

Oxygen XML WebHelp Respons

User Guide

Contents

Chapter 1. Getting Started.....	5
Installing.....	5
Activating.....	6
Upgrading.....	7
Generating Output.....	7
Running WebHelp Responsive from Oxygen XML Editor/Author.....	7
Running WebHelp Responsive from Command Line.....	8
Running WebHelp Responsive from an Integration Server.....	10
Running WebHelp Responsive from a Docker image.....	13
Increasing Memory Allocation for Java.....	14
Chapter 2. Layout and Features.....	16
Layout of the Responsive Page Types.....	16
Searching the Output.....	27
Built-in JavaScript-based Search Engine.....	28
Oxygen Feedback Search Engine.....	31
Context-Sensitive Help System.....	31
Accessibility.....	34
Writing Guidelines for Accessible Documentation.....	34
WebHelp Responsive VPAT Accessibility Conformance Report.....	40
Chapter 3. Adding Oxygen Feedback to WebHelp Responsive Documentation.....	63
Chapter 4. Developer Reference.....	66
Processing Stages.....	66
Publishing Templates.....	68
Publishing Templates Gallery.....	70
Publishing Template Package Contents.....	71
Transformation Parameters.....	104
XSLT-Import and XSLT-Parameter Extension Points.....	118
Chapter 5. Customizing WebHelp Responsive Output.....	122
Working with Publishing Templates.....	122
How to Create a Publishing Template.....	122
How to Edit a Packed Publishing Template.....	125

How to Add a Publishing Template to the Publishing Templates Gallery.....	125
How to Use a Publishing Template from a Command Line.....	126
How to Share a Publishing Template.....	127
Troubleshooting: Errors Encountered when Loading Templates.....	128
Converting Old Templates to Newer Versions.....	128
Changing the Layout and Styles.....	132
How to Use CSS Styling to Customize the Output.....	132
How to Insert Custom HTML Content.....	134
How to Change Numbering Styles for Ordered Lists.....	140
How to Add Syntax Highlights for Codeblocks in the Output.....	141
How to Show or Hide Navigation Links in Topic Pages.....	143
How to Change the Main Page Layout.....	143
Adding Graphics and Media Resources.....	149
How to Add a Logo Image in the Title Area.....	149
How to Add a Favicon.....	150
How to Add Video and Audio Objects.....	151
How to Add MathML Equations in WebHelp Output.....	153
Searching the Output.....	153
Built-in JS Based Search Engine Customizations.....	153
Oxygen Feedback Search Engine.....	158
Custom Search Engine.....	162
How to Display Custom Title in Search Results.....	169
How to Trigger a Search Query When WebHelp is Loaded.....	169
Configuring the Search Engine Optimization.....	170
Localization.....	171
How to Localize the Interface.....	171
How to Activate Support for Right-to-Left (RTL) Languages.....	174
Social Media and Google Tools.....	174
How to Add a Facebook Like Button.....	174
How to Add a Tweet Button.....	176
How to Integrate Google Analytics.....	179
Ant Extensions for WebHelp Responsive.....	180
XSLT Extensions for WebHelp Responsive.....	183

How to Use XSLT Extension Points from a Publishing Template.....	183
How to Use XSLT Extension Points from a DITA-OT Plugin.....	188
Miscellaneous Customization Topics.....	192
How to Copy Additional Resources to Output Directory.....	192
How to Add an Edit Link to Launch Oxygen XML Web Author.....	193
How to Flag DITA Content.....	195
How to View MathML Equations in HTML Output.....	196
How to Disable Caching in WebHelp Responsive Output.....	197
How to Add a Link to PDF Documentation.....	198
How to Add a Custom Component for WebHelp Output.....	198
How to Generate Google Structured Data.....	203
How to Group Related Links by Type.....	207
How to Use a Local Font in WebHelp Responsive Output.....	207
How to Use JQuery in WebHelp Responsive Output.....	210
Chapter 6. Glossary.....	212
Anchor.....	212
Block Element.....	212
Bookmap.....	212
DITA Map.....	212
DITA Open Toolkit.....	212
DITA-OT-DIR.....	213
Framework.....	213
Inline Element.....	213
Key Space.....	213
Root Map.....	213
WebHelp Output Directory.....	214
Index.....	a
Copyright.....	d

1.

Getting Started

WebHelp is a form of online help that consists of a series of web pages (XHTML format). Its advantages include platform independence, the ability to update content continuously, it can be viewed using a regular web browser, and a comments component can be embedded in the output to provide an efficient way to interact with users.

The **WebHelp Responsive** variant features a very flexible layout and is designed to adapt to any device and 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.

Oxygen XML WebHelp Responsive plugin is a standalone product that requires its own license key. It provides support for transforming DITA resources into WebHelp output by running a transformation script outside of **Oxygen XML Editor/Author**. This is especially useful if you want to automate the output process.

Browser Compatibility

This output format is compatible with the most recent versions of the following common browsers:

- Edge
- Chrome
- Firefox
- Safari
- Opera

Installing WebHelp Responsive

The requirements of the Oxygen XML WebHelp Responsive plugin for the *DITA Open Toolkit* are as follows:

- Java 17 or later.
- *DITA-OT* 4.1.2 (includes Saxon 10.x libraries).



Note:

The Oxygen XML WebHelp Responsive plugin has only been tested with this specific version: **(DITA-OT 4.1.2)**. It is possible for the plugin to work with other versions, but only this version is fully supported.

To install and integrate the Oxygen XML WebHelp Responsive plugin with the *DITA-OT*, follow these steps:

1. Download and install [Java 17 or later](#).
2. Download and unpack the [DITA Open Toolkit](#) version 4.1.2.

3. Go to [Oxygen XML WebHelp website](#), download the latest *DITA-OT* version of the Oxygen XML WebHelp Responsive plugin installation kit, and unzip it.
4. Copy all *plugin* directories from the unpacked archive to the `plugins` directory of the *DITA-OT* distribution. This is necessary to enable certain functionality. For example, the `com.oxygenxml.highlight` directory adds syntax highlight capabilities to your WebHelp output for `<codeblock>` elements that contain source code snippets (XML, Java, JavaScript).
5. [Register your license key \(on page 6\)](#).
6. In the `DITA-OT-DIR/bin` directory of the *DITA-OT*, run one of the following scripts, depending on your operating system:
 - Windows: `DITA-OT-DIR/bin/dita.bat --install`
 - Linux/macOS: `sh DITA-OT-DIR/bin/dita --install`
7. To test the installation, initiate a publishing process by using one of the options found here: [Generating WebHelp Responsive Output \(on page 7\)](#).

Related Information:

[Activating WebHelp Responsive License \(on page 6\)](#)

[Upgrading WebHelp Responsive \(on page 7\)](#)

[DITA-OT Installing Plug-ins](#)

[Generating WebHelp Responsive Output \(on page 7\)](#)

Activating WebHelp Responsive License

To register the license for the Oxygen XML WebHelp Responsive plugin for the DITA Open Toolkit, follow these steps:

1. Obtain a valid license for the **Oxygen XML WebHelp Plugin** at https://www.oxygenxml.com/buy_webhelp.html.
2. Create a file called `licensekey.txt`, copy your license key that you purchased for your Oxygen XML WebHelp Responsive plugin, and place this file in the `DITA-OT-DIR/` root directory or in the WebHelp plugin's dedicated folder (e.g. `DITA-OT-DIR/plugins/com.oxygenxml.webhelp.responsive/`). When you execute a [WebHelp Responsive transformation \(on page 7\)](#), the process reads the Oxygen XML WebHelp Responsive plugin license key from this file. If the file does not exist, or it contains an invalid license, an error message will be displayed.

Related Information:

[Installing WebHelp Responsive \(on page 5\)](#)

[Upgrading WebHelp Responsive \(on page 7\)](#)

Upgrading WebHelp Responsive



Important:

The first step of the procedure below instructs you to delete old directories/files before proceeding with the upgrade. It is recommended that you make a backup of these directories/files before deleting them. Make sure you make a backup of the `licensekey.txt` file since you will need the information contained in this file later in the upgrading procedure.

