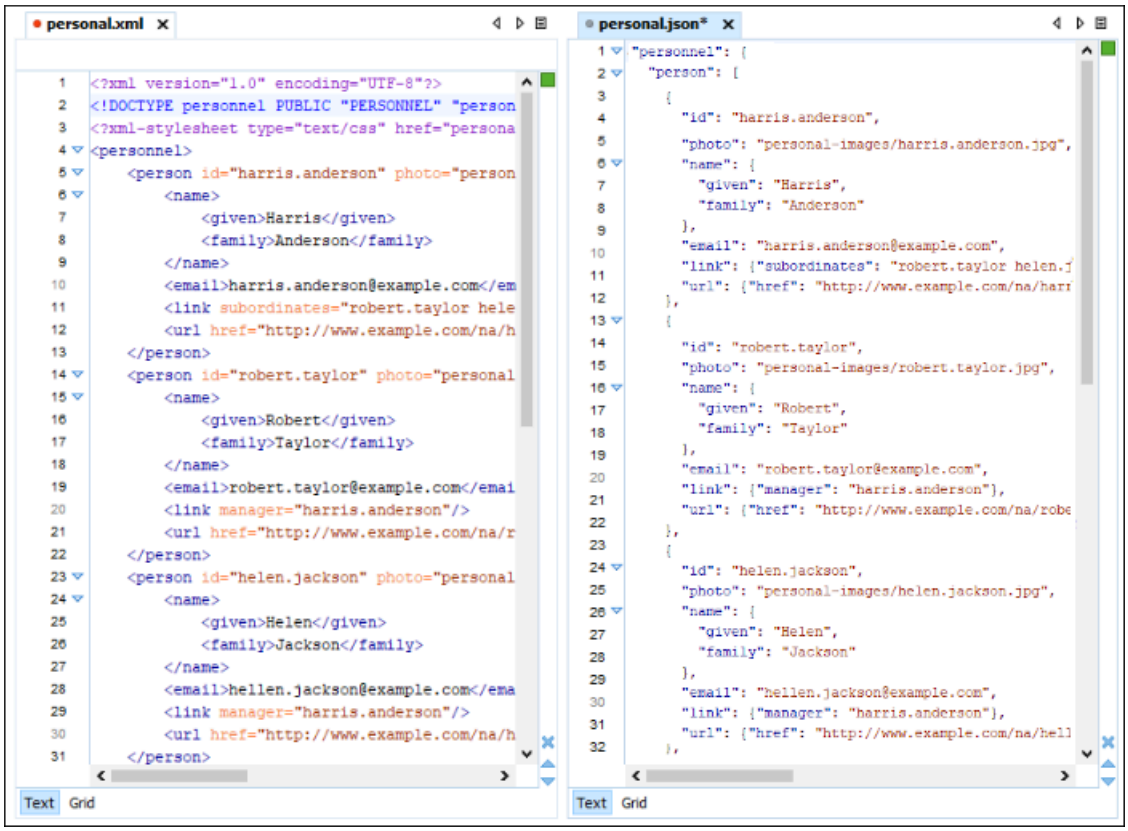
   Figure 380. XML to JSON Dialog Box

2. Choose or enter the **Input URL** of the XML document.
3. Choose the path of the **Output file** that will contain the resulting JSON document.
4. Select the **Open in Editor** option to open the resulting JSON document in the main editing pane.
5. Click the **Convert** button.

Result: The original XML document is now converted to a JSON document.

Figure 381. Example: XML to JSON Operation Result



Conversion Details

- If any elements contain attributes in the XML document, the attributes are converted to properties in the converted JSON document. If the XML document contains more than one element with the same name, they will be converted into an array of object in the converted JSON document.

  For example, the following XML document:

  ```
  <personnel>
    <person id="person.one">
      <name>Boss</name>
    </person>
    <person id="person.two">
      <name>Worker</name>
    </person>
  </personnel>
  ```

  it is converted to:

  ```
  {
    "personnel": {
      "person": [
        {
          "id": "person.one",
          "name": "Boss"
        },
        {
          "id": "person.two",
          "name": "Worker"
        }
      ]
    }
  }
  ```

- If the XML document contains elements with mixed content (text plus elements), the converted JSON document will contain a `#text` property with its value set as the text content. If there are multiple text nodes, the subsequent `#text` properties will contain a number (i.e. `#text1`, `#text2`).

  ```
  <person>
    <name>Boss <i>One</i>!</name>
  </person>
  ```

  is converted to:

  ```
  {
    "person": {
      "name": {
        "#text": "Boss",
  ```

```
      "i": "One"
      "#text1": "!"
    }
  }
}
```

- If the XML document contains element names that contains hexadecimal codes (for example, if they were escaped during a JSON to XML conversion *(on page 570)*), it will be converted to the normal character value in the converted JSON document.

  ```
  <_X24_id>personnel-id</_X24_id>
  ```

  is converted to:

  ```
  {"$id": "personnel-id"}
  ```

Related Information:

JSON to XML Converter *(on page 570)*

XSD to JSON Schema Converter

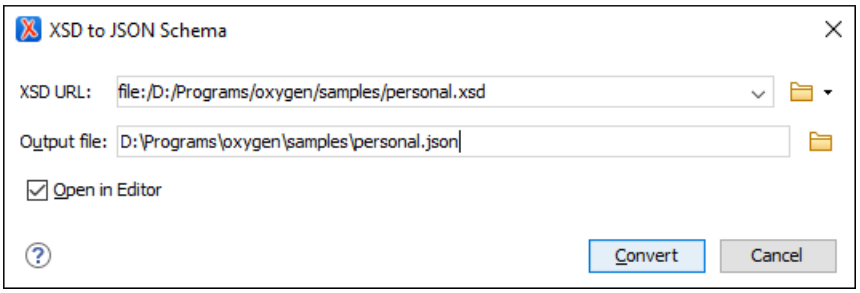
Oxygen XML Developer Eclipse plugin includes a tool for converting an XML Schema file (XSD) to a JSON Schema file. The **XSD to JSON Schema** action for invoking the tool can be found in the **Tools > JSON Tools** menu. It requires an additional add-on to be installed, so the first time you invoke the action, Oxygen XML Developer Eclipse plugin will present a dialog box asking if you want to install it. Once installed, you need to restart Oxygen XML Developer Eclipse plugin and the **XSD to JSON Schema** action will invoke the tool.

To convert an XML Schema (XSD) to a JSON Schema, follow these steps:

1. Select the **XSD to JSON Schema** action from the **Tools > JSON Tools** menu.

   Step Result: The **XSD to JSON Schema** dialog box is displayed:

   Figure 382. XSD to JSON Schema Dialog Box

2. Choose or enter the **XSD URL** of the XML Schema document. The conversion supports XSD versions 1.0 and 1.1.
3. Choose the path for the **Output file**.
4. Select the **Open in Editor** option to open the resulting JSON Schema document in the main editing pane.
5. Click the **Convert** button.

Result: The original XSD document is now converted to a JSON Schema document. The resulting JSON Schema will be *draft-07* and will contain:

- The `$id` of the schema, generated from XSD `targetNamespace`.
- The `$definitions` section, which declares complex and *enum* types.
- The `anyOf` section, which lists possible top-level elements as an array of objects.

Other Possible Results:

- If an XSD type extends another type, then its schema is combined with the schema of the base type using the `allOf` keyword.
- If an extension in XSD defines an element with the same name as an attribute in the base, a property named `rest` is generated to avoid name conflicts in JSON.
- If a property of a complex type is a collection property, the schema of the collection items will be wrapped in the JSON array schema.

Conversion Mappings

The following table lists the specific conversion mapping details.

XML Schema Type	JSON Schema Representation
anySimpleType	string, number, integer, boolean, null
anyType	string, number, integer, boolean, null, object, array
string	string
normalizedString	string
token	string
language	string
Name	string
NCName	string
ID	string
IDREF	string
IDREFS	array of strings
ENTITY	string
ENTITIES	array of strings
NMTOKEN	string
NMTOKENS	array of strings
boolean	boolean

XML Schema Type	JSON Schema Representation
base64Binary	array of integers
hexBinary	array of integers
float	number
decimal	number
integer	integer
nonPositiveInteger	integer
negativeInteger	integer
long	integer
int	integer
short	integer
byte	integer
nonNegativeInteger	integer
unsignedLong	integer
unsignedInt	integer
unsignedShort	integer
unsignedByte	integer
positiveInteger	integer
double	number
anyURI	string with "format":"uri"
QName	object with "namespaceURI", "localPart", "prefix"
duration	string
dateTime	string with "format":"date-time"
date	string with "format":"date"
time	string with "format":"time"

Conversion Limitations

In most cases, the conversion creates an equivalent schema, but there are some limitations:

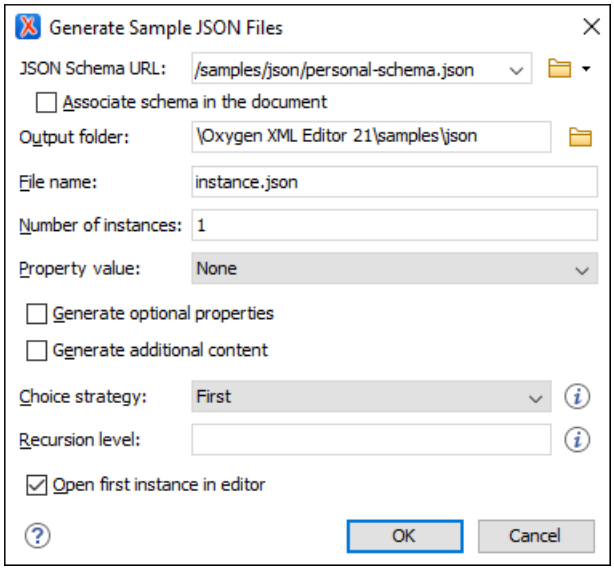
- Restrictions/facets are not taken into consideration when converting (*fractionDigits*, *pattern*, *totalDigits*, *whiteSpace*, *minInclusive*, *maxInclusive*, and the restrictions for length, except *enumeration)*. However, extensions and indicators are properly converted (*minOccurs*, *maxOccurs*, *group*, *sequence*, *choice*).
- The `<documentation>` element is not converted into `<description>`.

- The `@substitutionGroup` attribute for an element that has no declared type becomes a reference to the element that can substitute it.
- The `@block` attribute is not taken into consideration during the conversion.

Generating Sample JSON Files

Oxygen XML Developer Eclipse plugin includes a tool for generating sample JSON files. To generate sample JSON files from a JSON Schema, select **Generate Sample JSON Files** from the **XML Tools > JSON Tools** menu. The action opens a dialog box where you can configure a variety of options for generating the files.

Figure 383. Generate Sample JSON Files Dialog Box



The **Generate Sample JSON Files** dialog box includes the following fields and options:

Schema URL

The URL of the Schema location. You can specify the path by using the text field, the history drop-down menu, or the browsing actions in the **Browse** drop-down list.

Associate schema in the document

If enabled, the specified schema will be associated with the generated files.

Output folder

Path to the folder where the generated JSON instances will be saved.

File name

The name of the instance(s) that will be generated. By default, `instance.json` is used.

Number of instances

The desired number of JSON instances to be generated. When more than one instance is generated, the index of the instance will be added to its file name.

Property value

You can specify the way the values of the properties are generated. The following options are available:

- *None* - Assigns empty values for properties (a template file will be generated). This is the default value.
- *Default* - Assigns the name of the property as the value (for strings) or assigns the specified minimum value (for numbers).
- *Random* - Assigns random values according to schema restrictions.

Generate optional properties

If selected, the JSON instance will be generated with optional properties that are defined in the JSON schema. Otherwise, only the required properties will be generated.

Generate additional content

If selected, the JSON instance will be generated with additional properties that are defined in the JSON schema as `additionalProperties` and additional items that are defined as `additionalItems` (in the case of an Array).

Choice strategy

You can specify the way an instance will be generated from a schema that contains a `CombinedSchema` (with either *oneOf* or *anyOf*). The following options are available:

- *First* - The first defined schema in *oneOf* or *anyOf* will be used.
- *Random* - A random schema defined in *oneOf* or *anyOf* will be used.

Recursion level

This option controls the maximum allowed depth (must be a number), in case the selected schema contains recursive calls of `$ref` schemas referencing one another. By default, it is set to 1, meaning that the generation for the recursive calls will stop after the first iteration.

Open first instance in editor

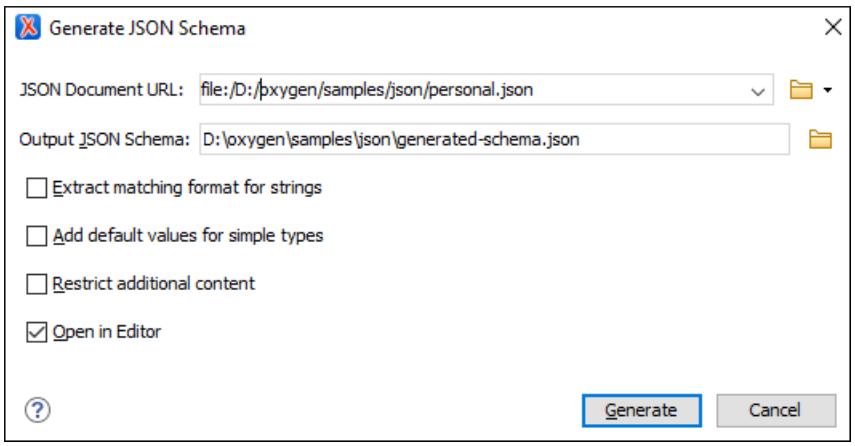
If selected, the first generated instance is opened in the editor.

You can click **OK** at any point to generate the sample JSON files.

Generating JSON Schema from a JSON File

Oxygen XML Developer Eclipse plugin includes a tool for generating a sample JSON Schema from a JSON file. To generate a sample JSON Schema, select **Generate JSON Schema** from the **XML Tools > JSON Tools** menu. The action opens a dialog box where you can configure a variety of options for generating the JSON Schema file.

Figure 384. Generate JSON Schema Dialog Box



The **Generate JSON Schema** dialog box includes the following fields and options:

JSON Document URL

The URL of the JSON file. You can specify the path by using the text field, the history drop-down menu, or the browsing actions in the **Browse** drop-down list.

Output JSON Schema

Path to the folder where the generated JSON Schema will be saved.

Extract matching format for strings

If selected, the generator will try to find a format that matches the string values from the JSON Document.

Add default values for simple types

If selected, the *default* values (*0* for number, "" for string, *false* for boolean) and *examples* for string will be added.

Restrict additional content

If selected, *additionalProperties* (for objects) and *additionalItems* (for arrays) will be set to *false* in the resulting schema. By default, these keys are not in the schema, meaning that providing additional content (according to the schema) is allowed.