To upgrade your version of the Oxygen XML WebHelp Responsive plugin for the *DITA-OT*, follow these steps:

1. Navigate to the `plugins` directory of your *DITA-OT* distribution and delete the following old Oxygen XML WebHelp Responsive plugin directories (`com.oxygenxml.highlight`, `com.oxygenxml.html.custom`, `com.oxygenxml.media`, `com.oxygenxml.webhelp.classic`, `com.oxygenxml.webhelp.common`, `com.oxygenxml.webhelp.responsive`).
2. Go to [Oxygen XML WebHelp website](#), download the latest *DITA-OT* version of the Oxygen XML WebHelp Responsive plugin installation kit, and unzip it.
3. Copy all *plugin* directories from the unpacked archive to the `plugins` directory of the *DITA-OT* distribution. This is necessary to enable certain functionality. For example, the `com.oxygenxml.highlight` directory adds syntax highlight capabilities to your WebHelp output for `<codeblock>` elements that contain source code snippets (XML, Java, JavaScript).
4. Open the `DITA-OT-DIR/plugins/com.oxygenxml.webhelp.responsive` directory, create a file called `licensekey.txt`, and copy your license key that you purchased for your Oxygen XML WebHelp Responsive plugin.

When you execute a [WebHelp Responsive transformation \(on page 7\)](#), the process reads the Oxygen XML WebHelp Responsive plugin license key from this file. If the file does not exist, or it contains an invalid license, an error message will be displayed.

5. In the `DITA-OT-DIR/bin` directory of the *DITA-OT*, run one of the following scripts, depending on your operating system:

- Windows: `DITA-OT-DIR/bin/dita.bat --install`
- Linux/macOS: `sh DITA-OT-DIR/bin/dita --install`

Related Information:

[Installing WebHelp Responsive \(on page 5\)](#)


[Activating WebHelp Responsive License \(on page 6\)](#)

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 WebHelp Responsive from Oxygen XML Editor/Author

To publish a *DITA map* ([on page 212](#)) as **WebHelp Responsive** output, follow these steps:

1. Select the  **Configure Transformation Scenario(s)** action from the **DITA Maps Manager** toolbar.
2. Select the **DITA Map WebHelp Responsive** scenario from the **DITA Map** section.
3. If you want to configure the transformation, click the **Edit** button.

Step Result: This opens an **Edit scenario** configuration dialog box that allows you to configure various options in the following tabs:

- **Templates Tab** - This tab contains a set of built-in skins that you can use for the layout of your WebHelp system output.
- **Parameters Tab** - This tab includes [numerous transformation parameters \(on page 104\)](#) 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.

Running WebHelp Responsive from Command Line

To publish to the WebHelp Responsive output from a command line outside of **Oxygen XML Editor/Author**, you can use the `dita` startup script that comes bundled with *DITA-OT* distribution.

`dita` Command Format

DITA-OT `dita` command has the following format:

```
DITA-OT-DIR/bin/dita --format=webhelp-responsive --input=input-file options
```

where the arguments are as follows:

dita

Windows - The `dita.bat` script located in: `DITA-OT-DIR (on page 213)\bin\`.

Linux/macOS - The `dita` script file located in: `DITA-OT-DIR (on page 213)/bin/`.

--format=webhelp-responsive

Specifies the output format (transformation type) for WebHelp Responsive transformation.

--input=input-file

The `input-file` represents the path to the DITA map that you want to process.

options

`options` include the following optional build parameters:

--output=dir

-o dir

Specifies the path of the output directory; the path can be absolute or relative to the current directory. By default, the output is written to the `out` subdirectory of the current directory.

--filter=file

Specifies filter file(s) used to include, exclude, or flag content.

Relative paths are resolved against the current directory and internally converted to absolute paths.

--temp=dir

-t dir

Specifies the location of the temporary directory.

--verbose

-v

Verbose logging.

--debug

-d

Debug logging.

--logfile=file

-l file

Write logging messages to a file.

--parameter=value

-Dparameter=value

Specify a value for a DITA-OT or Ant build parameter.

--propertyfile=file

Use build parameters defined in the referenced `.properties` file.

Build parameters specified on the command line override those set in the `.properties` file.

WebHelp and DITA-OT parameters

In addition to the [transformation parameters](#) that are specific to WebHelp Responsive (*on page 104*), you can use the [common DITA-OT transformation parameters](#) and the [HTML-based Output Parameters](#).

Command-Line Example

- Windows:

```
dita.bat
--format=webhelp-responsive
--input=c:\path\to\mySample.ditamap
--output=c:\path\to\output
-Dwebhelp.logo.image=myLogo.jpg
```

- Linux/macOS:

```
dita
--format=webhelp-responsive
--input=/path/to/mySample.ditamap
--output=/path/to/output
-Dwebhelp.logo.image=myLogo.jpg
```



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:

[DITA-OT Documentation: Building Output Using the dita Command](#)

[DITA-OT Documentation: Publishing DITA Content](#)

[DITA-OT Documentation: HTML-based output parameters](#)

[WebHelp Responsive Transformation Parameters \(on page 104\)](#)

Running WebHelp Responsive from an Integration Server

WebHelp output can be processed from an automatic publishing system, such as *Jenkins* or *Travis*.

Automating DITA to WebHelp Responsive Output with Jenkins

This procedure assumes that you have already [integrated, configured, and registered the Oxygen XML WebHelp Responsive plugin with the DITA Open Toolkit \(on page 5\)](#).

To integrate WebHelp output with the Jenkins continuous integration tool, follow these steps:

1. Create a Maven project to incorporate the DITA-OT that already integrates Oxygen XML WebHelp Responsive plugin.
2. Go to the root of your Maven project and edit the `pom.xml` file to include the following fragment:

```

<properties>
  <dita-ot-dir>${basedir}/tools/dita-ot</dita-ot-dir>
  <!--
    The path to the DITA map that you want to process.
  -->
  <input-file>path/to/input_file</input-file>

  <!--
    Specifies the path of the output directory.
  -->
  <output-dir>path/to/output_dir</output-dir>

  <!--
    The path to the WebHelp publishing template.
  -->
  <publishing-template>path/to/publishing_template</publishing-template>

  <!--
    DITA-OT optional build parameters.
  -->
  <options>-Dwebhelp.publishing.template=${publishing-template} -v</options>
</properties>

<plugin>
  <artifactId>exec-maven-plugin</artifactId>
  <groupId>org.codehaus.mojo</groupId>
  <executions>
    <execution>
      <!-- Run WebHelp Responsive transformation -->
      <id>run-webhelp-responsive</id>
      <phase>generate-sources</phase>
      <goals>
        <goal>exec</goal>
      </goals>
      <configuration>
        <executable>${dita-ot-dir}/bin/dita.bat --format=webhelp-responsive
          --input=${input-file} --output=${output-dir} ${options}</executable>
      </configuration>
    </execution>
  </executions>
</plugin>

```

```
</executions>
</plugin>
```

3. Go to the Jenkins top page and create a new Jenkins job. Configure this job to suit your particular requirements, such as the build frequency and location of the Maven project.

Automating DITA to WebHelp Responsive Output with Travis CI

This topic assumes you have a DITA project hosted on a GitHub public or private repository.

The goal of this tutorial is to help you set up a Travis continuous integration job that automatically publishes your DITA project to [GitHub pages](#) after every commit. The published website will contain a feedback link on each page that would allow a contributor to easily suggest changes to the documentation by creating a pull request on GitHub with just a few clicks.

Enable the Travis CI Build

1. [Sign in to Travis CI](#) with your GitHub account, accepting the GitHub [access permissions confirmation](#).
2. Once you are signed in, and you have synchronized your GitHub repositories, go to your [profile page](#) and enable Travis CI for the repository you want to build.

Configure the Travis CI Build in your GitHub Project

1. Checkout your GitHub project locally.
2. Copy the `.travis` folder from [here](#) to the root directory of your project.
3. In the root of your GitHub project, add a file called `.travis.yml` with the following content:

```
language: dita
install:
  - echo "Installed"
script:
  - sh .travis/publish.sh
after_success:
  - sh .travis/deploy.sh
env:
  global:
    - DITAMAP=/path/to/your/ditamap/file
    - DITAVAl=/path/to/your/ditaval/file
    - ANT_OPTS=-Xmx1024M
```



Note:

Replace `/path/to/your/ditamap/file` and `/path/to/your/ditaval/file` with the appropriate paths to your *DITA map* and *ditaval* files.

4. Create a GitHub personal access token by following [this procedure](#).
5. Define an environment variable in the repository settings that has the name `GH_TOKEN` and the value equal with the GitHub personal access token created earlier.

Register Your License Key

1. Edit your `.gitignore` file (or create it if it does not already exist) and add the following line:

```
licenseKey.txt
```

2. Copy your WebHelp license to the root of your GitHub project in a file called `licenseKey.txt`.



Important:

The `licenseKey.txt` file should not be committed to GitHub as it contains a license key that is issued only to you.

3. Encrypt the license key file and add it to the `.travis.yml` configuration file. This way only the Travis CI server will be able to decrypt it during the build process.

Commit to GitHub

1. Commit the following files and folders and push the commit to GitHub:

```
git add .gitignore licenseKey.txt.enc .travis.yml .travis/
git commit -m "Set up the Travis CI publishing system"
git push
```

2. Create a `gh-pages` branch in your GitHub project where the WebHelp Responsive output will be published. You can follow the procedure [here](#).

Running WebHelp Responsive from a Docker image

This topic explains how to install the **WebHelp Responsive** plugin in a Docker image.

To install the **Oxygen XML WebHelp Responsive** plugin in a Docker image, follow these steps:

1. Download and install Docker.
2. Create a folder (for example, `webhelp-docker`).
3. Move the `licensekey.txt` file for the **WebHelp Responsive** plugin to the newly created folder.
4. Create a new file named `Dockerfile` with the following content and store it in the newly created folder:

```
# Use the latest DITA-OT image as parent
FROM ghcr.io/dita-ot/dita-ot:3.6.1

# Build argument form the WebHelp download link
ARG WEBHELP_DOWNLOAD_LINK
```

```

# Download the WebHelp zip kit.
RUN curl -o /tmp/oxygen-webhelp.zip ${WEBHELP_DOWNLOAD_LINK}

# Unzip the WebHelp kit to the plugins directory of the DITA-OT distribution.
RUN unzip /tmp/oxygen-webhelp.zip -d /opt/app/plugins

# Remove the WebHelp zip.
RUN rm /tmp/oxygen-webhelp.zip

# Copy the license key.
COPY licensekey.txt /opt/app/

# Install the WebHelp plugins.
RUN dita --install

```

5. Build an image from the **Dockerfile** by running the following command:

```

docker image build --build-arg
  WEBHELP_DOWNLOAD_LINK=https://www.oxygenxml.com/InstData/WebHelp/oxygen-webhelp-dot-3.x.zip
  -t webhelp-docker:23.1 ${PATH_TO_DOCKERFILE}

```

6. Run a WebHelp Responsive transformation from docker:

```

docker run -it \
-v ${PATH_TO_DITAMAP}:/src webhelp-docker:23.1 \
-i /src/map.ditamap \
-o /src/out \
-f webhelp-responsive -v

```



Attention:

Make sure that you do not violate the license model. More information can be found in the [Oxygen XML WebHelp Responsive plugin End-User License Agreement](#).

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.

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 HTML document).
- From the Chemistry PDF CSS processor (the transformation of the merged HTML document to PDF).

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. Uncheck the **Prefer using the "dita" command** option
5. Locate the **JVM Arguments** and increase the default value. For instance, to set 2 gigabytes as the maximum amount of memory, you can use: `-Xmx2g`. If you do not specify the **-Xmx** value in this field, by default, the application will use a maximum of 512 megabytes when used with a 32-bit Java Virtual Machine and one gigabyte with a 64-bit Java Virtual Machine.

**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

- **If the DITA-OT process fails with Out Of Memory Error:** 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/macOS**

```
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=-Xmx2048m
```

2.

Layout and Features

The **WebHelp Responsive** features a very flexible layout and 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.

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 1. WebHelp Responsive Output on a Normal Screen

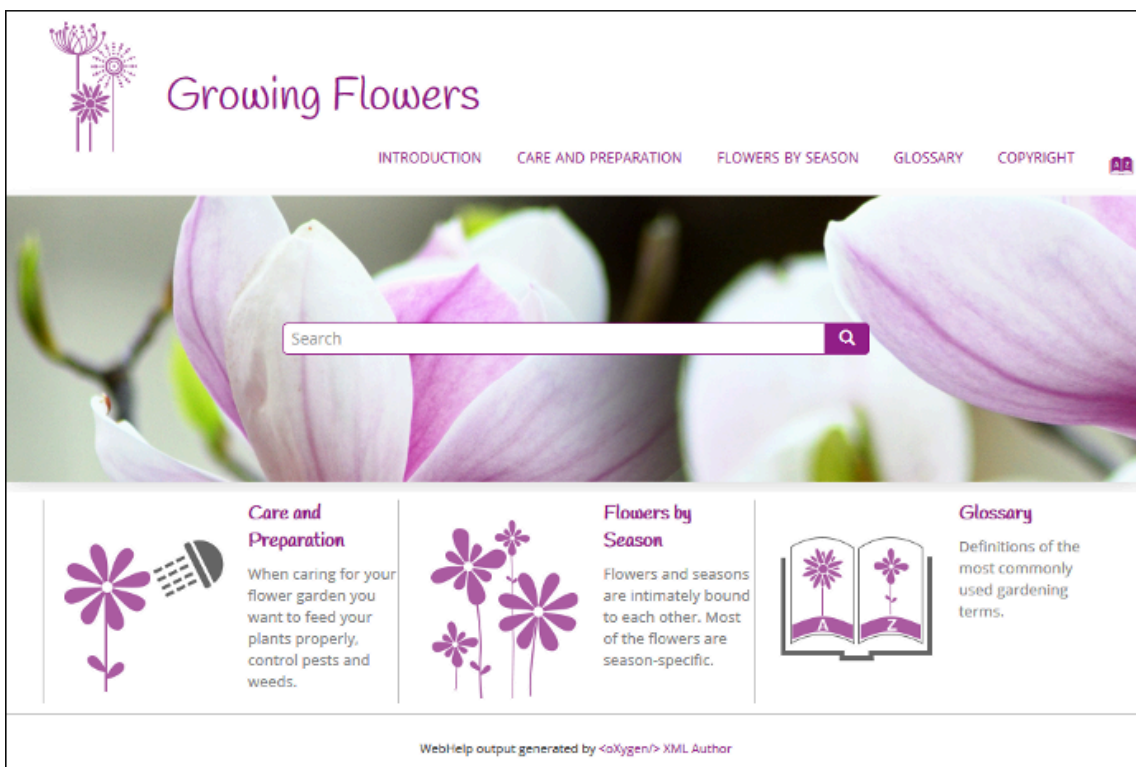
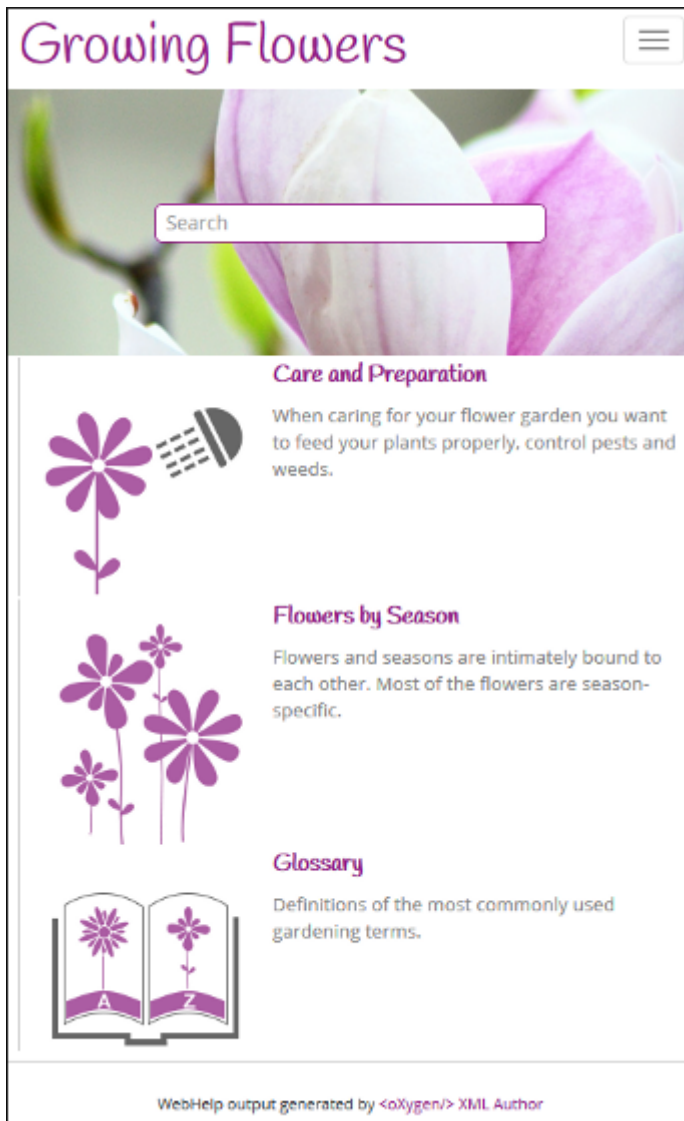