Open in Editor

If selected, the generated JSON Schema is opened in the editor.

You can click **Generate** at any point to generate the JSON Schema.

Generate Documentation

Oxygen XML Developer Eclipse plugin includes a tool for generating documentation for XSLT, XML Schema, XQuery, and WSDL documents.

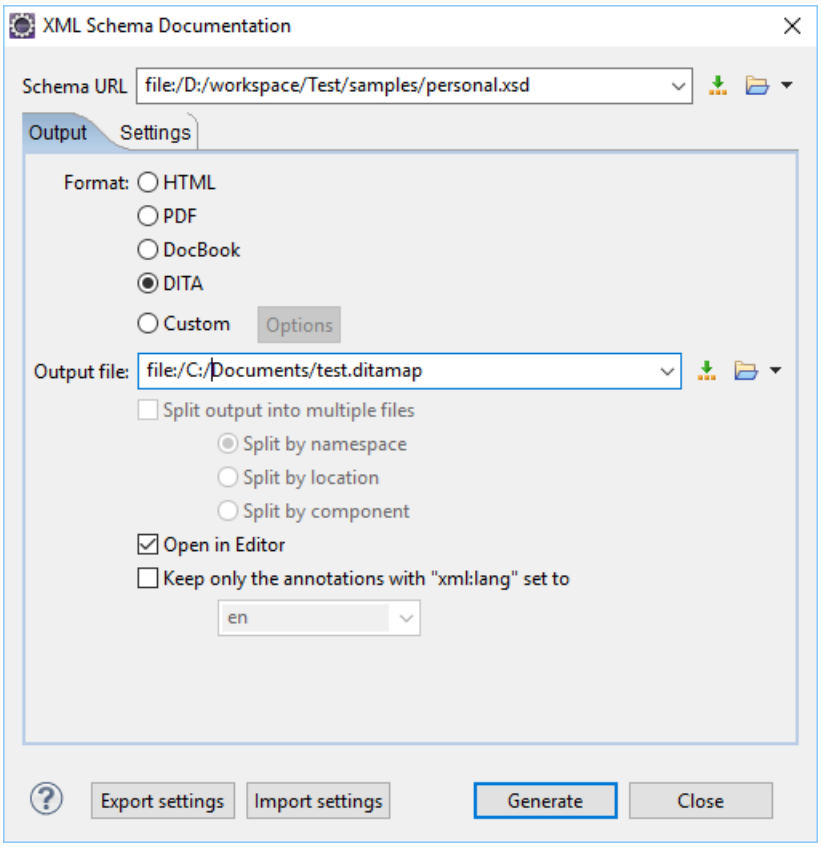
Generating Documentation for an XML Schema

Oxygen XML Developer Eclipse 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 *(on page 192)*.

Figure 385. XML Schema Documentation Dialog Box



The **Schema URL** field of the dialog box must contain the full path to the XML Schema (XSD) file that will have documentation generated. 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 *(on page 470)*.
  - **PDF** - The documentation is generated in PDF output format *(on page 473)*.
  - **DocBook** - The documentation is generated in DocBook output format *(on page 473)*.

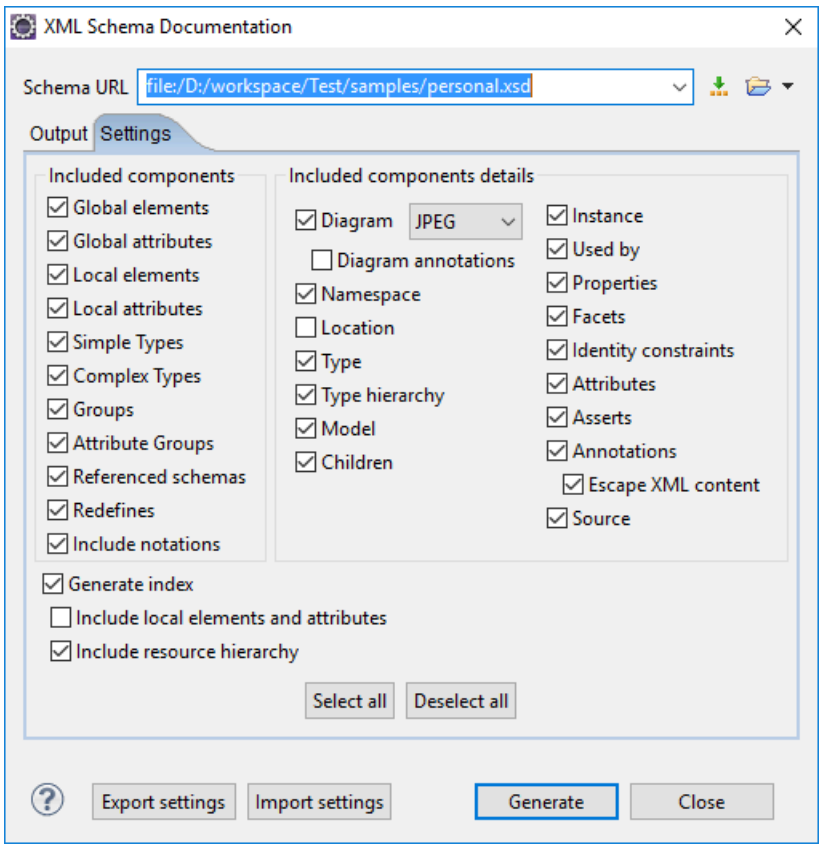
    - **DITA** - The documentation is generated in DITA output format *(on page 473)*.
    - **Custom** - The documentation is generated in a custom output format *(on page 473)*, allowing you to control the output. Click the **Options** button to open a **Custom format options** dialog box where you can 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. For DITA and DocBook documents, this option appears as **Open in Editor** and the result will be opened in Oxygen XML Developer Eclipse plugin (in the current editor).

  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 386. 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 *(on page 88)* 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 selected, 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 deselected by default.

Export settings - Save the current settings in a settings file for further use (for example, if you need the exported settings file for generating the documentation from the command-line interface).

Import settings - Reloads the settings from the exported file.

Generate - Use this button to generate the XML Schema documentation.

Tip: This function can be executed from an automated command-line script, but it requires an additional **Oxygen XML Scripting License**. For more details, see https://www.oxygenxml.com/oxygen_scripting.html.

Related Information:

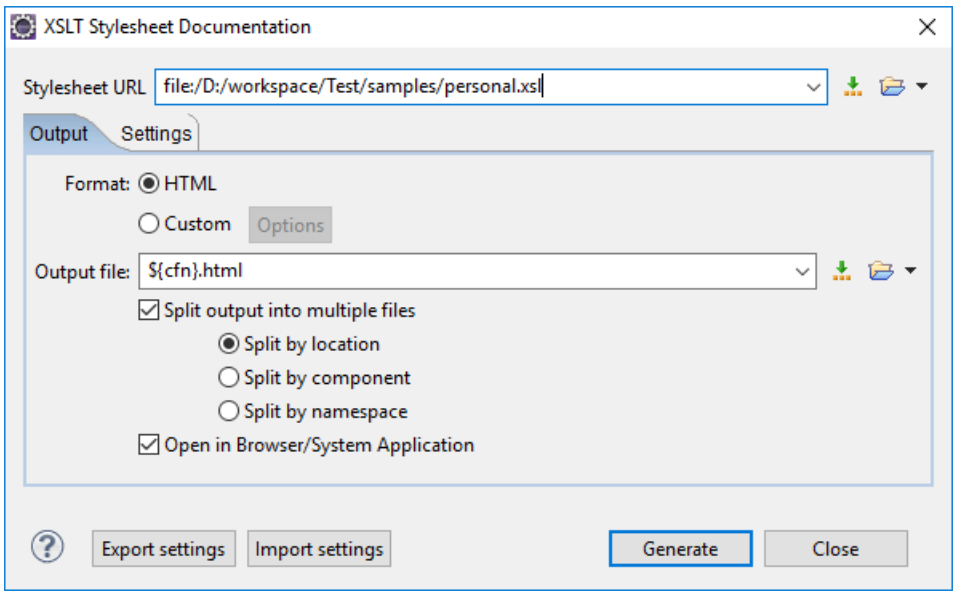
Customizing PDF or DocBook Output of Generated XML Schema Documentation *(on page 474)*

Generating Documentation for an XSLT Stylesheet

You can use Oxygen XML Developer Eclipse 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 *(on page 403)*, 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 *(on page 192)*.

Figure 387. 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:

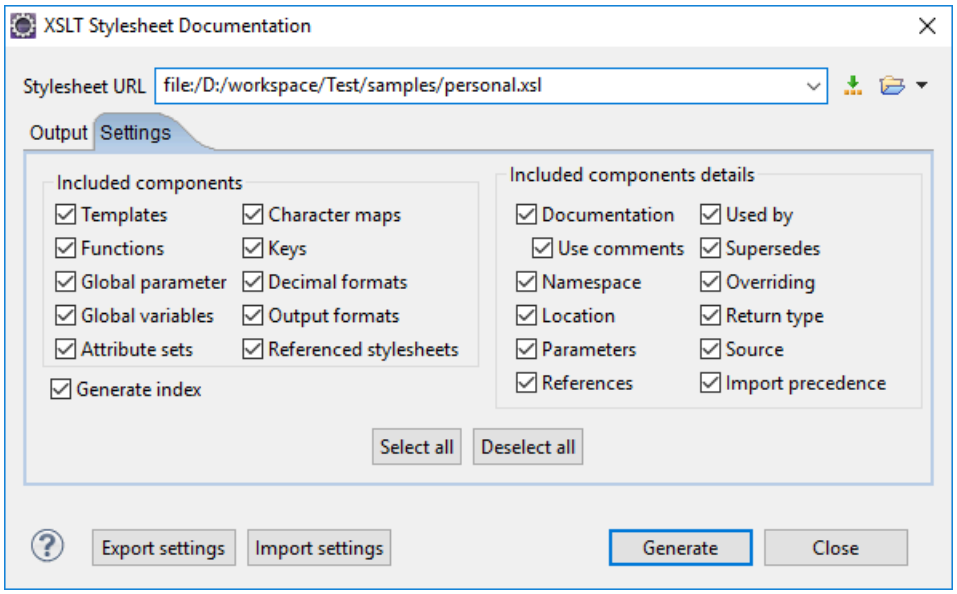
- **Format** - Allows you to choose between the following formats:
  - **HTML** - The documentation is generated in HTML output format *(on page 401)*.
  - **Custom** - The documentation is generated in a custom output format *(on page 403)*, allowing you to control the output. Click the **Options** button to open a **Custom format options** dialog box *(on page 404)* 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 388. 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 Eclipse 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 (on page 1325)*. 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 *(on page 403)* instead of the built-in HTML format *(on page 401)* 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, if you need the exported settings file for generating the documentation from the command-line interface).

Import settings - Reloads the settings from the exported file.

Generate - Use this button to generate the XSLT documentation.

Tip: This function can be executed from an automated command-line script, but it requires an additional **Oxygen XML Scripting License**. For more details, see https://www.oxygenxml.com/oxygen_scripting.html.

Related Information:

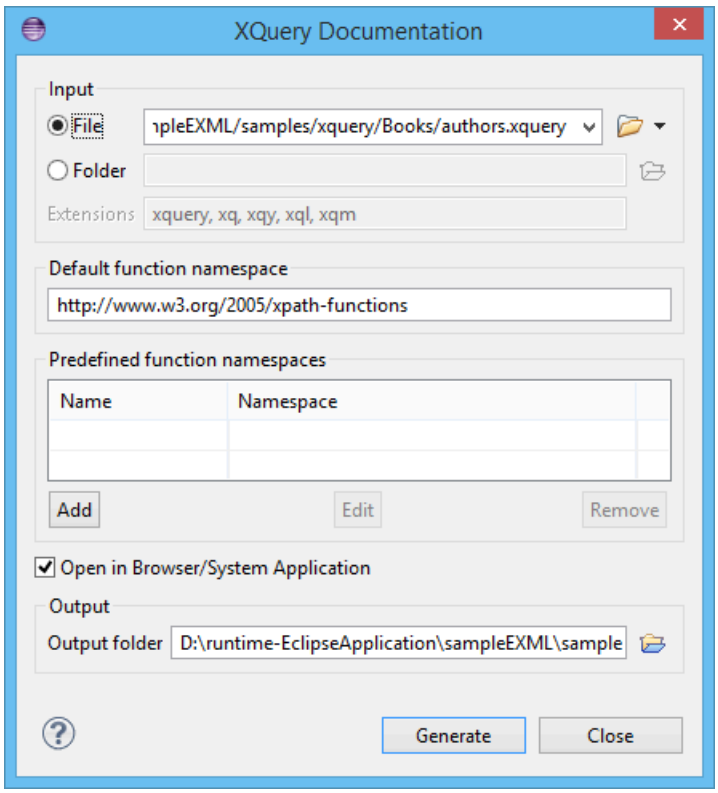
XSLT Stylesheet Component Documentation Support *(on page 387)*

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 *(on page 192)*.

The dialog box allows you to configure a set of parameters for the process of generating the HTML documentation.

Figure 389. 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 to be used for generating the documentation.
  - **Folder** - The directory that contains the files to be used for generating 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

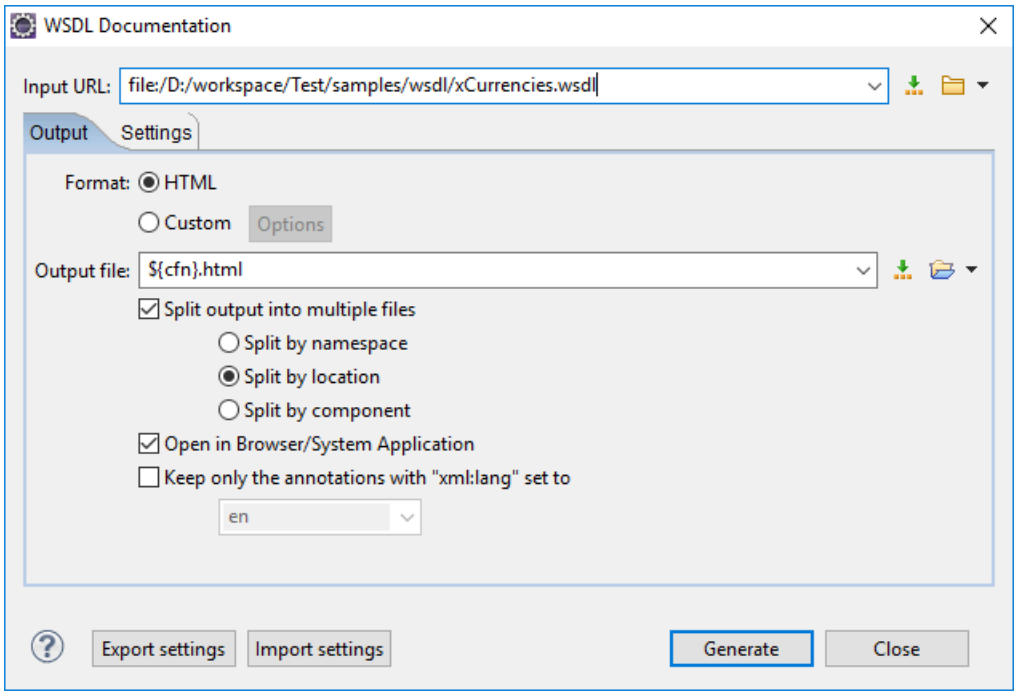
You can use Oxygen XML Developer Eclipse plugin to generate detailed documentation for the components of a WSDL document in HTML format. You can select the WSDL components to include in your output and the

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 *(on page 518)* 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 *(on page 192)*.