Figure 2. WebHelp Responsive Output on a Narrow Screen



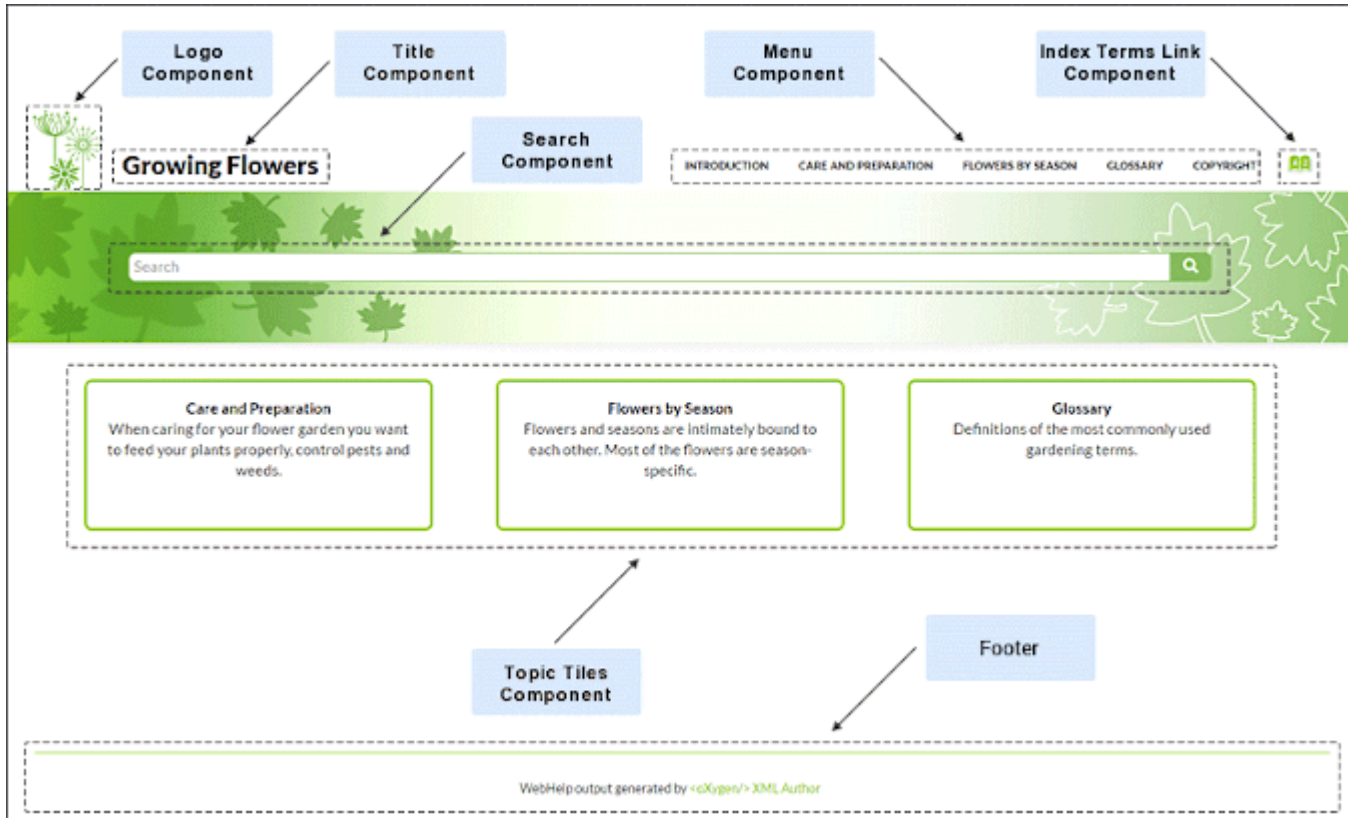
Main Page

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 3. Main Page - Tiles Layout

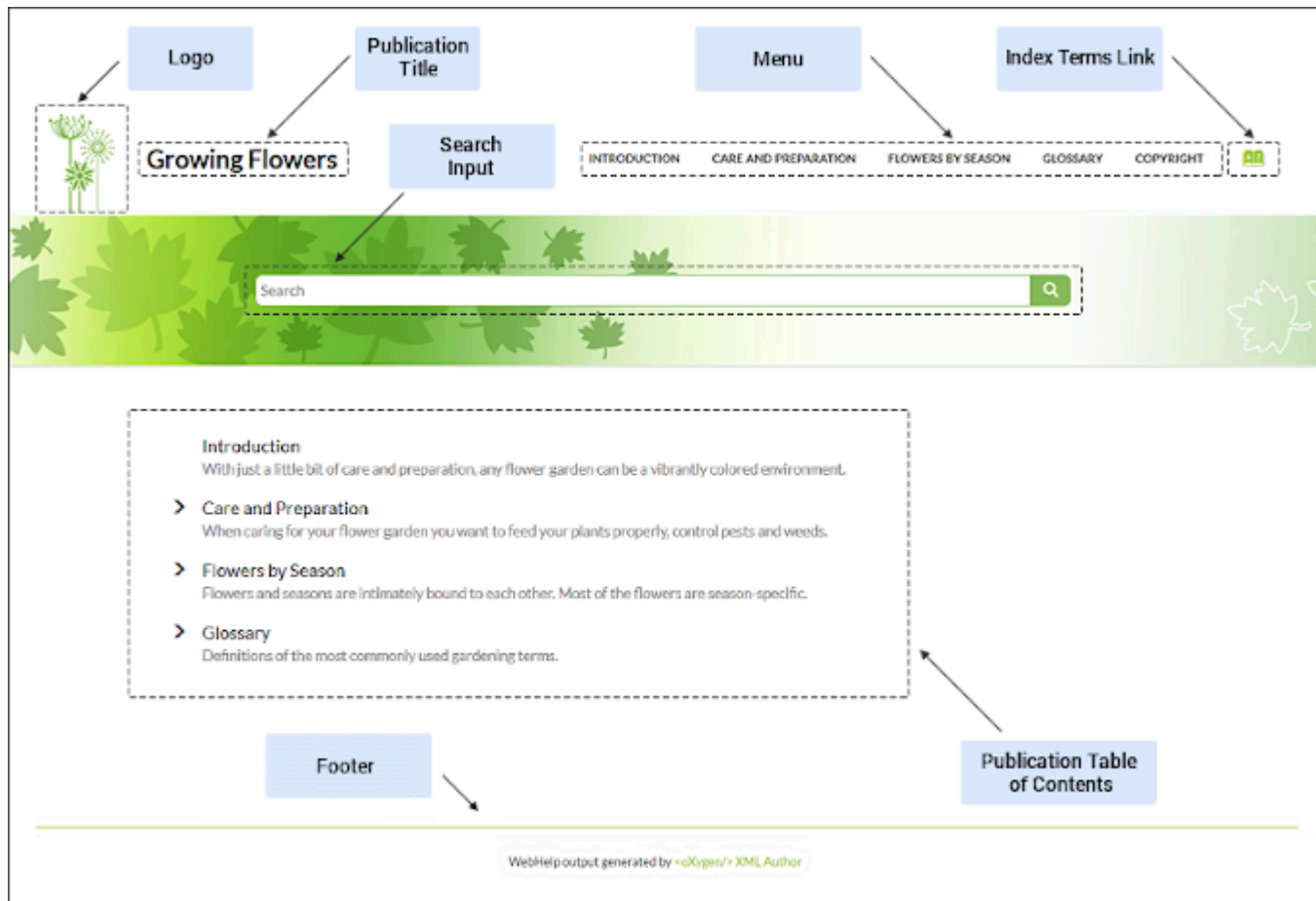


1. Logo Component (on page 19)
2. Title Component (on page 19)
3. Search Input Component (on page 20)
4. Menu Component (on page 20)
5. Index Terms Link Component (on page 20)
6. Topic Tiles Component (on page 20)
7. Footer Component (on page 20)

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 4. Main Page - Tree Layout



1. Logo Component (on page 19)
2. Title Component (on page 19)
3. Search Input Component (on page 20)
4. Menu Component (on page 20)
5. Index Terms Link Component (on page 20)
6. Table of Contents Component (on page 20)
7. Footer Component (on page 20)

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 105](#)). For the target URL, use the [webhelp.logo.image.target.url](#) parameter ([on page 105](#)).