Figure 390. 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 *(on page 517)*.
  - **Custom** - The documentation is generated in a custom output format *(on page 518)*, 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.

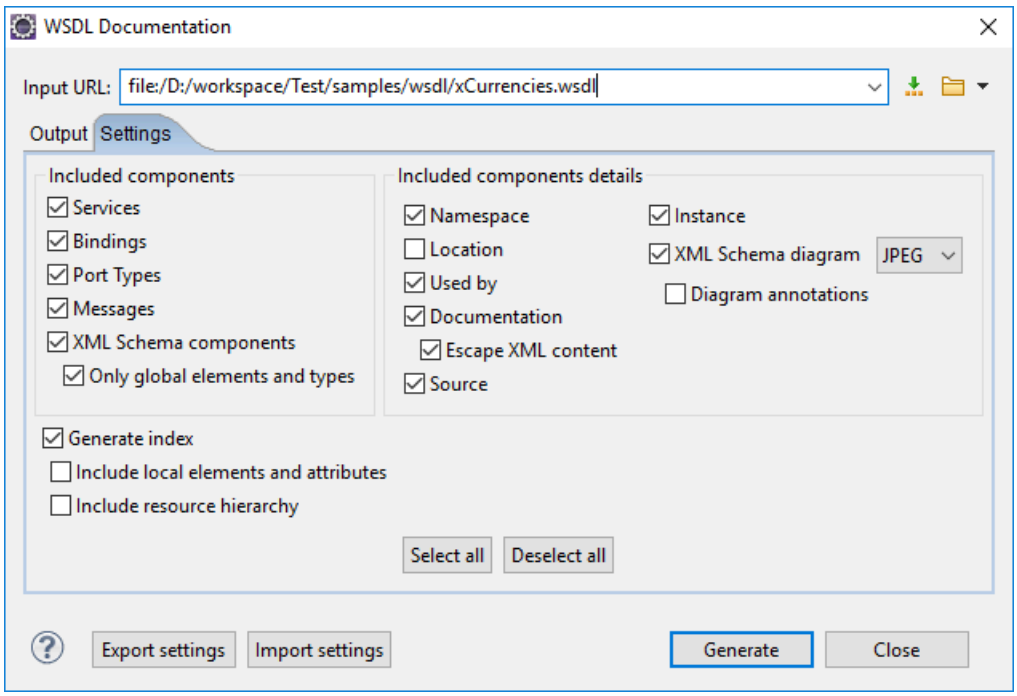
Note: To set the browser or system application that will be used, go to **Window > Preferences > General > Web Browser** and specify it there. This will take precedence over the default system application settings.

- **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 391. 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 selected, 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 deselected by default.

Export settings - Save the current settings in a settings file for further use (for example, if you need the exported settings file for generating the documentation from the command-line interface).

Import settings - Reloads the settings from the exported file.

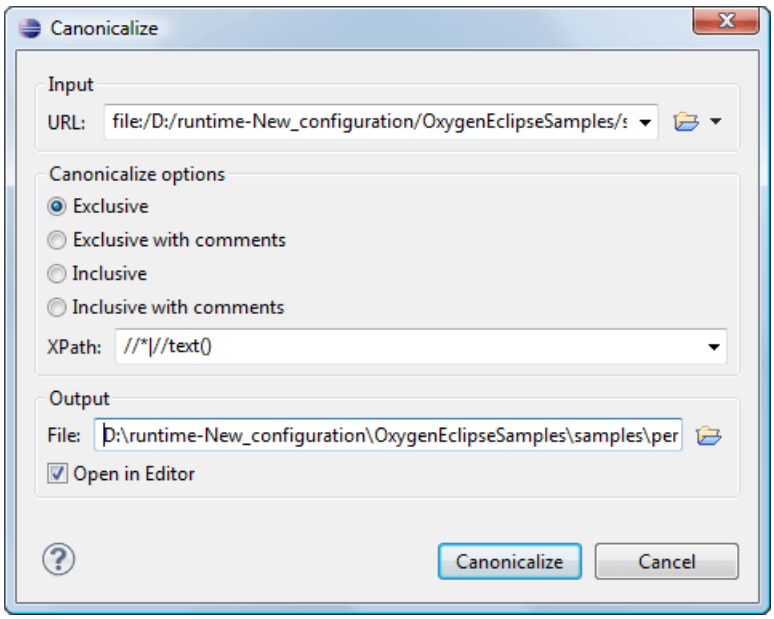
Generate - Use this button to generate the WSDL documentation.

Tip: This function can be executed from an automated command-line script, but it requires an additional **Oxygen XML Scripting License**. For more details, see https://www.oxygenxml.com/oxygen_scripting.html.

Canonicalizing Files

You can select the *canonicalization (on page 1323)* 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 392. 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 (on page 1323)* 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 (on page 1323)* method is used.
- **Inclusive** - If selected, the inclusive (uncommented) *canonicalization (on page 1323)* 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 standpoint 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 (on page 1323)* 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 selected, the output file will be opened in the editor.

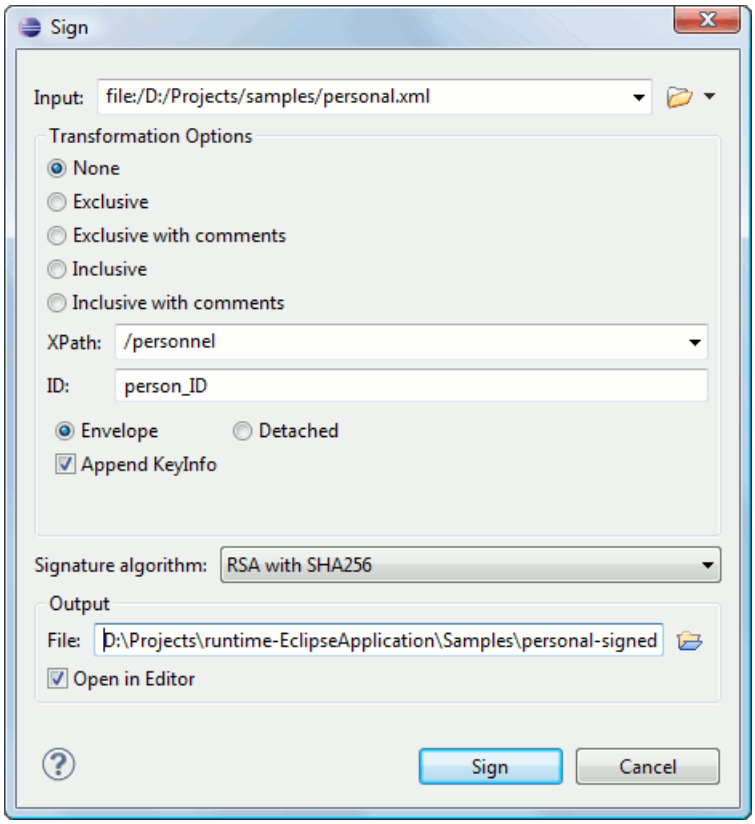
Related Information:

Digital Signatures Overview *(on page 356)*

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 393. Signature Settings Dialog Box



The following options are available:

Note: If Oxygen XML Developer Eclipse 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 *(on page 127)* where you can configure a valid certificate.

- **Input** - Available if the **Sign** action was selected from the **XML Tools** menu. Specifies the location of the input URL.
- **Transformation Options** - See the Digital Signature Overview *(on page 356)* section for more information about these options.
    - **None** - If selected, no *canonicalization (on page 1323)* algorithm is used.
    - **Exclusive** - If selected, the exclusive (uncommented) *canonicalization (on page 1323)* 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 (on page 1323)* method is used.
    - **Inclusive** - If selected, the inclusive (uncommented) *canonicalization (on page 1323)* 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 standpoint 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 (on page 1323)* 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 *(on page 356)* for more information.
- **Detached** - If selected, the *detached* signature is used. See the Digital Signature Overview *(on page 356)* for more information.
- **Append KeyInfo** - If this option is selected, 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 selected, the output file will be opened in Oxygen XML Developer Eclipse plugin.

Related Information:

Digital Signatures Overview *(on page 356)*

Verifying Signature *(on page 362)*

Example of How to Digitally Sign XML Files or Content *(on page 362)*

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 Information:

Digital Signatures Overview *(on page 356)*

Signing Files *(on page 360)*

Example of How to Digitally Sign XML Files or Content *(on page 362)*

WSDL SOAP Analyzer Tool

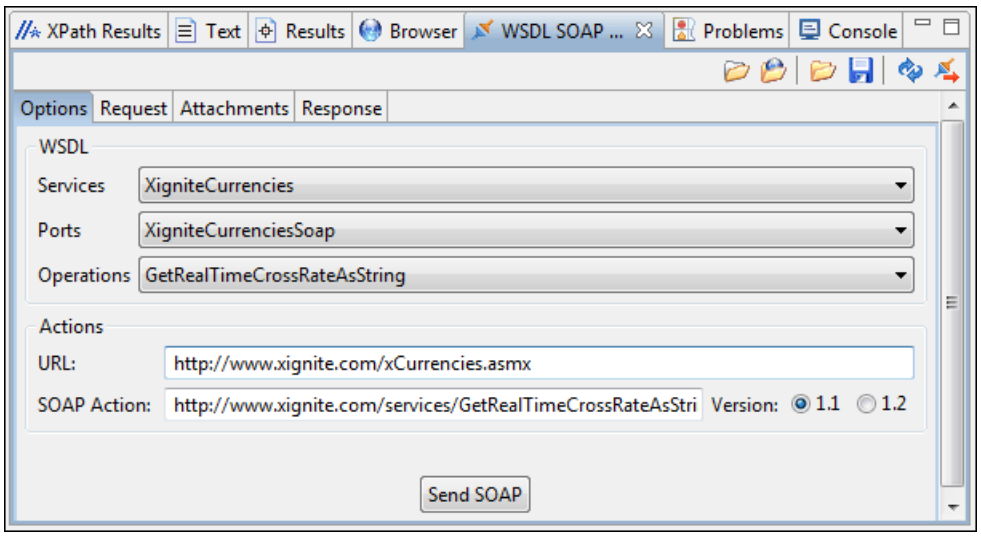
WSDL SOAP Analyzer is a tool that helps you test if the messages defined in a Web Service Descriptor (WSDL) are accepted by a Web Services server.

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).

Oxygen XML Developer Eclipse 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** *(on page 192)* view.

Figure 394. 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 Eclipse 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 (on page 1324)* 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 Eclipse plugin remembers the modified request for each one. You can click 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 Eclipse 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 Eclipse 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 *(on page 521)* 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

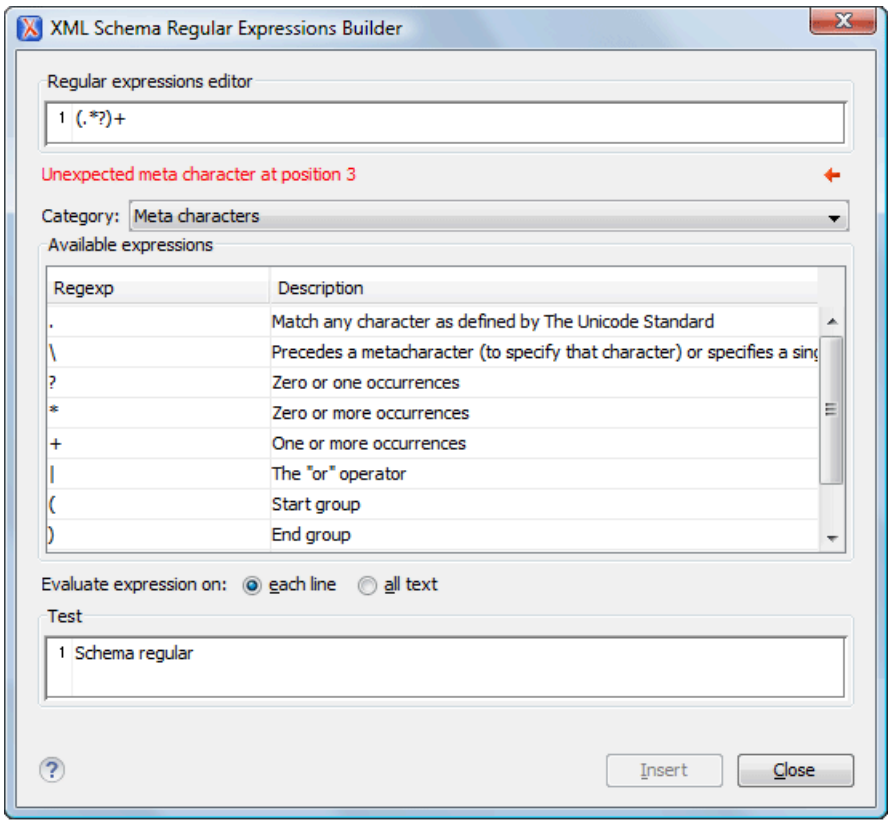
To open and test a remote WSDL file the steps are the following:

1. Go to **Window > Show View > Other > Oxygen XML Developer Eclipse plugin > WSDL SOAP Analyzer**.
2. Click the **Choose WSDL** button and enter the URL of the remote WSDL file.
3. Click the **OK** button.
   This will open the **WSDL SOAP Analyzer** tool *(on page 519)*. In the **Saved SOAP Request** tab you can open directly a previously saved Web Service SOAP Call (WSSC) file, thus skipping the analysis phase.

XML Schema Regular Expressions Builder Tool

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 395. 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 that will have the regular expression 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 *(on page 168)* or schema **Design** mode *(on page 169)*. Accordingly, the **Insert** button will be not available 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.

18.

Common Problems

This section provides a collection of common performance and other types of problems that might be encountered when using Oxygen XML Developer Eclipse 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 Eclipse plugin.

Out of Memory on External Processes

Problem

Oxygen XML Developer Eclipse plugin throws an *Out Of Memory* error when trying to generate PDF output with the built-in Apache FOP processor.

Cause

The amount of allocated memory might be insufficient.

Solutions

- Open the **Preferences** dialog box *(on page 36)*, go to **XML > XSLT-FO-XQuery > FO Processors**, and increase the value of the **Memory available to the built-in FOP** option *(on page 116)*.
- 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 115)*

Custom Engines Preferences *(on page 130)*

How to Enable Debugging for FO Processor Transformations *(on page 749)*

Performance Issues with Large Documents

Problem

The performance of the application slows down considerably over time when working with large documents.

Cause

A possible cause is that the application needs more memory to run properly.

Solutions

- You can increase the maximum amount of memory available to Oxygen XML Developer Eclipse plugin by setting the `-vmargs` and `-Xmx` parameters in the command used to launch the Eclipse platform.

  ⚠ **Attention:** The maximum amount of memory should be less than 75% of the physical amount of memory available on the machine. Otherwise, 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.

Application Reports Errors When Starting After Installing a New Version

Problem

Sometimes after installing a new version of Oxygen XML Developer Eclipse plugin, various errors are reported when the application starts.

Cause

This problem may occur if you install the application in a folder where an older version of the application was previously installed. Especially on Mac OS X, there is a possibility for older resources, libraries from the previous application, to remain in the installation folder and break the functionality of the newer version of the application.

Solution

Close the application and completely uninstall it *(on page 35)*, then install the application again. The user-specific settings are saved in a separate folder in the user home directory so they will not be lost.

- On Mac OS X, you can move the entire application installation folder to the **Trash**, then re-install.
- On Linux and Windows, you can uninstall using the facilities provided by the installer *(on page 35)*, then re-install.

Application Takes Several Minutes to Start

Problem

Oxygen XML Developer Eclipse plugin seems to take an abnormally long amount of time to start.

Cause

Some anti-virus software can cause Java applications, such as Oxygen XML Developer Eclipse plugin, to start very slowly due to scanning compressed archives (such as the *JAR* libraries that all Java applications use).

Solution

A possible solution is to add the Oxygen XML Developer Eclipse plugin folder to the list of exceptions in the anti-virus software settings.

Compatibility Issue Between Java and Certain Graphics Card Drivers

Problem

Under certain settings, a compatibility issues 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.

Solution

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 Eclipse plugin, it is no longer associated to the appropriate file types (such as XML, XSL, XSD). How can I re-create the file associations?

Cause

The upgrade damaged the file associations in the LaunchService Database on your OS X machine.

Solution

You can rebuild the LaunchService Database with the following procedure. This will reset all file associations and rescan the entire file system searching for applications that declare file associations and collect them in a database used by Finder.

1. Find all the Oxygen XML Developer Eclipse 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 Eclipse 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 Eclipse plugin icon.
9. Double-click the file.
10. Accept the confirmation.

Result: When you start Oxygen XML Developer Eclipse 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 log files and attach them 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 *(on page 8)* 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 the Oxygen XML Developer Eclipse plugin log files, follow these steps:

1. Create a text file called `log4j2.properties` in the `lib` folder of the installed *plugin* folder, with the following content:

```
name = PropertiesConfig
appender.R2.type = RollingFile
appender.R2.name = R2
appender.R2.fileName = ${sys:user.home}/Desktop/oxygenLog/oxygen.log
appender.R2.filePattern = ${sys:user.home}/Desktop/oxygenLog/oxygen%i.log.gz
appender.R2.layout.type = PatternLayout
appender.R2.layout.pattern = %r %p [ %t ] %c - %m%n
appender.R2.policies.type = Policies
appender.R2.policies.size.type = SizeBasedTriggeringPolicy
appender.R2.policies.size.size=12MB
appender.R2.strategy.type = DefaultRolloverStrategy
appender.R2.strategy.max = 20
rootLogger.level = debug
rootLogger.appenderRefs = R2
```

```
rootLogger.appenderRef.R2.ref = R2
```

2. Restart the application.
3. Reproduce the error.
4. Close the application.
5. Delete the `log4j2.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, do not forget to delete the `log4j2.properties` file when you are done with the procedure.

Result: The resulting log files are named `oxygen.log` and `oxygen#.log.qz` (for example, `oxygen.log`, `oxygen1.log.gz`, `oxygen2.log.gz`, etc.) and are located in the `Desktop\oxygenLog` folder.

DITA Map Transformation Fails (Cannot Connect to External Location)

Problem

DITA map (on page 1324) transformation fails because it cannot connect to an external location.

Solution

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 Map WebHelp Transformation Fails (Duplicate Topic References Found)

Problem

DITA Map WebHelp transformation fails with a message that indicates duplicate topic references were found.

Cause

By default the WebHelp transformation uses the `force-unique` parameter set to **true** to force the transformation to create unique output files for each instance of a resource when a map contains multiple references to a single topic. However, there are cases when this feature does not work as expected and the duplicate topic references are not handled properly.

Solution

To solve this issue, you should manually set a unique `@copy-to` attribute on any duplicate topic reference that was not handled automatically by DITA-OT:

```
<map>
...
  <topicref href="../topics/MyTopic.dita"/>
...
```

```
  <topicref href="../topics/MyTopic.dita" copy-to="../topics/MyTopic-2.dita"/>
</map>
```

DITA-OT Transformation Takes a Long Time to Process

Problem

A DITA transformation takes an extremely long time to process (over an hour, for example).

Cause

Large delays in DITA-OT processing are usually caused by intensive disk operations, CPU usage, or connections to remote websites. The DITA-OT processing is very disk-intensive, each stage takes the entire content from the transformation temporary files folder, reads it, modifies it, and then writes it back.

Solution

There are several things you can try to troubleshoot this problem:

- If you are using a shared or remote drive, it is recommended to specify a local drive for the output and temporary files directory (edit the transformation scenario and in the **Output** tab, select a local directory for **Temporary files directory** and **Output directory**).
- If you want to test if the publishing has a problem downloading remote resources, you could disable the network adapter on the computer and then try to publish. The purpose is to see if the publishing finishes without any reported error about obtaining a certain HTTP resource.
- Using DTDs instead of XML Schemas is faster. This is because of a default transformation parameter called `args.grammar.cache` that only works for DTD-based DITA topics.
- You can increase the memory available to Oxygen XML Developer Eclipse plugin *(on page 1305)*. Sometimes, just increasing the amount of memory available to the DITA-OT process may be enough to lower the time necessary for the publishing to run.
- You can enable some logging to help you determine which stage in the process is taking a long time. Edit the transformation scenario and in the **Advanced** tab, enter **logger org.apache.tools.ant.listener.ProfileLogger** in the **Additional arguments** field. Then go to **Options > Preferences > DITA > Logging** and select **Always** for the **Show console output** option.
- You could try disabling antivirus applications since the publishing process is very disk intensive and certain antivirus application might slow down the process.
- If the published DITA map is part of a larger DITA project with lots of maps and topics, references from topics in the current map to topics in other sub-projects might result in problems resolving those references. You could look in the output folder to see if the number of HTML documents match the number of DITA topics in your map.

DITA PDF Transformation Fails

Problem

The DITA to PDF transformation fails.

Cause

To generate the PDF output, Oxygen XML Developer Eclipse plugin uses the DITA Open Toolkit. This process sometimes results in errors. For information about some of the most common errors, see DITA PDF Processing Common Errors.

Solution

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.fo** parameter to **yes**. Run the transformation, go to the temporary directory of the transformation, open the `topic.fo` file and go to the line indicated by the error. Depending on the XSL FO context try to find the DITA topic that contains the text 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 send it to support@oxygenxml.com.

Related Information:

How to Enable Debugging for FO Processor Transformations *(on page 749)*

DITA PDF Processing Common Errors

There are cases when the PDF processing fails when trying to publish DITA content to a PDF file. This topic lists some of the common problems and possible solutions.

Problem: Cannot Save PDF

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\default\Desktop\bev\out\pdf\test.pdf
(The process cannot access the file because it is being used by another process)
```

Solution: Cannot Save PDF

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.

Problem: Table Contains More Cells Than Defined in Colspec

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
```

Solution: Table Contains More Cells Than Defined in Colspec

To resolve this issue, correct the `@colspec` attribute on the table that caused the issue. To locate the table that caused the issue:

1. Edit the transformation scenario and set the parameter `clean.temp` to `no`.
2. Run the transformation, open the `topic.fo` file in Oxygen XML Developer Eclipse plugin, and look in it at the line specified in the error message `(See position 179:-1)`.

Related Information:

How to Enable Debugging for FO Processor Transformations *(on page 749)*

DITA to CHM Transformation Fails - Cannot Open File

Problem

The DITA to CHM transformation fails with the following error: `[exec] HHC5010: Error: Cannot open "fileName.chm". Compilation stopped.`

Cause

This error occurs when the CHM output file is opened and the transformation scenario cannot rewrite its content.

Solution

To solve this issue, close the CHM help file and run the transformation scenario again.

DITA to CHM Transformation Fails - Compilation Failed

Problem

The DITA to CHM transformation fails with the following error: `[exec] HHC5003: Error: Compilation failed while compiling fileName`.

Cause 1

One possible cause for this error is that the processed file does not exist.

Solution 1

To solve this issue, fix the file reference before executing the transformation scenario again.

Cause 2

Another possible cause for this error is that the processed file has a name that contains space characters.

Solution 2

To solve the issue, remove any spacing from the file name and run the transformation scenario again.

Error: 'Address Family Not Supported by Protocol Family; Connect'

Problem

I have experienced the following error: *"Address Family Not Supported by Protocol Family; Connect"*. How do I solve it?

Cause

This seems to be an IPv6 connectivity problem. By default, the Java runtime used by Oxygen XML Developer Eclipse 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.

Solution

A quick solution 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 on a single line, add `-Djava.net.preferIPv4Stack=true`. Then save the file and copy it to the Oxygen XML Developer Eclipse plugin installation folder (may need admin access).
2. Restart Oxygen XML Developer Eclipse 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`.

Error After Switching Oxygen Products in Eclipse

Problem

On an Eclipse deployment, after installing and using the **Editor** product, I later decided to uninstall it and use **Author** or **Developer** instead. After installing the new product, I received an error that looked like this:

```
java.lang.Exception
      at org.eclipse.ui.internal.ViewReference.createErrorPart(ViewReference.java:112)
      at org.eclipse.ui.internal.ViewReference.createPart(ViewReference.java:98)
      at org.eclipse.ui.internal.e4.compatibility.CompatibilityPart.createPart
                                                          (CompatibilityPart.java:279)
      at org.eclipse.ui.internal.e4.compatibility.CompatibilityPart.create
                                                          (CompatibilityPart.java:317)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke
                                                (DelegatingMethodAccessorImpl.java:43)
      at java.lang.reflect.Method.invoke(Method.java:498)
...
```

Solution

You need to manually delete the perspective for the product you uninstalled by following this procedure:

1. Go to **Window > Preferences > General > Perspectives**.
2. Select the perspective for the product you uninstalled (for example, **<oXygen/> XML**) and click **Delete**.
3. Restart Eclipse.
4. Go to **Window > Open Perspective** and select the perspective for the new product (for example, **<oXygen/> XML Author**).

Hunspell Spell Checker is Unusable on Your Platform Error

Problem

When trying to use the **Check Spelling** option, I receive the error **Hunspell spell checker is unusable on your platform. It has crashed the application in a previous session**.

Cause

There are instances where Oxygen XML Developer Eclipse plugin determines that an internal component (such as the spell checker) has crashed the application and disables that component from running in the future (to prevent a possible future crash).

Solution

To re-enable the spell checker component, follow these steps:

1. Close Oxygen XML Developer Eclipse plugin.
2. Open the `%APPDATA%/com.oxygenxml` folder and look for a file called something like **HunspellCrashGuard*.txt**. Delete that file.
3. Restart Oxygen XML Developer Eclipse plugin.

High Resolution Scaling Issues

Problem

I encounter scaling detection issues in a high resolution display. For example, some GUI components are too small.

Cause

This sometimes happens when using multiple displays with different resolutions because the application cannot detect the correct scaling setting.

Solution

You can use the *com.oxygenxml.hidpi.scaling* custom system property *(on page 158)* to force a specific scaling setting. For example: `-Dcom.oxygenxml.hidpi.scaling=1.5` for 150% scaling.

High Resolution Scaling Issues on Linux

Problem

On Linux when bundled with Oracle OpenJDK 11 or newer, Oxygen XML Developer Eclipse plugin does not automatically scale high resolution images when using the system's scaling settings.

Cause

This happens because Java 11 (and higher) does not detect the system scaling setting for HiDPI displays on Linux operating system.

Solution

In the Oxygen XML Developer Eclipse plugin installation folder, create a new file named *custom_commons.vmoptions*. Inside the file, manually add `-Dsun.java2d.uiScale=2`. This command instructs Java to use 2x (200%) scaling.

Images Appear Stretched Out in the PDF Output

Problem

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.

Solution

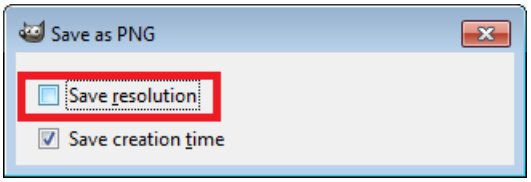
PDF output from XML content is obtained by first obtaining an intermediary XML format called XSL-FO and then applying an XSL-FO processor to it to obtain the PDF. This stretching problem is caused by the fact that all XSL-FO processors take into account the DPI (dots-per-inch) resolution when computing 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/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.

Increasing the Memory for the Ant Process

Problem

The Ant build process runs out of memory.

Solution

For details about setting custom JVM arguments to the Ant build process, see JVM Arguments.

Mac Touch Bar Function Keys Do Not Work

Problem

I am using a Mac that has a Touch Bar but its function keys do not work in Oxygen XML Developer Eclipse plugin.

Causes

By default, the Touch Bar function keys are not enabled for Oxygen XML Developer Eclipse plugin.

Solution

To enable the Touch Bar function keys for Oxygen XML Developer Eclipse plugin, follow these steps:

1. Go to **System Preferences** and select **Keyboard**.
2. Click **Shortcuts**.
3. From the left sidebar, select **Function Keys**.
4. Click the **+** symbol, select **Oxygen** from the list of apps, and click **Add**.

Mac OS X Keyboard no Longer Works After Inserting Accented Characters

Problem

Mac OS X has a method for inserting accented characters that is triggered by holding a key and then selecting the desired character from a contextual menu. When using this method in OS X 10.14 (Mojave) to insert accented characters in Oxygen XML Developer Eclipse plugin, the keyboard can no longer be used unless I restart the application.