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 143](#)) 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 114](#)).

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 143](#)) is done in the DITA map.

Tree Table of Contents

An area that contains first and second-level topic titles from your publication.

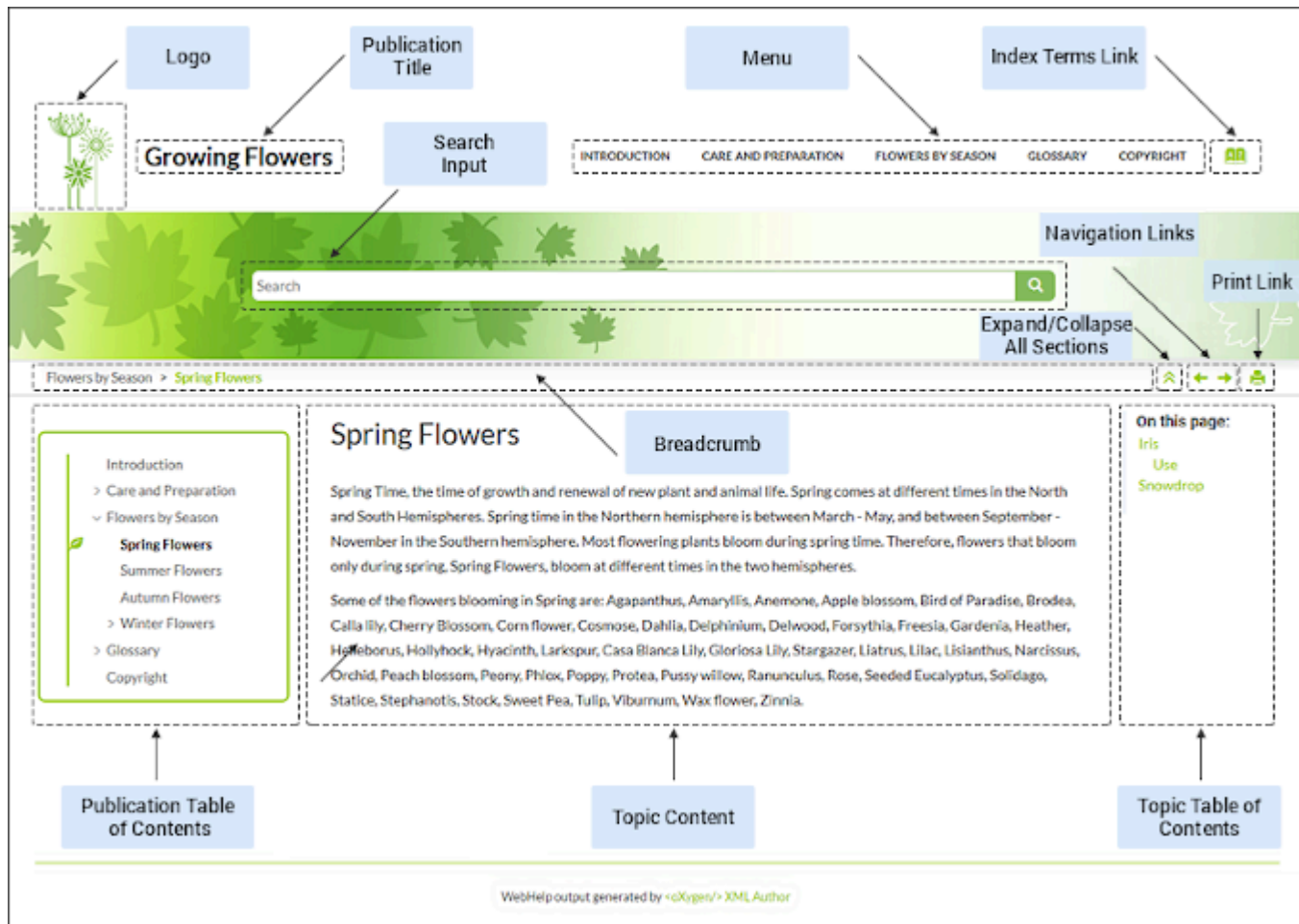
Page Footer

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 5. Topic Page



1. Logo Component (on page 22)
2. Title Component (on page 21)
3. Search Input Component (on page 22)
4. Menu Component (on page 22)
5. Index Terms Link Component (on page 22)
6. Expand/Collapse All Sections Component (on page 22)
7. Navigation Links Component (on page 22)
8. Print Link Component (on page 22)
9. Breadcrumb Component (on page 22)
10. Publication Table of Contents Component (on page 23)
11. Topic Content Component (on page 22)
12. Topic Table of Contents Component (on page 23)

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 105). For the target URL, use the `webhelp.logo.image.target.url` parameter (on page 105).

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 \(on page 143\)](#) 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 114).

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 of the screen. You can use the  button to collapse the table of contents (or the  button to expand it).

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. You can use the  button to collapse the table of contents (or the  button to expand it).

Page Footer

WebHelp Responsive output footer.

Search Results 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.


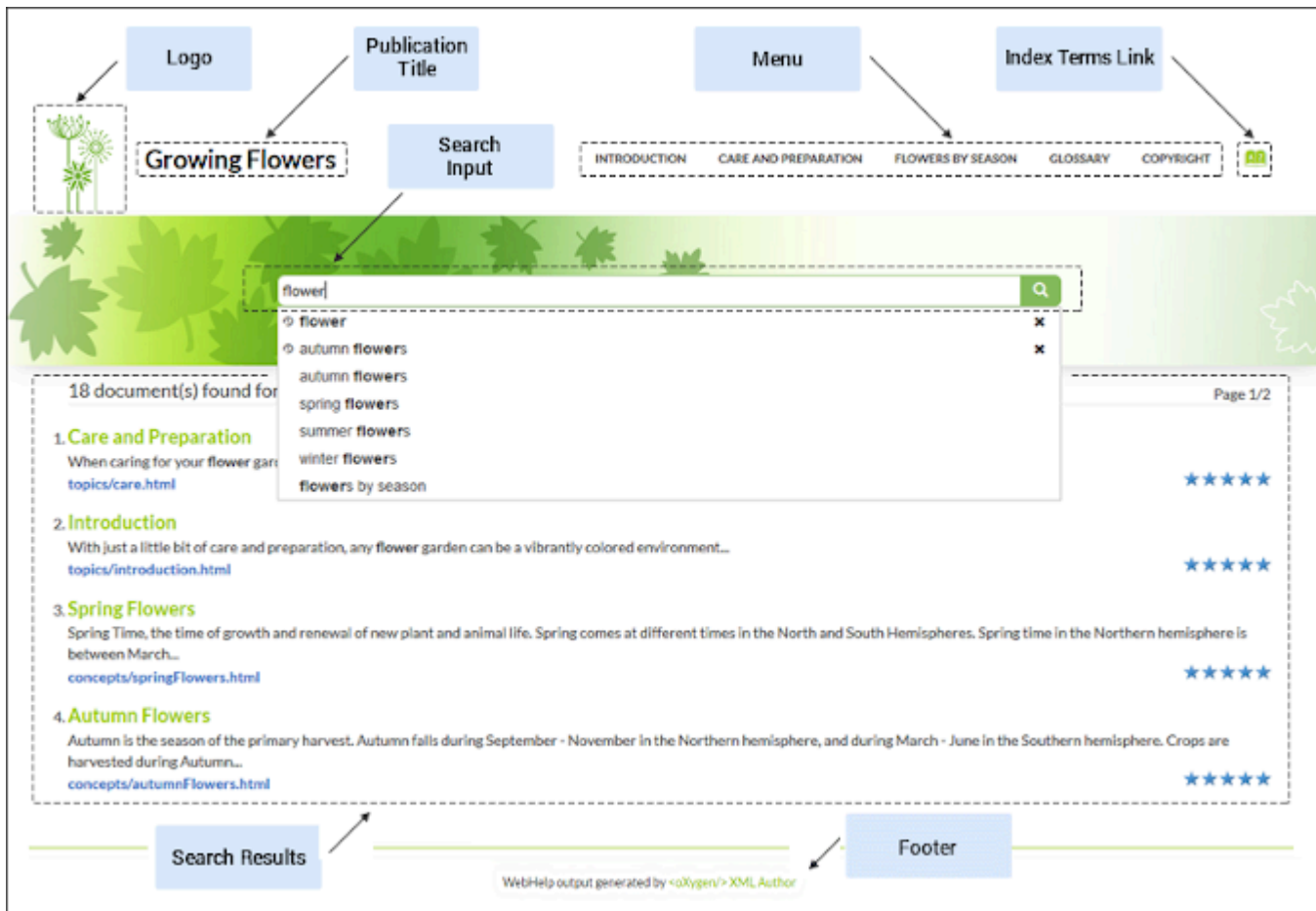
When you enter search terms in the **Search** field, the results are displayed in a results page. When you click on a result, the topic is opened in the main pane and the search results are highlighted. If you want to remove the colored highlights, click the  **Toggle Highlights** button at the top-right side of the page. The **Search** field also includes an *autocomplete* feature.