Causes

This problem is caused by some sort of incompatibility issue with Java 8 and OS X 10.14 (Majave).

Solution

To prevent this problem, you should use an Oxygen XML Developer Eclipse plugin distribution that includes OpenJDK 12:

1. Uninstall Oxygen XML Developer Eclipse plugin:

a. All data stored in the installation directory will be removed, including any customizations or any other data you have stored within that directory. Therefore, you should make a backup of any data you want to keep.
b. Remove the application by manually deleting the installation folder and all its contents.

2. Reinstall Oxygen XML Developer Eclipse plugin using the distribution that includes OpenJDK:
   a. Go to the Oxygen Download page.
   b. In the **Mac OS X** tab, click on the **macOS 10.11 and later (Includes OpenJDK 12.0.1)** link to download this particular distribution.
   c. Follow the instructions to install the downloaded installation package.

Machine Signature Mismatch Error

Problem

I receive an error indicating that *the current license was already activated on a License Server* or that the *License Server's Machine Signature does not match*.

During the license activation process, the license key becomes bound to a particular license server deployment. This means that a code that uniquely identifies your license server deployment (called *Machine Signature*) is sent to the Oxygen XML Developer Eclipse plugin servers, which in turn will sign the license key. The *Machine Signature* is computed from the list of network interfaces of the machine where you deployed the license.

When starting the license server, if you receive an error stating that your server's Machine Signature does not match, there are several possible causes:

Possible Cause 1

The license key was moved to a new machine that hosts your license server.

Solution

Revert to your previous configuration.

Possible Cause 2

A new network interface was changed, added, or activated in the machine that hosts your license server.

Note: A specific example of when this could happen is if the Bluetooth or the WiFi module is activated/deactivated.

Solution

If reverting is not possible, contact the Oxygen support team.

Possible Cause 3

The license server was restarted from a different location as the previous restart. For example, some server configurations will have the Apache Tomcat server installed in a versioned folder (`/usr/local/apache-`

`tomcat-V.V.V`) with a symbolic link to the typical folder (`/usr/local/tomcat`). The server can be restarted from either location, but it is recommended to always restart from the typical folder (`/usr/local/tomcat`) and always restart from the same location.

Solution

The server simply needs to always be restarted from the same location.

MSXML 4.0 Transformation Issues

Problem

When running a transformation scenario that uses the MSXML 4.0 transformer, I receive an error that looks like this:

```
Could not create the 'MSXML2.DOMDocument.4.0' object.
Make sure that MSXML version 4.0 is correctly installed on the machine.
```

Cause

It is likely that the latest MSXML 4.0 service pack is not installed on your computer.

Solution

To fix this issue, go to the Microsoft website and get the latest MSXML 4.0 service pack.

Navigation to a Web Page is Canceled when Viewing CHM on a Network Drive

Problem

When viewing a CHM on a network drive, I only see the TOC and an empty page that displays the message: **Navigation to the web page was canceled**.

Cause

This is actually normal behavior. The Microsoft viewer for CHM does not display the topics for a CHM open on a network drive.

Solution

As a workaround, copy the CHM file on your local system and view it there.

References Outside the Main DITA Map Folder

Problem

A reference to a DITA topic, *map*, or binary resource (for example, an image) that is located outside of the folder where the main *DITA map (on page 1324)* is located leads to problems when publishing the content using the DITA Open Toolkit.

Cause

DITA-OT often has trouble resolving references 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.

Solution

To solve this, try one of the following solutions:

- Create another *DITA map* that is located in a folder path above all referenced folders and reference the original *DITA map* from this new map. Then transform this *DITA map* instead.
- Edit the transformation scenario and in the **Parameters** tab, change the value of the **fix.external.refs.com.oxygenxml** parameter to `true`. 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.

  Important: The **fix.external.refs.com.oxygenxml** parameter is only supported when the DITA-OT transformation process is started from Oxygen XML Developer Eclipse plugin.
- For PDF output, you can edit the transformation scenario and in the **Parameters** tab set the value of the **generate.copy.outer** parameter to `3`. This parameter specifies whether to generate output files for content that is not located in or beneath the directory containing the DITA map file. By setting the value of this parameter to `3`, the transformation scenario shifts the output directory so that it contains all output for the publication.

  Important: This method is recommended for transformation scenarios that use an external DITA-OT.

Syntax Highlights Not Available in Eclipse Plugin

Problem

I associated the `.ext` extension with Oxygen XML Developer Eclipse plugin in Eclipse but an `.ext` file opened with the Oxygen XML Developer Eclipse plugin plugin does not have syntax highlights.

Solution

Associate an extension with Oxygen XML Developer Eclipse plugin in Eclipse versions 3.6-3.8, 4.2-4.15 by following these steps:

1. Associate the `.ext` extension with the Oxygen XML Developer Eclipse plugin plugin:
   a. Open the **Preferences** dialog box *(on page 36)* 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 Eclipse 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 *(on page 36)* and go to **General > Content Types**.

b. Add `*.ext` to the **File associations** list for the **Text > XML > Oxygen XML Developer Eclipse plugin** content type.

3. Click the **OK** button in the Eclipse preferences dialog box.

Result: Now when an `*.ext` file is opened, the icon and the syntax highlights should be the same as for XML files opened with the Oxygen XML Developer Eclipse plugin plugin.

TocJS Transformation Does not Generate All Files for a Tree-Like TOC

Problem

The *TocJS* transformation of a *DITA map (on page 1324)* does not generate all the files needed to display the tree-like table of contents.

Solution

To get a complete 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 the *DITA-OT-DIR*`/plugins/com.sophos.tocjs/basefiles` folder to the transformation output folder.
3. Copy the *DITA-OT-DIR*`/plugins/com.sophos.tocjs/sample/basefiles/frameset.html` file to the transformation 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"`.

Windows Service Reports Incorrect Function Error When Starting (TCP Server)

Problem

My TCP floating license server reports an "Incorrect Function" error message when starting the Windows service.

Cause

This usually appears because the Windows service launcher cannot locate a Java virtual machine on your system.

Solution

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 Process Terminated Unexpectedly Error (TCP Server)

Problem

My TCP floating license server reports a "Process Terminated Unexpectedly" error message for the Windows service.

Cause

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).

Solution

Re-deploy your license key. For more information, see the Setting up a Floating License Server section *(on page 27)*.

XSLT Debugger Is Very Slow

Problem

When I run a transformation in the **XSLT Debugger** *perspective (on page 1327)*, it is very slow.

Solution

If the transformation produces HTML or XHTML output, you can disable rendering of output in the XHTML output view *(on page 131)* during the transformation process. To view the XHTML output result do one of the following:

- Run the transformation in the **Editor** *perspective (on page 1327)* and make sure the **Open in Browser/ System Application** option *(on page 701)* is selected.
- Run the transformation in the **XSLT Debugger** *perspective (on page 1327)*, save the text output area to a file, and use a browser application for viewing it (for example, Firefox or Internet Explorer).

19.

Glossary

Active Cell

Active cell refers to the selected cell where data is entered when you begin typing. Only one cell is active at a time. The *active cell* is bounded by a heavy border.

Anchor

An ***Anchor*** is used in various types of links to take the user to a specific location within the target document. It is designated in a URL or in the value of the `@href` attribute with a # symbol followed by the anchor that is defined in a target ID (for example `href="MyTopic.dita#anchor`).

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, paragraphs and list items 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.

Canonicalize

To ***canonicalize*** something means to convert it to a standard format that everyone generally uses. When using the term with regard to XML, it refers to the process of converting data that has more than one possible representations into a standardization that conforms to the specification of an XML document or document subset. It is helpful for applications that require the ability to test whether or not the content of an XML document or subset has been changed.

Content Completion Assistant

The ***Content Completion Assistant*** refers to a very helpful mechanism in Oxygen XML Developer Eclipse plugin that offers a list of proposed items that could be inserted at the current location, depending on the current context, editing mode, and type of document. It also tries to determine the most logical choice in the current editing context and displays that proposal at the beginning of the list.
For more information about this feature and how to invoke it, depending on your editing context, see the following:

- Content Completion Assistant in Text Mode *(on page 234)*
- Content Completion Assistant in Grid Mode *(on page 275)*
- Content Completion in XSLT Stylesheets *(on page 369)*
- Content Completion in XML Schema *(on page 448)*
- Content Completion in XQuery *(on page 486)*
- Content Completion Assistance in WSDL Documents *(on page 501)*
- Content Completion in CSS Stylesheets *(on page 522)*
- Content Completion in Relax NG Schemas *(on page 531)*
- Content Completion in NVDL Schemas *(on page 546)*
- Content Completion in JavaScript Documents *(on page 585)*
- Content Completion in Schematron Documents *(on page 594)*
- Content Completion in SQF *(on page 618)*

Dockable

A ***Dockable*** window is one that can be moved and resized, and either floated or pinned to a location, allowing you to configure the workspace according to your preferences.

Document Type Association

In general terms, a ***Document Type Association*** is a set of rules that associate a document type with a *framework (on page 1325)*. In Oxygen XML Developer Eclipse plugin, ***Document Type Association*** also specifically refers to a preferences page *(on page 50)* where you can create new custom *frameworks* or edit existing ones. Note that *frameworks* (document types) that come built-in with Oxygen XML Developer Eclipse plugin are read-only, but you can **Extend** *(on page 50)* or **Duplicate** *(on page 50)* them to configure them as custom *frameworks*.

DITA Map

A ***DITA map*** is a component of the DITA *framework (on page 1325)* that provides the means for 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 (on page 1323)* to generate a deliverable using an output type such as XHTML, PDF, HTML Help, or Eclipse Help.

DITA-OT-DIR

DITA_OT_DIR refers to the default directory that is specified for your DITA Open Toolkit distribution in the **Window > Preferences > Oxygen XML Developer Eclipse plugin > DITA** preferences page *(on page 47)*.

For example, if you are using DITA-OT 3.4.1 that comes bundled with Oxygen XML Developer Eclipse plugin, the default directory is: `[OXYGEN_INSTALL_DIR]/frameworks/dita/DITA-OT3.x`. You can also specify a custom directory.

Foldable Element

A ***foldable element*** refers to elements that can be collapsed and expanded in Oxygen XML Developer Eclipse plugin. *Foldable elements* are marked with a small triangle (/) on the left side of the editor panel and you can use that triangle to quickly collapse or expand them. This feature is helpful when you are working with large documents and you want to temporarily hide blocks of content. You can right-click the triangle to access additional collapse and expand actions (**Collapse Other Folds**, **Collapse Child Folds**, **Expand Child Folds**, **Expand All**).

Framework

A ***framework*** refers to a package that contains resources and configuration information to provide ready-to-use support for an XML vocabulary or document type. A *framework* is associated to an XML document type according to a set of rules. It also includes a variety of settings that improve editing capabilities for its particular file type. Oxygen XML Developer Eclipse plugin includes a **Document Type Configuration Dialog Box** *(on page 52)* that allows you to define the set of rules and customize various authoring mechanisms for new or existing *frameworks*.

IDML

IDML is an abbreviation for Adobe InDesign Markup files.

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

Java Archive (JAR) 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.

Key Space

The concept of a ***Key Space*** in DITA refers to a set of all possible keys that can be used in a *DITA map* structure. A ***Key Space*** is established when a *root map (on page 1328)* defines a set of effective key bindings.

Keystore

A ***Keystore*** is an encrypted file that contains private keys and certificates. There are two types of *keystores* that are supported in Oxygen XML Developer Eclipse plugin:

- **Java Key Store (JKS)**
- **Public-Key Cryptography Standards version 12 (PKCS-12)**

Master File

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. For more information, see the Master Files Support *(on page 200)* section.

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 for Oxygen XML Developer Eclipse 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 among 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* 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 Eclipse plugin End-User License Agreement.

Oxygen Publishing Template

Oxygen Publishing Template defines all the aspects related with the **look and feel(layout and styles)** for the **WebHelp Responsive** output.

The template is self-contained and packed as a ZIP archive making it easy t share with others. It represents the main method for customizing the *WebHelp Responsive* output.

Related Information:

Publishing Template Package Contents for WebHelp Responsive Customizations *(on page 819)*

Perspective

In Oxygen XML Developer Eclipse plugin, a ***perspective*** refers to an interface layout geared towards a specific editing environment. Each *perspective* includes a unique set of interface objects, toolbars, views, and features. You can change the *perspective* by selecting the respective icon in the top-right corner of Oxygen XML Developer Eclipse plugin or by selecting the *perspective* from the **Window > Perspective > Open Perspective** menu.

The *perspectives* that are available in Oxygen XML Developer Eclipse plugin are:

- **Editor** *(on page 161)* - The most commonly used *perspective* and it is used to edit XML documents.
- **XSLT Debugger** *(on page 162)* - Used to detect problems in an XSLT transformation by executing the process step by step in a controlled environment.
- **XQuery Debugger** *(on page 164)* - Used to detect problems in an XQuery transformation process by executing the process step by step in a controlled environment
- **Database** *(on page 165)* - Used to browse and manage databases.

Plugin

A ***plugin*** (also referred to as *add-on* or *extension*) is a component that adds specific features to an existing application. Oxygen XML Developer Eclipse plugin supports *plugins* to enable numerous customizations.

Pretty-Print

Pretty-print refers to formatting and indenting the source code in **Text** mode to make the content easier to view and analyze. The formatting actions that are available in Oxygen XML Developer Eclipse plugin include:

- **Format and Indent Element** - Available in the **Source** submenu of the contextual menu for the current element.
- **Format and Indent** - Available on the toolbar for the entire current document.
- **Format and Indent** - Available in the contextual menu of the **Text** view *(on page 702)* and applies to all the content in that view.
- **Format and Indent Files** - Available in the contextual menu of the **Navigator** view *(on page 192)* for one or more selected files.

QName

QName stands for "qualified name" and defines a valid identifier for elements and attributes. *QNames* are used as URI references to reference particular elements or attributes within XML documents.

Quick Assist

The ***Quick Assist*** feature gives you easy access to some of the most commonly used actions for the specific type of document you are editing. 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 in **Text** mode. You can also invoke the quick assist menu by using the **Ctrl + 1** (**Meta 1** on Mac OS X) keyboard shortcuts.

Quick Fix

The ***Quick Fix*** support in Oxygen XML Developer Eclipse plugin helps you resolve errors that appear in an XML document by offering proposals to fix problems such as missing required attributes or invalid elements. *Quick Fixes* are available in **Text** mode and they can be presented and activated in several ways.

- When hovering over an area of text where a validation error or warning occurs, the *Quick Fix* proposals can be presented as links 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. Clicking that icon will allow you select from the available proposals.
- If you place the cursor in the highlighted area where a validation error or warning occurs, you can also access the *Quick Fix* menu by pressing **Ctrl + 1 (Command + 1 on OS X)** on your keyboard.

Oxygen XML Developer Eclipse plugin also provides support for defining and customizing a library of *Quick Fixes* using the Schematron language *(on page 607)*.

Root Map

A ***Root Map*** (or master map) specifies a *DITA map (on page 1324)* that defines a hierarchical structure of submaps that are contained within the *root map*. Essentially, the *root map* defines a scope and provides the

mechanism to allow your defined keys to be propagated throughout the entire map structure (this mechanism is also known as a *key space (on page 1326)*).

WebHelp Output Directory

WebHelp_OUTPUT_DIR refers to the output directory where WebHelp transformation files will be generated.

The output directory can be specified using the **Output Directory** text field in the **Output** tab of the transformation scenario dialog box.

When running the WebHelp transformation from a command line, the output directory can be specified using the `-o` or `--output` option.

Working Set

A ***Working Set*** refers to a set of files that will be used for the scope of search and refactoring operations. Many of the search and refactoring wizards include a step where you can specify the scope for the operation and you can choose one or more *working sets* to restrict the scope to that specified set of files.

XML Catalog

An ***XML Catalog*** maps a system ID or a URI reference for a resource (stored either remotely or locally) to a local copy of the same resource. Whenever 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 connection is slow.

Oxygen XML Developer Eclipse plugin includes default global catalogs as well as default catalogs for each of the built-in *frameworks (on page 1325)*, and you can also create your own. Oxygen XML Developer Eclipse plugin uses these *XML Catalogs* to resolve references for document validation and transformations. For more information, see Working with XML Catalogs *(on page 312)*.

Index

A

Add fonts to built-in FO processor
746, 747
Add hyphenation libraries to built-in DITA-OT FO processor
1116
Add hyphenation libraries to built-in FO processor
748
Add PDF image support to built-in DITA-OT FO processor
1116
Add PDF image support to built-in FO processor
748
Add Schematron Quick Fix
607
Add-on
1327
Add-ons
1327
Annotation preferences
79
Ant transformation scenario
729
  Options tab
  730
  Output tab
  731
  Parameters tab
  731
Archive preferences
37
Archives
1126
  Browse
  1126
  Edit files
  1130
  EPUB
  1128
  File browser
  1126
  Migrate OOXML to DITA
  1130
  Migrate OOXML to TEI
  1130
  Modify
  1126
  ODF
  1128
  OOXML
  1128
Associate schema directly in JSON documents
562
Associate schema directly in XML documents
310
Associate schema in a framework configuration
311
Associate schema in a validation scenario defined in framework
308
Associate schema through a validation scenario
305, 560
Associate schema to a JSON document
560
Associate schema to an XML document
303
Attributes view in Design mode
452
Attributes view in Text mode
243
Author Action dialog box
59
Author editing mode
  Entities view
  247
  MathML equations in HTML output
  905
  Model view
  245
Automatic Spell Check

213
Automatic validation
279
Automatic validation in JSON
554

B

BaseX database connection
1176
BaseX database contextual menu actions
1177
BaseX XQJ connection
1179
Batch transformation
769
Berkeley DB XML database connection
1153
Berkeley DB XML database contextual menu actions
1155
Berkeley DB XML debugging
1159, 1193
Bidirectional text in Grid mode
275
Built-in frameworks
DITA Map
659
DITA Topic
658
DocBook 4
646
DocBook 5
650
DocBook 5.1
650
DocBook 5.1 Assembly
655
DocBook 5.1 Topic
656
DocBook Targetset Map
657
EPUB
665
JATS
664
jTEI
663
TEI ODD
662
TEI P5
661
XHTML
660
Built-in XML refactoring operations
329, 1241

C

Canonicalize files tool
359, 1296
Certificates preferences
127
Change file permissions on remote FTP server
186
Change language for interface
159
Changing the font size in Text mode
225
Character Map dialog box
217
Check Spelling
205
Check Spelling in multiple Files
214
Check Well-Formedness action
277
Checking Well-Formedness in JSON
553
CHM Error HHC5003: Compilation failed while compiling file
1312, 1313
CHM Error HHC5010: Cannot open file
1312, 1313
Close file
183
Code template preferences
102

Code templates
238
Common problems and solutions
1306
Compile XSL Stylesheet for Saxon tool
404, 1272
Configure Calabash with XEP
753
Configure the Application
scenarios management
111
Configure transformation scenario
762
Configure Transformation Scenario dialog box
765
Configuring options
145
Configuring Oxygen
36
Console view
220
Console view preferences
112
Content Completion Assistant in Grid Mode
275
Content completion helper views
238
Content completion in HTML
624
Content completion in JSON
563
Content completion in Text Mode
234
Content completion preferences
77
Contextual menu actions in JSON
577
Contextual menu actions in Text Mode
258
Contextual menu of current editor tab
189
Convert database to XML Schema
478, 1272
Convert JSON to XML
570, 1274
Convert schema to another schema language
475, 1269
Convert XML to JSON
573, 1277
Convert XSD to JSON Schema
568, 1280
Copy/Paste in Grid Mode
273
Create DTD from learned document structure
312
Create new transformation scenario
691
Create new validation scenario
288
Create Schematron Quick Fix
607
Creating Markdown documents
629
Creating new documents
171
Creating publishing templates
848, 968
CSS processors preferences
118
CSS stylesheets
Content completion
522
Custom CSS properties
522
Editing features
521
Folding
524
Format and indent (pretty print)
524
Minifying
524
Outline view
523

Syntax highlighting
523
Validation
522
CSS validation preferences
38
Current license was already activated
31, 1318
Custom editor variables
39
Custom system properties
155
Custom validation engine preferences
82
Custom XML refactoring operations
337, 1249
Custom XSLT/XQuery transformation engine preferences
130
Customize document templates
179
Customizing default options
145

D

Data Source Explorer view
1131
Data Sources preferences page
40
Database connection preferences
40
Database drivers
45
Database perspective
165
Databases
1131
Connections
1135
BaseX
1176
Contextual menu actions
1177
XQJ
1179
Berkeley DB XML
1153
Contextual menu actions
1155
Debugging
1159, 1193
eXist
1159
Connection wizard
1160
Contextual menu actions
1161
Manual configuration
1160
Generic JDBC
1174
IBM DB2
1136
Contextual menu actions
1139
JDBC-ODBC
1175
MarkLogic
1163
Contextual menu actions
1170
Debugging
1167, 1190
MarkLogic development
1166
Microsoft SQL Server
1140
Contextual menu actions
1143
MySQL
1171
Oracle
1144
Contextual menu actions
1146

PostgreSQL
1149
Contextual menu actions
1152
SharePoint
1193
Contextual menu actions
1194
WebDAV
1181
Contextual menu actions
1181
Data Source Explorer view
1131
SQL support
1183
Table Explorer view
1133
XQuery
1186
Debugging
1190
Berkeley DB XML
1159, 1193
MarkLogic
1167, 1190
Drag and drop from Data Source Explorer
1186
Transformations
1187
Validation
1187
Debug PDF transformation
1116
Debugger preferences
131
Debugging XQuery
1210
Breakpoints
1229
Debugging Java extensions
1235
Identify expressions
1228
Information views
1215
Breakpoints view
1216
Context view
1217
Messages view
1219
Nodes/Values Set view
1224
Output Mapping Stack view
1220
Stack view
1220
Templates view
1223
Trace view
1222
Variables view
1225
XPath Watch view
1218
Layout
1211
Performance profiling
1230
Profiling
Hotspots view
1233
Invocation Tree view
1232
Steps in a typical debugging process
1227
Toolbar
1212
Debugging XSLT
1210
Breakpoints
1229
Debugging Java extensions

1235
Debugging XSLT that call Java extensions
1235
Identify expressions
1228
Information views
1215
Breakpoints view
1216
Context view
1217
Messages view
1219
Nodes/Values Set view
1224
Output Mapping Stack view
1220
Stack view
1220
Templates view
1223
Trace view
1222
Variables view
1225
XPath Watch view
1218
Layout
1211
Performance profiling
1230
Profiling
Hotspots view
1233
Invocation Tree view
1232
Steps in a typical debugging process
1227
Supported processors
1236
Toolbar
1212
Debugging XSLT that call Java extensions
1235
Design editing mode
Attributes view
452
Components
419
Grouping components
443
xs:all
436
xs:alternative
432
xs:any
437
xs:anyAttribute
438
xs:assert
442
xs:attribute
424
xs:attributeGroup
426
xs:choice
436
xs:complexType
427
xs:element
420
xs:field
441
xs:group
433
xs:import
434
xs:include
434
xs:key
440
xs:keyRef
440
xs:notation

435
xs:openContent
443
xs:override
435
xs:redefine
435
xs:schema
419
xs:selector
441
xs:sequence
436
xs:simpleType
430
xs:unique
439
Contextual menu actions
411
Editing actions
410
Facets view
409
Navigation
407
Outline view
450
Palette view
408
Design mode preferences
87
Detect Master Files
201
Detect Master Files from Project
201
Diff appearance preferences
47
DIFF preferences
45
Digital Signature
Canonicalize files
359, 1296
Certificates
358
Sign files
360, 1298
Verify signature
362, 1300
Digital Signatures
356
Example of how to digitally sign XML content
362
Overview
356
DITA and Markdown documents
633
DITA Logging preferences
49
DITA Map document type
659
DITA Map PDF - based on HTML5 & CSS
679
Abbreviated-Form
1106
Abbreviated-form element
1100
Accessibility
1069, 1069
Back matter
1024
Style topics
1026
Bookmap styling
1103
Bookmarks
1050
Change labels
1051
Initial state
1052
Limit section depth
1051
Remove numbering
1052

Code Blocks
1097
Comments
1073
Styling
1079
Cover page
1017
Add empty pages
1021
Add text
1020
Background image
1018
Copyright page
1022
Place cover on left side
1021
Place cover on right side
1021
Title styling
1020
Custom transformation parameters
1101
Customization CSS
981
Debugging
982, 990
XPath expressions
985, 986
Deep numbering
1031
Double sided pagination
1047
Force even number of pages
1049
Force odd number of pages
1049
Start chapters on odd page
1048
Style blank pages
1048
Style first page
1049
Draft watermarks
1080
Conditional draft watermark
1080, 1080
Error parsing
1106
Figures
Figure numbering
1092
Flagging content
1081
Fonts
1069, 1070
Asian languages
1071, 1071
Linux
1071
Footer
992
Add copyright
998
Add topics
999
Footnotes
1061
Reset Counter
1062
Style Markers and Calls
1061
Front matter
1024
Page breaks
1026
Style topics
1026
Glossentry
1106
Glossgroup
1106
Header

992
Add background image
1000
Add links
1001
Change header at each chapter
995, 996
Change separators
1004
Only keep chapter title
1003
Style text
1002
Styling
1005
Underlined header
1007, 1046, 1047
XPath
1005
Header and Footer
Font
994
Size
994
Hyphenation
1065
Enable or disable for tables
1068
Entire map
1067
I/O exception
1106
i18n
1073
Images
1088
Centering
1091
Control image size
1090
Resolution
1089
Rotate
1089
SControl image size
1090
Side by side with text
1089, 1089
Size
1088
Wide
1089
Index
1052
Add a leader
1057
Change number format
1058
Change style and letters
1056
Style labels
1057
Table style layout
1059
Links
1087
List of Figures
1046
List of Tables
1046
Localization
1073
MathML equation customization
1084
Metadata
1008
Changing the keywords
1015
Changing the title property
1015
Cover page
1012
Custom
1011

Footer
1014
Header
1014
Key values
1016
Keywords
1015
Title property
1015
Use key value in CSS
1016
Notes
1100
Numbering
1027, 1031
Continuous numbering
1034
Remove prefix
1033
Page breaks
1062
Add a blank page after a topic
1064
Avoid in lists and tables
1062
Enforce number of lines
1064
Force before or after topic
1063
Page size
Change setting for an element
991
Changing
990
Orientation
991
Styling
1082
Table of contents
1035
Change header
1038
Display short description in TOC
1040
Display topic before TOC
1040
Increase depth
1037
Remove TOC entries
1040
Start on odd page
1039
Style
1038
Tables
1093
Automatic layout
1095
Layout
1094
Rotate
1093
Text bleeding
1094
Wide
1093
Tracked changes
1073
Hide footnotes
1079
Styling
1079
Trademarks
1101
Translation
1073
Unknown host
1106
DITA Map to CHM (Compiled Help) transformation
682
DITA Map to Kindle transformation
683
DITA Map to MS Office Word transformation

681
DITA Map to PDF transformation
681
DITA Map to WebHelp Responsive transformation
672
DITA Map transformations
671
DITA Maps
Transformation scenarios
671
DITA new topics preferences
48
DITA PDF - based on HTML5 & CSS
Comments
Styling
1079
Custom transformation parameters
1101
Customization CSS
981
Debugging
982
Figures
Figure numbering
1092
i18n
1073
Localization
1073
Tracked changes
Styling
1079
Translation
1073
DITA preferences
47
DITA publishing preferences
49
DITA to Word
681
DITA topic
Transformation scenarios
685
DITA Topic document type
658
DITA topic transformations
685
DITA-OT preferences
47
DITA-OT transformation scenario
716
Advanced tab
726
Feedback tab
725
Filters tab
725
FO Processor tab
722
Output tab
727
Parameters tab
724
Skins tab
717
Templates tab
718
DocBook 4 document type
646
DocBook 5 document type
650
DocBook 5.1 Assembly
655
DocBook 5.1 document type
650
DocBook 5.1 Topic document type
656
DocBook olink
647, 651
DocBook Targetset Map document type
657
DocBook to DITA transformation
689
DocBook to EPUB transformation

690
DocBook to PDF output customization
1117
DocBook to PDF transformation
690
DocBook to WebHelp transformation
686
DocBook transformations
686
Document template preferences
107
Document templates
Creating
177, 178
Customizing
179
Sharing
182
Document type association preferences
50
Document type configuration dialog box
52
Association rules tab
53
Author tab
56
Actions subtab
57
Content Completion subtab
69
Contextual Menu subtab
67
CSS subtab
56
Menu subtab
66
Toolbar subtab
69
Catalogs tab
73
Classpath tab
55
Extensions tab
76
Schema tab
55
Templates tab
72
Transformation tab
73
Validation tab
74
Document Types
646
Drag and Drop in Grid Mode
272
Drag and Drop in Text Mode
232
DTD Entities
321, 321
Duplicate transformation scenario
764

E

Edit mode preferences
85
Edit validation scenario
293
Editing actions in Grid Mode
270
Editing Modes
168
Design
169, 406
Grid
168
Text
168
Editing publishing templates
850, 970
Editing XML markup in Text Mode
227
Editing XSLT
Compile XSL Stylesheet for Saxon tool
404, 1272