Figure 6. Search Results Page



1. Logo Component (on page 24)
2. Title Component (on page 24)
3. Search Input Component (on page 25)
4. Menu Component (on page 25)
5. Index Terms Link Component (on page 25)
6. Search Results Component (on page 25)
7. Footer Component (on page 25)

Search Results 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 105*). For the target URL, use the `webhelp.logo.image.target.url` parameter (*on page 105*).

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 143*) 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 114*).

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.

Auto-complete Suggestions in the Search Input Field

When you are typing in the search input 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

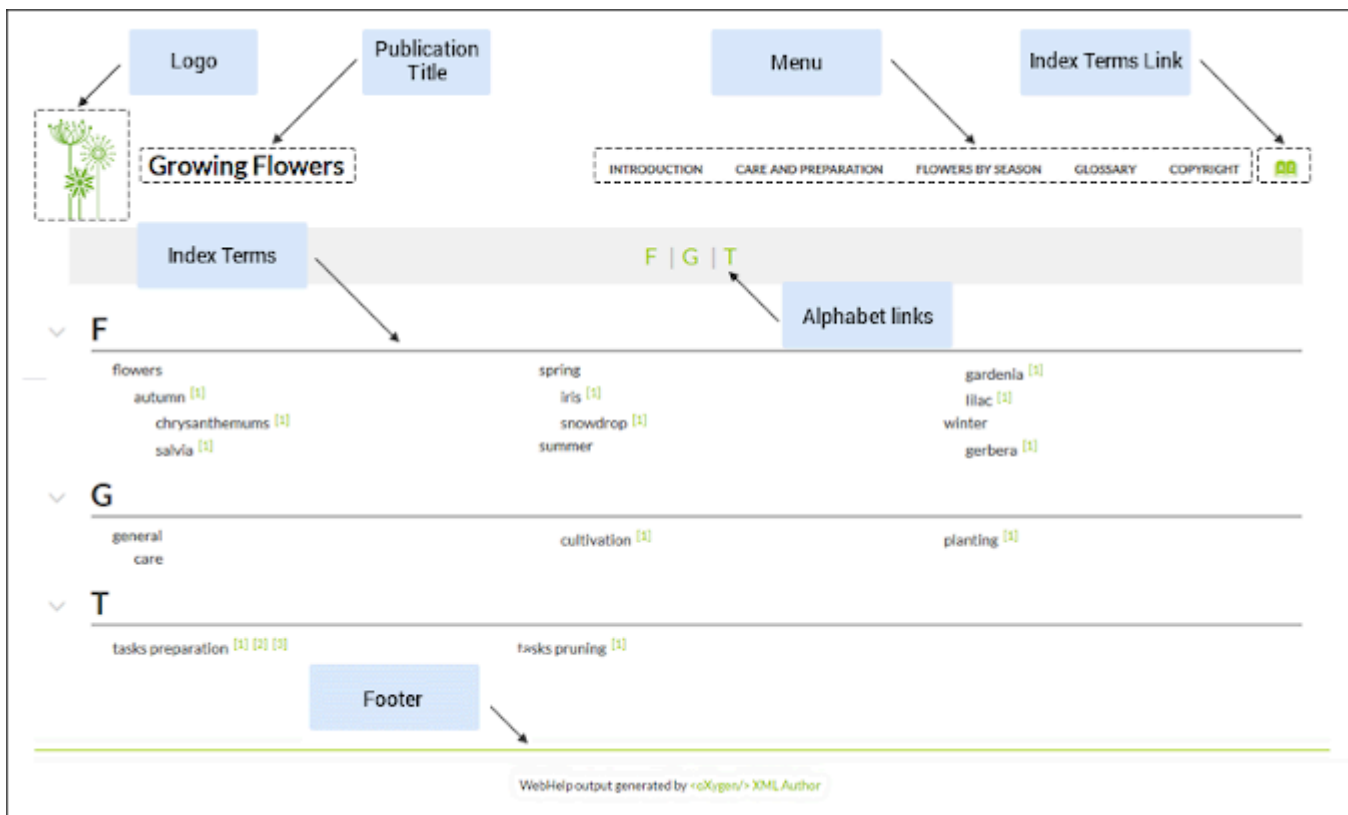
[WebHelp Responsive Search Engine \(on page 28\)](#)

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 7. Index Terms Page



1. Logo Component (on page 27)
2. Title Component (on page 27)
3. Menu Component (on page 27)
4. Index Terms Link Component (on page 27)
5. Index Terms Component (on page 27)

6. [Alphabet Links Component \(on page 27\)](#)

7. [Footer Component \(on page 27\)](#)

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 \(on page 105\)](#). For the target URL, use the [webhelp.logo.image.target.url parameter \(on page 105\)](#).

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 \(on page 143\)](#) 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 \(on page 114\)](#).

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.

Searching the Output

WebHelp Responsive for DITA output can be configured to work with multiple search engines.

The plugin comes with a [built-in JavaScript-based search engine \(on page 28\)](#) configured by default. This search engine runs client-side and is recommended for a small or medium-sized documentation, offering a fairly good set of features such as sorting results by relevance, 5-star rating mechanism, and custom rules for element scoring.

In addition to the built-in search engine, the WebHelp Responsive for DITA output can be easily configured to use the [Oxygen Feedback as an external search engine \(on page 31\)](#). This is a more advanced search engine that runs server-side, suitable for large documentations, and with a wider range of features such as semantic search, faceted search, and labels search.

A general comparison between the built-in search engine and the search engine from Oxygen Feedback:

Built-in JS based search engine

As a client-side search process, it has the following characteristics:

- Runs in the user's web browser.
- Can be slower and less efficient than server-side searching, especially for large amounts of content.
- Requires downloading the entire search index to the user's device, which can be a burden for slower connections or limited devices.
- Can be limited by the resources available on the user's device, such as processing power and memory.

Oxygen Feedback Search Engine

As a server-side process, the **Oxygen Feedback** search engine:

- Runs on a separate server.
- It is faster and more efficient than client-side searching, especially for large amounts of content.
- Returns only the relevant search results to the user's web browser, reducing the amount of data that needs to be downloaded.
- Is not limited by the resources available on the user's device, as the server can scale to handle large amounts of content.

If none of the above options represent a choice, then the WebHelp Responsive transformation can be configured to work with other external search engines such as the [Google search engine \(on page 162\)](#).

Related information

[How to Integrate Google Search in WebHelp Responsive Output \(on page 162\)](#)

Built-in JavaScript-based Search Engine

The client-side JavaScript-based search engine comes preconfigured in the WebHelp responsive plugin. It is enabled by default when you run the WebHelp Responsive transformation.


Search Index

The search index is created when you publish your documentation to WebHelp by traverses all HTML pages (for DITA topics) from output folder 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.



Tip:

You can use the `searchQuery` URL parameter to perform a search operation when WebHelp is loaded. This opens the Search Results page with the specified search query processed. The URL should look something like this:

```
http://localhost/webhelp/search.html?searchQuery=deploying%20feedback
```

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 \(on page 153\)](#).

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 multiple words in CJK (Chinese, Japanese, Korean) languages that often have them appear in strings without a space separator, you may 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.



Note:

Phrase searches (two or more consecutive words in an exact order) do not work for CJK (Chinese, Japanese, Korean) languages.



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 \(on page 23\)](#)

Oxygen Feedback Search Engine

Starting with version 3.0, Oxygen Feedback can be configured as an **ready-to-use external search engine** for the **Oxygen WebHelp Responsive** output. You can read more about benefits and activation in the [Oxygen Feedback](#) documentation.