Editor perspective
161
Editor preferences
76
Editor variables
148
Custom editor variables
155
Elements view in Text mode
247
Entities view
247
EPUB document type
665
Error: 1248 WARN
org.apache.fop.apps.FOUserAgent
1311
Error: Anttask - Error rendering fo file
1311
Error: Could not create MSXML object
1319
Error: Incorrect Function Windows service error
32, 1321
Error: java.io.FileNotFoundException
1311
Error: Navigation to the web page was canceled
1319
Error:
org.eclipse.ui.internal.ViewReference.createErrorPart
1314
Error: OutOfMemory
1305, 1305
Error: Process Terminated Unexpectedly
33, 1321
Error: ViewReference.createErrorPart
1314
Executing SQF in Other Documents
614
eXist database
Connection wizard
1160
Manual configuration
1160
eXist database connection
1159
eXist database contextual menu actions
1161
Export actions
57
Export Global Options
146
Export Global Transformation Scenarios
147
Export Global Validation Scenarios
147
Export to Excel
276
Exporting Markdown documents
627
Extension points
1237

F

Facets view in Design mode
409
File properties
190
Find All Elements action
203
Find All Elements dialog box
203
Finding/Replacing text
Find All Elements dialog box
203
Flatten Schema tool
478
Floating license server
18
FO processor preferences
115
Folding in Text Mode
231
Font preferences
108
Font size configuration in Text mode

225
FOP Error
1310, 1311
Format and Indent Files tool
256
Format and Indent in Text Mode
251
Format/Indent multiple files
256
Formatting preferences
89
Formatting Schematron Quick Fix Content
614
Framework directory locations
51
Frameworks
646
FTP connection settings
109

G

Generate Documentation tools
1274, 1285
Generate JSON Schema tool
581, 1284
Generate Sample JSON Files tool
575, 1283
Generate Sample XML Files tool
461, 1264
Generate WSDL Documentation tool
514, 1293
Generate XML Schema Documentation tool
466, 1285
  Customize PDF output for DITA
  475
  Customize PDF output for DocBook
  474
  Output formats
  470
Generate XSLT Stylesheet Documentation tool
398, 1289
  Custom format
  403
  HTML format
  401
Generic JDBC database connection
1174
Getting familiar with the interface
2
Getting Started
2
Getting Started with Oxygen
  Help
  6
  Interface
  2
  Resources to help you with Oxygen
  4
  Supported document types
  3
Grid editing mode
168, 268
  Bidirectional text
  275
  Content Completion Assistant
  275
  Copy/Paste
  273
  Drag and Drop
  272
  Editing actions
  270
  Export to Excel
  276
  Special characters
  275
Grid editing modes
  Layout
  268
  Navigation
  269
Grid mode preferences
85

H

Help

Randomize XML text content
8
Support related resources
6
Using the Help menu
7
Help resources
952
Highlight ID occurrences in Text Mode
258
Highlights
220
HTML documents
Content completion
624
Editing features
621
Folding
625
Minification
625
Minifying
625
Outline view
626
Syntax highlighting
624
Validation
623
HTTP authentication schemes
189
HTTP connection settings
110
HTTP floating license server
Replacing
26
Upgrading
27
HTTP license server
19
Management and Statistics
23
HTTPS troubleshooting
186
I
IBM DB2 database connection
1136
IBM DB2 database contextual menu actions
1139
Image not found
1092
Import csv files
1198
Import data dynamically
1206
Import Global Options
146
Import Global Transformation Scenarios
147
Import Global Validation Scenarios
147
Import preferences
112
Import txt files
1198
Importing data
1198
Importing data from a database
1203
Importing data from Excel
1200
Importing data from HTML files
1206
Importing Data from text files
1198
Increase memory
955
Indenting Schematron Quick Fix Content
614
Installing Oxygen
10
Floating license servers
Machine signature does not match
31, 1318

Group deployment
12
Licensing
13
TCP floating license server
All Platforms distribution
29
Replacing
30
Upgrading
31
Windows 32-bit
27
Incorrect Function error
32, 1321
Process Terminated Unexpectedly error
33, 1321
Replacing
29
Upgrading
29
Transfer license key
18
Upgrading
33
Integrate Schematron Quick Fixes in a framework
616
Integrate Schematron rules in a framework
592
Introduction
1
J
JATS document type
664
Java system properties
155
JavaScript documents
583
Content Completion
585
Editing features
583
Outline view
587
Syntax highlighting
586
Validation
585
JDBC-ODBC database connection
1175
JSON documents
550
Associate schema
560
Associate schema directly in JSON documents
562
Associate schema through a validation scenario
560
Content completion
563
Contextual menu actions
577
Folding
565
Grid mode editing features
551
JSON to XML converter
570, 1274
Outline view
565
Presenting validation errors
555
Schema annotations
565
Syntax highlighting
565
Text mode editing features
550
Validating JSON schema
580
Validation
552
Against a schema
553

Automatic validation
554
Check Well-formedness
553
Manual validation
554
Resolving references to remote schemas
559
Sharing scenarios
559
Validation scenarios
557
XML to JSON converter
573, 1277
XSD to JSON Schema converter
568, 1280
JSON schema
553, 580
JSON Schema
580
JSON to XML tool
570, 1274
JSON validation scenarios
557
jTEI document type
663

K

Kerberos
189

L

License key configuration
37
License options
37
Licensing Oxygen
13
Floating license
Reserve floating license
17
Floating licenses
15
Named-user license
14
Release a floating license
16
Request a floating license
HTTP license server
15
TCP license server
16
Localizing SQF messages
615
Localizing the user interface
159
Localizing XML refactoring operations
355, 1263
Locking a floating license
17

M

Mac accented characters
1317
Mac Function Keys
1317
Mac restart
1317
Mac text input freeze
1317
Mac Touch Bar
1317
Machine signature does not match
31, 1318
Manual validation actions
279
Manual validation in JSON
554
Markdown documents
626
Actions in Markdown Editor
629
Creating Markdown documents
629
DITA
633
Markdown Editor

627
Rules and specifications
634
Syntax highlighting
632
Validation
632
Markdown Editor
627
Markdown preferences
109
MarkLogic database connection
1163
MarkLogic database contextual menu actions
1170
MarkLogic debugging
1167, 1190
MarkLogic for the developer
1166
Master Files
200
Add files
202
Detecting
201
Enabling
201
Overview
200
MathML equations in HTML output in Author mode
905
MathML equations in WebHelp output
876, 938
Memory issues
955
Microsoft SQL Server database connection
1140
Microsoft SQL Server database contextual menu actions
1143
Model view
245
Modular XML files
315
Move XML resources
320
MSXML 3.0 and 4.0 preferences
137
MSXML.NET preferences
138
MySQL database connection
1171

N

Navigating in Text Mode
223
Navigator view
192
Navigator view preferences
112
Network connection preferences
109
New document from templates
177, 178
New Document Wizard
171
New from Templates Wizard
177, 178
New project
6, 191
NISO Journal Article Tag Suite document type
664
Non-XML files
190
NTLM
189
NVDL schemas
542
Component Dependencies view
547
Content completion
546
Diagram Editor
543

Contextual menu actions
545
Full Model view
543
Introduction
543
Logical Model view
544
Outline view
547
Refactoring actions
549
Search actions
549
Syntax highlighting
547
Validation
546

O

Olink element
647, 651
Open file in system application
183
Open preferences
97
Open remote document
183
Open using FTP/SFTP/WebDAV dialog box
184
Opening file
182
Options Menu
Preferences
36
Oracle database connection
1144
Oracle database contextual menu actions
1146
Out of memory
955
Out Of Memory error
115
Outline view
240
Outline view in Design mode
450
Outline view preferences
144
OutOfMemoryError
955
oxy:allows-child-element function
62
oxy:allows-global-element function
64
oxy:current-selected-element function
65
oxy:is-required-element function
65, 66
oxy:platform function
66
oxy:selected-elements function
65

P

Palette view in Design mode
408
PDF output
951
PDF Output Customization
DocBook to PDF output
1117
PDF output using Calabash XProc processor
753
PDF Processors
952
Performance preferences
97
Perspective
XML
161
Perspectives
161
Database
165
XQuery Debugger

164
XSLT Debugger
162
PostgreSQL database connection
1149
PostgreSQL database contextual menu actions
1152
Preferences
36
Archive
37
Code Templates
102
Content Completion
77
Content Completion annotations
79
CSS formatting
91
CSS Processors
118
CSS Validator
38
Custom Editor Variables
39
Custom Engines
130
Custom Validation Engines
82
Data Sources
40
Debugger
131
DIFF
45
Diff Appearance
47
DITA
47
DITA Logging
49
DITA New topics
48
DITA publishing
49
Document Templates
107
Document Type Association
50
Document type configuration dialog box
52
Document Type locations
51
Document Validation
84
Edit Modes
85
Editor
76
FO Processor
115
Fonts
108
Format
89
FTP/SFTP
109
Grid mode
85
HTTP(S)/WebDAV
110
Import
112
JavaScript formatting
92
Mark Occurrences
96
Markdown
109
MSXML XSLT
137
MSXML.NET XSLT
138
Network Connection Settings

109
Open/Save
97
Relax NG parser
124
Sample XML Files Generator
119
Saxon-HE/PE/EE XQuery
134
Saxon-HE/PE/EE XQuery Advanced
136
Saxon-HE/PE/EE XSLT
140
Saxon-HE/PE/EE XSLT Advanced
143
Saxon6 XSLT
143
Schema Design mode
87
Schematron parser
124
Spell Check
98
Spell Check Dictionaries
100
Syntax Highlight
101
Templates
102
Text Diagram
88
Trusted Hosts
110
View
112
Whitespaces
95
XML
112
XML Catalog
121
XML formatting
92
XML Parser
123
XML Refactoring
127
XML Schema parser
126
XML Signing Certificates
127
XML Structure Outline
144
XPath
133
XPath Content Completion
80
XPath formatting
96
XProc
128
XQuery
134
XQuery formatting
96
XQuery Profiler
132
XSD Content Completion
81
XSLT
137
XSLT Content Completion
81
XSLT Profiler
132
XSLT-XQuery
129
XSLTProc
143
Presenting validation errors in JSON
555
Problems and solutions
1306
Process overview

953
Projects
191
Adding items to projects
6, 191
Creating new projects
6, 191
Managing resources
192
Master files
200
Add files
202
Detecting
201
Enabling
201
Overview
200
Move resources
198
Navigator view
192
Rename resources
198
Publishing
669

Q

Quick Assist feature in Text Mode
257
Quick fix support in XML documents
301

R

Read-only files
220
Rectangular Selection feature
232
Refactoring XML Documents
325, 1238
Regular expressions syntax
204
Relax NG schemas
525
Component Dependencies View
538
Content completion
531
Custom datatype library
542
Diagram Editor
526
Contextual menu actions
529
Full Model view
526
Introduction
526
Logical Model view
527
Symbols
528
Editing in Master Files context
526
Moving resources
537
Outline view
533
Quick Assist feature
541
Refactoring actions
539
Renaming resources
537
Resource Hierarchy/Dependencies view
535
Search actions
539
Syntax highlighting
531
Validation
530
Release a floating license
16
Rename XML resources

320
Request a floating license from a TCP license server
16
Request a floating license from an HTTP license server
15
Reserving a floating license
17
Reset Global Options
146
Resolve schemas through XML catalogs
315
Resource Hierarchy/Dependencies view for XML documents
318
Restricting Schematron Quick Fix operations
613
Results view
248
Run DITA-OT Integrator transformation
684

S

Sample XML File Generator preferences
119
Save file
183
Save preferences
97
Save remote document
183
Save template as button
720
Saxon 6 XSLT preferences
143
Saxon HE/PE/EE advanced XQuery preferences
136
Saxon HE/PE/EE advanced XSLT preferences
143
Saxon HE/PE/EE XQuery preferences
134
Saxon HE/PE/EE XSLT preferences
140
Schema annotations in JSON
565
Schema annotations in Text Mode
237
Schema Design mode
169, 406
Schema Diagram Editor
169, 406
Schematron
  Editing features
  589
  Editing in Master Files context
  591
  Integrating in a framework
  592
  Sharing
  592
  Unit test (XSpec)
  604
  Validation
    Against Schematron
    591
Schematron Quick Fix Operations
  Add
  609
  Delete
  609
  Replace
  609
  String Replace
  609
Schematron Quick Fixes
303, 606, 606
  Content Completion
  618
  Customizing
  607
  Defining
  607
  Embedded
  620

Executing SQF in Other Documents 614
Formatting and Indenting 614
Highlight occurrences 618
Integrating in a framework 616
Localization 615
Multiple similar quick fixes 615
Refactoring actions 619
Restricting operations 613
Search actions 619
Sharing 616
SQF Operations 609
Use-When Condition 613
User Entry operation 613
Validation 618
Schematron schemas
Content completion 594
Syntax highlighting 595
Schematron Schemas
Embedded 300, 595
Highlight occurrences 601
Moving resources 600
Outline view 597
Quick Assist feature 603
Refactoring actions 601
Renaming resources 600
Resource Hierarchy/Dependencies view 598
Search actions 601
Validation 594
Search actions for IDs 316
Searching documents 202
Selecting content in Text Mode 232
Set indent to zero in Text Mode 255
SFTP connection settings 109
Share transformation scenarios 770
Share validation scenarios 298
SharePoint contextual menu actions 1194
SharePoint database connection 1193
Sharing document templates 182
Sharing JSON validation scenarios 559
Sharing publishing templates 852, 971
Sharing Schematron Quick Fixes 616
Sharing Schematron rules 592
Sharing XML refactoring operations 354, 1262

Shortcut actions in Text Mode
225
Show change tracking/comments in DocBook PDF
691
Sign files tool
360, 1298
Signing XML document preferences
127
Simple text editor
190
Single sign-on
189
Smart Editing in Text Mode
224
Special character preferences
97
Special characters
215
Character Map
217
Fallback Font Support
216
Inserting symbols
217
Unrecognized characters
216
Unsupported characters
216
Special characters in Grid mode
275
Spell check preferences
98
Spell checking
205
Add dictionaries
208
Add term lists
210
Custom list of terms
210
Customizing dictionaries and term lists
207
Dictionaries and term lists
207
Forbidden words
210
Ignored words
213
Learned words
212
Multiple files
214
Replace dictionaries
211
Spell Checking
Automatic spell check
213
SQF
303, 606, 606
Content Completion
618
Customizing
607
Defining
607
Embedded
620
Executing SQF in Other Documents
614
Formatting and Indenting
614
Highlight occurrences
618
Integrating in a framework
616
Localization
615
Multiple similar quick fixes
615
Operations
609
Refactoring actions
619
Restricting operations