Related information

[Adding Oxygen Feedback to WebHelp Responsive Documentation \(on page 63\)](#)

[How to Configure Faceted Search in WebHelp Output \(on page 158\)](#)

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 to 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 `appidname` value (and multiple `@appidname` 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 `@appidname` attributes should be unique within the context of a single [root map \(on page 213\)](#). For example, suppose that you need two different functions of an application to both open the same WebHelp page.

Example: The `<resourceid>` Element 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: The `<resourceid>` Element 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 `cshelp.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.

```
cshelp.html?contextId=myDITATopic
```

**Note:**

The `contextId` parameter is not case-sensitive.

**Attention:**

Prior to version 24.1, the method was to invoke the `index.html` file. The system still works using this method but it has been deprecated and its functionality will be removed in a future version.

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 212)* 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/cshelp.html?contextId=topicID&appname=myApplication
```

**Tip:**

The `webhelp.csh.disable.topicID.fallback` parameter (on page 105) can be set `true` to use a topic ID fallback when `resourceid` information is not available when computing the mapping for context sensitive help.

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/suction_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 (CAL S 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 1. 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>
    </tbody>
  </tgroup>
</table>
```

```

</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 2. 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 3. 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 \(on page 34\)](#).

Evaluation Methods Used:

The following applications were used for testing **Oxygen XML WebHelp Responsive**:

- Desktop browsers: Chrome, Firefox, Safari, Edge.
- 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

Standard/Guideline	Included In Report
	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 the [WCAG 2.0 Conformance Requirements](#).

Table 1: Success Criteria, Level A

Criteria	Conformance Level	Remarks and Explanations
<p>1.1.1 Non-text Content (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p>	Partially Supports	Text alternatives are provided for many instances of non-text content, with exceptions that include perma-links for subtopics and sections.

Criteria	Conformance Level	Remarks and Explanations
<ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 		
<p><u>1.2.1 Audio-only and Video-only (Prerecorded)</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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.
<p><u>1.2.2 Captions (Prerecorded)</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	The product does not provide prerecorded media that requires captions.
<p><u>1.2.3 Audio Description or Media Alternative (Prerecorded)</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	<p>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.</p> <p>See: G58: Placing a link to the alternative for time-based media immediately next to the non-text content</p>
<p><u>1.3.1 Info and Relationships</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p>	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
<ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 		<ul style="list-style-type: none"> • 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. <p>The authors of the input DITA document are responsible for:</p> <ul style="list-style-type: none"> • 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.
<p><u>1.3.2 Meaningful Sequence</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	<p>The product presents content in a meaningful sequence.</p> <p>Authors should use Unicode right-to-left mark (RLM) or left-to-right mark (LRM) to mix text direction inline.</p>
<p><u>1.3.3 Sensory Characteristics</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	<p>Authors should ensure that items are referenced in the content in ways that do not depend on sensory perception.</p>
<p><u>1.4.1 Use of Color</u> (Level A)</p>	Supports	<p>(Cobalt template) Color is not used as the only visual means of convey-</p>

Criteria	Conformance Level	Remarks and Explanations
<p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 		<p>ing information, indicating an action, prompting a response, or distinguishing a visual element.</p>
<p><u>1.4.2 Audio Control</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	<p>There is no sound that plays automatically.</p>
<p><u>2.1.1 Keyboard</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Partially Supports	<p>Most of the content is operable through a keyboard interface, with exceptions that include:</p> <ul style="list-style-type: none"> • The submenus (the user cannot tab to the submenus). • The top-level links in the main page accordion cannot be accessed. • The facets component does not have full keyboard support.
<p><u>2.1.2 No Keyboard Trap</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	<p>The product does not contain content that traps the keyboard focus.</p>

Criteria	Conformance Level	Remarks and Explanations
<p><u>2.1.4 Character Key Shortcuts</u> (Level A 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	Supports	The product does not include character key shortcuts.
<p><u>2.2.1 Timing Adjustable</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	The product does not include time limits.
<p><u>2.2.2 Pause, Stop, Hide</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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.
<p><u>2.3.1 Three Flashes or Below Threshold</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	The product does not contain flashing content.
<p><u>2.4.1 Bypass Blocks</u> (Level A)</p> <p>Also applies to:</p>	Supports	Each page contains a link at the top that goes directly to the main content area. Each page contains <i>ARIA</i> land-

Criteria	Conformance Level	Remarks and Explanations
<p>Revised Section 508</p> <ul style="list-style-type: none"> • 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 		<p>marks that identify the available regions.</p>
<p><u>2.4.2 Page Titled</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	<p>Supports</p>	<p>Each page contains a non-empty <code><title></code> element in the <code><head></code> section.</p>
<p><u>2.4.3 Focus Order</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	<p>Supports</p>	<p>Focusable components receive focus in an order that preserves meaning and operability.</p>
<p><u>2.4.4 Link Purpose (In Context)</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	<p>Supports</p>	<p>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.</p> <p>The authors can create hypertext links using text that describes the purpose of the hypertext.</p> <p>There is no control that allows the user to choose between short or long link text (G189 / SCR30).</p>

Criteria	Conformance Level	Remarks and Explanations
<p><u>2.5.1 Pointer Gestures</u> (Level A 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	Supports	The WebHelp Responsive output does not rely on path-based or multipoint gestures and does not provide controls that require complex gestures.
<p><u>2.5.2 Pointer Cancellation</u> (Level A 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	Supports	The product has operations that are activated on the pointer up event.
<p><u>2.5.3 Label in Name</u> (Level A 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	Supports	The names of the user interface components contain the text that is presented visually.
<p><u>2.5.4 Motion Actuation</u> (Level A 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	Supports	The product does not contain functionality that can be operated by device or user motion.
<p><u>3.1.1 Language of Page</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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.
<p><u>3.2.1 On Focus</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p>	Supports	No changes of context occur when any component receives focus.

Criteria	Conformance Level	Remarks and Explanations
<ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 		
<p><u>3.3.2 On Input</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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.
<p><u>3.3.1 Error Identification</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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 <i>aria-invalid</i> information is provided.
<p><u>3.3.2 Labels or Instructions</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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.
<p><u>4.1.1 Parsing</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p>	Partially Supports	Several HTML validation errors are reported by the W3C validator.

Criteria	Conformance Level	Remarks and Explanations
<ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 		
<p><u>4.1.2 Name, Role, Value</u> (Level A)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Partially Supports	The <i>Home</i> link from the breadcrumb does not have an associated <i>aria-label</i> .

Table 2: Success Criteria, Level AA

Criteria	Conformance Level	Remarks and Explanations
<p><u>1.2.4 Captions (Live)</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	No live audio content is used.
<p><u>1.2.5 Audio Description (Prerecorded)</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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.
<p><u>1.3.4 Orientation</u> (Level AA 2.1 only)</p> <p>Also applies to:</p>	Supports	Content does not restrict its view and operation to a single display orientation.

Criteria	Conformance Level	Remarks and Explanations
Revised Section 508 – Does not apply		
<p><u>1.3.5 Identify Input Purpose</u> (Level AA 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	Supports	The content does not contain input fields that collect information about the user.
<p><u>1.4.3 Contrast (Minimum)</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Partially Supports	The missing words element from the search results page does not have the contrast ratio 4.5:1.