613
Search actions
619
Sharing
616
Use-When Condition
613
User Entry operation
613
Validation
618
SQF attribute: use-for-each
615
SQF multi-lingual support
615
SQL transformation scenario
762
StackOverflowException error
84
Status information
220
Storing XML refactoring operations
354, 1262
StratML documents
Editing features
582
Supported document types
667
Supported processors for XSLT/XQuery debugging
1236
Syntax highlight preferences
101
Syntax highlighting in Text Mode
250
Syntax highlights in Text mode
251
System properties
155

T

Table Explorer view
1133
TEI ODD document type
662
TEI P5 document type
661
Templates tab
718
Text Diagram preferences
88
Text editing mode
168, 222
Attributes view
243
Content completion
234
Contextual menu actions
258
Drag and Drop
232
Editing XML markup
227
Elements view
247
Entities view
247
Folding
231
Format and indent
251
Format and indent multiple files
256
Highlight ID occurrences
258
Model view
245
Navigation
223
Rectangular selection
232
Schema annotations
237
Selecting content
232
Set indent to zero

255
Shortcut actions
225
Smart Editing
224
Syntax highlighting
250
Syntax highlights
251
Validation errors
281
Views
240
Touch Bar on Mac
1317
Transform DITA document to MS Word
681
Transformation parameters
956
Transformation Scenarios
669
Ant
729
Apply transformation scenarios
764
Batch transformation
769
Built-in transformation scenarios
670
Configure
762
Configure Transformation Scenario dialog box
765
Configure XSLT processor extension paths
706, 745
Custom XSLT processor
706, 745
Debugging PDF transformations
1116
DITA map
671
DITA Map PDF - based on HTML5 & CSS
679
DITA Map to CHM (Compiled Help
682
DITA Map to Kindle
683
DITA Map to MS Office Word
681
DITA Map to PDF
681
DITA Map to WebHelp Responsive
672
DITA topic
685
DITA-OT
716
DocBook
686
DocBook PDF - Show Change Tracking and Comments
691
DocBook to DITA
689
DocBook to EPUB
690
DocBook to PDF
690
DocBook to WebHelp
686
Duplicate
764
Integrate external XProc engine
753
New
691
Run DITA-OT Integrator
684
Sharing
770
SQL
762
Supported XSLT processors
703, 742

XML to PDF with CSS
714
XML with XQuery
707
XML with XSLT
691
XProc
749
XQuery
754
XSLT
732
Transformation Scenarios view
770
Transformation types
762
Transforming Documents
669
Trusted hosts connection settings
110

U

Unicode support
215
Uninstalling Oxygen
35
Upgrading Oxygen
33
Use-When SQF Condition
613
User Entry SQF operation
613
Using a floating license offline
17

V

Validating JSON Documents
Automatic validation
554
Check Well-formedness
553
Manual validation
554
Resolving references to remote schemas
559
Sharing scenarios
559
Validating JSON Documents against a schema
553
Validating JSON schema
580
Validating XML Documents
277
Against a schema
279
Against a schema with embedded Schematron
300, 595
Against Schematron
591
Automatic validation
279
Check Well-formedness
277
Create new validation scenarios
288
Custom validators
283
Customizing error messages
283
Edit validation scenarios
293
Manual validation
279
Resolving references to remote schemas
298
Sharing scenarios
298
Text Mode
281
Validating XSLT that call Java extensions
366
Validation engine 'StackOverflowException' error
84
Validation errors in Text mode
281
Validation preferences

84
Verify Signature tool
362, 1300

W

WebDAV connection
1181
WebDAV connection settings
110
WebDAV contextual menu actions
1181
WebDAV over HTTPS
186
WebHelp Classic
914
Add custom HTML content
929
Add Favicon
935
Add logo image
936
Add videos in DocBook
936
Adding additional resources
937
Changing icons in the table of contents
933
Customize highlights in the table of contents
934
Customize ordered lists
932
Disable caching
950
Edit scoring in search results
938
Indexing Japanese content
939
Integrate Facebook
942
Integrate Google Analytics
946
Integrate Google Search
947
Integrate Twitter
944
Layout and features
915
Localize email notifications
940
Localizing the interface for DocBook transformations
941
Publishing on SharePoint Site
950
Right-to-Left languages
942
Skin Builder
927
Use custom CSS
928
WebHelp Classic with Feedback
Deploying
920
Installing
920
WebHelp feature matrix
774
WebHelp Responsive
774
Adding audio objects
874
Adding favicon
873
Adding logo
872
Adding publishing templates to gallery
850
Adding videos
874
Adding welcome message
867
Built-in templates
817
Changing numbers for ordered lists
863

Changing scoring values in search results
876
Comments section
787
Context-sensitive help
788
Converting old publishing templates
853
Converting old templates to version 20
854
Converting old templates to version 21
855
Converting publishing templates to version 22
853
Converting version 20 publishing templates to version 21
853
Copy resources to output directory
872
Creating publishing templates
848, 968
CSS styling
855
Custom templates
817
Customizing main page layout
865
Customizing output with CSS
855
Customizing output with HTML content
857
Customizing the footer
868
Customizing the menu
865
Customizing the tiles
870
Deploying on SharePoint
904
Edit link to launch Web Author
901
Editing publishing templates
850, 970
Excluding topics from search results
877
Facebook Like button
884
Flagging DITA content
903
Google Analytics
888
Google Search
890
Index terms layout page
784
Inserting HTML
857
Localizing interface
881
Main layout page
776
Navigation links
865
Optimizing Japanese content indexing
880
Optimizing search results
878
Previous/Next arrows
865
Publishing template descriptor file
819
Publishing template HTML fragment extensions
825
Publishing template HTML page layout files
830
Publishing template package
819
Publishing template resources
822
Publishing template transformation parameters
823
Publishing template XSLT extensions
824
Publishing templates could not be loaded

852
Publishing templates troubleshooting
852
Right-to-left languages
883
Search features
785
Search layout page
782
Search rules
785
Sharing publishing templates
852, 971
Sharing templates
817
Topic layout page
779
Transformation parameters
906
Tweet button
886
Using publishing templates from a command line
851
XSLT-Import extension point
912
XSLT-import extension points
892
XSLT-Parameter extension point
912
WebHelp Responsive Templates tab
Save template as button
720
WSDL documents
500
Component Dependencies view
510
Content completion
501
Editing in Master Files context
501
Generate documentation for WSDL documents
514, 1293
Custom format
518
HTML format
517
Highlight occurrences
511
Moving resources
509
Outline view
503
Quick Assist feature
513
Refactoring actions
511
Renaming resources
509
Resource Hierarchy/Dependencies view
507
Search actions
511
SOAP analyzer
519, 1300
Test remote files
521, 1302
Syntax highlighting
502
Validation
501
WSDL SOAP Analyzer tool
519, 1300
Test remote files
521, 1302

X

XHTML document type
660
XInclude
321, 322
XInclude 1.1 features
322
XLIFF documents
Editing features

582
XML catalog preferences
121
XML Catalogs
312
XML documents
222
Associate schema
303
Associate schema directly in XML documents
310
Associate schema in a framework configuration
311
Associate schema in a validation scenario defined in framework
308
Associate schema through a validation scenario
305
Author Mode editing
Entities view
247
MathML equations in HTML output
905
Model view
245
Code templates
238
DTD Entities
321, 321
Editing in Master Files context
315
Grid Mode editing
268
Content Completion Assistant
275
Copy/Paste
273
Drag and Drop
272
Editing actions
270
Export to Excel
276
Learn document structure
312
Moving resources
320
Outline view
240
Quick Assist feature
257
Quick fix support
301
Refactoring
325, 1238
Built-in operations
329, 1241
Custom operations
337, 1249
Localizing operations
355, 1263
Sharing operations
354, 1262
Storing operations
354, 1262
Renaming resources
320
Resolve schemas through XML catalogs
315
Resource Hierarchy/Dependencies view
318
Search actions for IDs
316
Text Mode editing
222
Attributes view
243
Content completion
234
Contextual menu actions
258
Drag and Drop
232
Editing XML markup

227
Elements view
247
Entities view
247
Folding
231
Format and indent
251
Format and indent multiple files
256
Highlight ID occurrences
258
Model view
245
Navigation
223
Rectangular selection
232
Schema annotations
237
Selecting content
232
Set indent to zero
255
Shortcut actions
225
Smart Editing
224
Syntax highlighting
250
Validation
277
Against a schema
279
Against a schema with embedded Schematron
300, 595
Automatic validation
279
Check Well-formedness
277
Create new validation scenario
288
Custom validators
283
Customizing error messages
283
Edit validation scenario
293
Manual validation
279
Resolving references to remote schemas
298
Sharing scenarios
298
Text Mode
281
XInclude
321, 322
XML catalogs
312
XML documents without a schema
312
XML parser preferences
123
XML refactoring preferences
127
XML Refactoring tool
325, 1238
Built-in operations
329, 1241
Custom operations
337, 1249
Localizing operations
355, 1263
Sharing operations
354, 1262
Storing operations
354, 1262
XML Schema Design mode
169, 406
XML Schema Diagram Editor
169, 406

Attributes view
452
Components
419
Contextual menu actions
411
Edit namespaces
446
Editing actions
410
Facets view
409
Navigation
407
Outline view
450
Palette view
408
Validation
445
XML Schema Regular Expression Builder tool
480, 1302
XML Schemas
406
Attributes view
452
Component Dependencies view
456
Content completion
448
Convert DB Structure to XML Schema tool
478, 1272
Design mode editing
169, 406
Components
419
Grouping components
443
xs:all
436
xs:alternative
432
xs:any
437
xs:anyAttribute
438
xs:assert
442
xs:attribute
424
xs:attributeGroup
426
xs:choice
436
xs:complexType
427
xs:element
420
xs:field
441
xs:group
433
xs:import
434
xs:include
434
xs:key
440
xs:keyRef
440
xs:notation
435
xs:openContent
443
xs:override
435
xs:redefine
435
xs:schema
419
xs:selector
441
xs:sequence
436

xs:simpleType
430
xs:unique
439
Contextual menu actions
411
Edit namespaces
446
Editing actions
410
Facets view
409
Navigation
407
Palette view
408
Validation
445
Editing in Master Files context
446
Flatten Schema tool
478
Generate documentation for XML Schema
466, 1285
Customize PDF output for DITA
475
Customize PDF output for DocBook
474
Output formats
470
Generate Sample XML Files tool
461, 1264
Generate/Convert Schema tool
475, 1269
Highlight occurrences
458
Moving resources
456
Outline view
450
Quick Assist feature
460
Refactoring actions
458
Regular Expressions Builder
480, 1302
Renaming resources
456
Resource Hierarchy/Dependencies view
454
Search actions
458
Set version
484
Syntax highlighting
449
Text mode editing
446
Validation
447
XML Schema 1.1 support
482
XML to JSON tool
573, 1277
XML to PDF with CSS transformation scenario
714
CSS tab
714
Output tab
715
XML transformation with XQuery
707
XML transformation with XSLT
691
XPath Expressions
1119
Catalogs
1124
Prefix mapping
1124
XPath Builder view
1119
XPath Results view
1122

XProc Scripts
Editing features
588
XProc transformation scenario
749
Inputs tab
750
Options tab
752
Outputs tab
751
Parameters tab
751
XProc tab
750
XQuery
485
Content completion
486
Folding
488
Format and Indent
487
Generate HTML documentation
494, 1292
Input view
493
Outline view
488
Sequence view
496
Syntax highlighting
487
Transformations
495
Sequence view
496
Updating XML documents
499
XQJ
1189
Updating XML documents
499
Validation
485
XQuery Builder view
490
XQuery Debugger perspective
164, 1210
Breakpoints
1229
Debugging Java extensions
1235
Identify expressions
1228
Information views
1215
Breakpoints view
1216
Context view
1217
Messages view
1219
Nodes/Values Set view
1224
Output Mapping Stack view
1220
Stack view
1220
Templates view
1223
Trace view
1222
Variables view
1225
XPath Watch view
1218
Layout
1211
Performance profiling
1230
Profiling
Hotspots view
1233

Invocation Tree view
1232
Steps in a typical debugging process
1227
Toolbar
1212
XQuery Documentation tool
494, 1292
XQuery preferences
134
XQuery Profiler preferences
132
XQuery transformation scenario
754
FO Processor tab
712, 760
Output tab
712, 760
XQuery tab
707, 755
XSD to JSON Schema tool
568, 1280
XSL-FO processor
746
XSLT
Component Dependencies view
384
Component documentation
387
Content completion
369
Content completion in XPath expressions
370
Editing features
363
Editing in Master Files context
363
Generate documentation for XSLT stylesheets
398, 1289
Custom format
403
HTML format
401
Highlight occurrences
385
Input view
380
Moving resources
384
Outline view
376
Quick Assist feature
394
Quick fix support
367
Refactoring actions
389
Renaming resources
384
Resource Hierarchy/Dependencies view
382
Search declarations
386
Search references
386
Syntax highlighting
375
Text Value Templates
388
Unit test (XSpec)
396
Validating XSLT that call Java extensions
366
Validating XSLT with custom engines
366
Validation
364
XPath Tooltip Helper
374
XSLT Debugger perspective
162, 1210
Breakpoints
1229
Debugging Java extensions

1235
Debugging XSLT that call Java extensions
1235
Identify expressions
1228
Information views
1215
Breakpoints view
1216
Context view
1217
Messages view
1219
Nodes/Values Set view
1224
Output Mapping Stack view
1220
Stack view
1220
Templates view
1223
Trace view
1222
Variables view
1225
XPath Watch view
1218
Layout
1211
Performance profiling
1230
Profiling
Hotspots view
1233
Invocation Tree view
1232
Steps in a typical debugging process
1227
Supported processors
1236
Toolbar
1212
XSLT preferences
137
XSLT Profiler preferences
132
XSLT transformation scenario
732
FO Processor tab
700, 740
Output tab
700, 741
XSLT tab
692, 732
XSLTProc preferences
143

Copyright

Oxygen XML Developer Eclipse plugin User Manual

Syncro Soft SRL.

Copyright © 2002-2020 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 Websites referenced elsewhere within this documentation, and Syncro Soft SRL does not necessarily endorse the products, services, or information described or offered within them. Syncro Soft cannot guarantee that these links will work all the time and has 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 Eclipse plugin End-User License Agreement, as well as information regarding support for this product, while under warranty, is available through the Oxygen XML Developer Eclipse plugin End-User License Agreement.

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 Third-party License Agreements webpage.

Terms and conditions. For the terms and conditions for using Oxygen XML Developer Eclipse plugin, see Oxygen XML Developer Eclipse plugin End-User License Agreement.

Documentation. For the most current versions of documentation, see the Oxygen XML Developer Eclipse plugin User Manual.

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 support webpage.