<p><u>1.4.4 Resize text</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Partially Supports	<p>Text can be resized up to 200 percent without loss of content or functionality and without using assistive technology.</p> <p>Some text content has dimensions specified in pixels rather than em units.</p>
<p><u>1.4.5 Images of Text</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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.
<p><u>1.4.10 Reflow</u> (Level AA 2.1 only)</p> <p>Also applies to:</p>	Partially Supports	The majority of the content can be presented without loss of information

Criteria	Conformance Level	Remarks and Explanations
Revised Section 508 – Does not apply		<p>or functionality, and without requiring scrolling in two dimensions.</p> <p>Long URLs determine the page to display the horizontal scroll bar.</p>
<p><u>1.4.11 Non-text Contrast</u> (Level AA 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	Supports	(Cobalt template) There is no contrast issue regarding user interface components or graphical objects.
<p><u>1.4.12 Text Spacing</u> (Level AA 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	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.
<p><u>1.4.13 Content on Hover or Focus</u> (Level AA 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	Partially Supports	<p>Tooltips and submenus are not dismissible.</p> <p>Also, the tooltips are not hoverable.</p>
<p><u>2.4.5 Multiple Ways</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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	<p>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.</p> <p>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.</p>
<p><u>2.4.6 Headings and Labels</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p>	Supports	<p>Headings and labels describe the topic or purpose.</p> <p>DITA authors can ensure that this criterion is met.</p>

Criteria	Conformance Level	Remarks and Explanations
<ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 		
<p><u>2.4.7 Focus Visible</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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.
<p><u>3.1.2 Language of Parts</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	DITA authors can ensure that this criterion is met.
<p><u>3.2.3 Consistent Navigation</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 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	Repeated components appear in the same relative in each page.
<p><u>3.2.4 Consistent Identification</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p>	Partially Supports	The output uses labels, names, and text alternatives consistently for items that have the same functionality.

Criteria	Conformance Level	Remarks and Explanations
<ul style="list-style-type: none"> • 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 		<p>Text alternatives are provided for many instances of non-text content, with exceptions that include:</p> <ul style="list-style-type: none"> • Permalinks for subtopics and sections. • Enlarge images action. <p>The <i>Home</i> link from the breadcrumb does not have an associated <i>aria-label</i>.</p>
<p><u>3.3.3 Error Suggestion</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Does Not Support	<p>The Search input does not have the <i>aria-required</i> information set and does not contain a text description specifying that it is a required field.</p>
<p><u>3.3.4 Error Prevention (Legal, Financial, Data)</u> (Level AA)</p> <p>Also applies to:</p> <p>Revised Section 508</p> <ul style="list-style-type: none"> • 501 (Web)(Software) • 504.2 (Authoring Tool) • 602.3 (Support Docs) 	Supports	<p>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.</p>
<p><u>4.1.3 Status Messages</u>(Level AA 2.1 only)</p> <p>Also applies to:</p> <p>Revised Section 508 – Does not apply</p>	Supports	<p>The pages do not contain status messages as defined by this criterion.</p>

Table 3: Success Criteria, Level AAA

Criteria	Conformance Level	Remarks and Explanations
<p><u>1.2.6 Sign Language (Prerecorded)</u> (Level AAA)</p> <p>Revised Section 508 – Does not apply</p>	Not Evaluated	
<p><u>1.2.7 Extended Audio Description (Prerecorded)</u> (Level AAA)</p> <p>Revised Section 508 – Does not apply</p>	Not Evaluated	
<p><u>1.2.8 Media Alternative (Prerecorded)</u> (Level AAA)</p> <p>Revised Section 508 – Does not apply</p>	Not Evaluated	
<p><u>1.2.9 Audio-only (Live)</u> (Level AAA)</p> <p>Revised Section 508 – Does not apply</p>	Not Evaluated	
<p><u>1.3.6 Identify Purpose</u> (Level AAA 2.1 only)</p> <p>Revised Section 508 – Does not apply</p>	Not Evaluated	
<p><u>1.4.6 Contrast Enhanced</u> (Level AAA)</p> <p>Revised Section 508 – Does not apply</p>	Not Evaluated	
<p><u>1.4.7 Low or No Background Audio</u> (Level AAA)</p> <p>Revised Section 508 – Does not apply</p>	Not Evaluated	
<p><u>1.4.8 Visual Presentation</u> (Level AAA)</p> <p>Revised Section 508 – Does not apply</p>	Not Evaluated	
<p><u>1.4.9 Images of Text (No Exception) Control</u> (Level AAA)</p>	Not Evaluated	

Criteria	Conformance Level	Remarks and Explanations
Revised Section 508 – Does not apply		
<p data-bbox="145 322 603 405"><u>2.1.3 Keyboard (No Exception)</u> (Level AAA)</p> <p data-bbox="145 443 603 479">Revised Section 508 – Does not apply</p>	Not Evaluated	
<p data-bbox="145 546 496 582"><u>2.2.3 No Timing</u> (Level AAA)</p> <p data-bbox="145 620 603 656">Revised Section 508 – Does not apply</p>	Not Evaluated	
<p data-bbox="145 721 528 757"><u>2.2.4 Interruptions</u> (Level AAA)</p> <p data-bbox="145 795 603 831">Revised Section 508 – Does not apply</p>	Not Evaluated	
<p data-bbox="145 896 587 931"><u>2.2.5 Re-authenticating</u> (Level AAA)</p> <p data-bbox="145 969 603 1005">Revised Section 508 – Does not apply</p>	Not Evaluated	
<p data-bbox="145 1070 587 1106"><u>2.2.6 Timeouts</u> (Level AAA 2.1 only)</p> <p data-bbox="145 1144 603 1180">Revised Section 508 – Does not apply</p>	Not Evaluated	
<p data-bbox="145 1245 539 1281"><u>2.3.2 Three Flashes</u> (Level AAA)</p> <p data-bbox="145 1319 603 1355">Revised Section 508 – Does not apply</p>	Not Evaluated	
<p data-bbox="145 1420 639 1503"><u>2.3.3 Animation from Interactions</u> (Level AAA 2.1 only)</p> <p data-bbox="145 1541 603 1576">Revised Section 508 – Does not apply</p>	Not Evaluated	
<p data-bbox="145 1644 475 1680"><u>2.4.8 Location</u> (Level AAA)</p> <p data-bbox="145 1718 603 1753">Revised Section 508 – Does not apply</p>	Not Evaluated	
<p data-bbox="145 1818 603 1901"><u>2.4.9 Link Purpose (Link Only)</u> (Level AAA)</p> <p data-bbox="145 1939 603 1975">Revised Section 508 – Does not apply</p>	Not Evaluated	
<p data-bbox="145 2042 603 2078"><u>2.4.10 Section Headings</u> (Level AAA)</p>	Not Evaluated	

Criteria	Conformance Level	Remarks and Explanations
Revised Section 508 – Does not apply		
<u>2.5.5 Target Size</u> (Level AAA 2.1 only)	Not Evaluated	
Revised Section 508 – Does not apply		
<u>2.5.6 Concurrent Input Mechanisms</u> (Level AAA 2.1 only)	Not Evaluated	
Revised Section 508 – Does not apply		
<u>3.1.3 Unusual Words</u> (Level AAA)	Not Evaluated	
Revised Section 508 – Does not apply		
<u>3.1.4 Abbreviations</u> (Level AAA)	Not Evaluated	
Revised Section 508 – Does not apply		
<u>3.1.5 Reading Level</u> (Level AAA)	Not Evaluated	
Revised Section 508 – Does not apply		
<u>3.1.6 Pronunciation</u> (Level AAA)	Not Evaluated	
Revised Section 508 – Does not apply		
<u>3.2.5 Change on Request</u> (Level AAA)	Not Evaluated	
Revised Section 508 – Does not apply		
<u>3.3.5 Help</u> (Level AAA)	Not Evaluated	
Revised Section 508 – Does not apply		
<u>3.3.6 Error Prevention (All)</u> (Level AAA)	Not Evaluated	
Revised Section 508 – Does not apply		

Revised Section 508 Report

N/A

Chapter 3: Functional Performance Criteria (FPC)

Criteria	Conformance Level	Remarks and Explanations
302.1 Without Vision	Partially Supports	<p>Most of the content is accessible without vision with exceptions that include:</p> <ul style="list-style-type: none"> • 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	<p>Most of the content is accessible with limited vision with exceptions that include:</p> <ul style="list-style-type: none"> • 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.
302.3 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.

Criteria	Conformance Level	Remarks and Explanations
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 multipoint 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 multipoint 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 41)	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.

Criteria	Conformance Level	Remarks and Explanations
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	<p>The majority of object roles, state(s), properties, boundary, name, and description are programmatically determinable.</p> <p>The <i>Home</i> link from the breadcrumb does not have an associated <i>aria-label</i>.</p>
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	<p>Information, structure, and relationships conveyed through presentation can be programmatically determined or are available in text.</p> <p>See WCAG 1.3.1 (on page 42).</p>
502.3.7 Hierarchical Relationships	Supports	The content is hierarchically structured using language-specific ele-

Criteria	Conformance Level	Remarks and Explanations
		ments 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.
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 41)	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 57) and Chapter 5 (on page 58) sections.
602.3 Electronic Support Documentation	See the WCAG 2.x section (on page 41)	See information in the WCAG section.

Criteria	Conformance Level	Remarks and Explanations
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.

3.

Adding Oxygen Feedback to WebHelp Responsive Documentation

You can add Oxygen Feedback in WebHelp Responsive for DITA output to benefit from a modern commenting system, advanced ready-to-use search engine, administrative interface, and **Oxygen XML Editor/Author** integration.

Comments Component

A comments component is presented 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.

External Search Engine

Oxygen Feedback can be [configured as an external search engine](#) for the **Oxygen WebHelp Responsive** output. This function can be enabled from the *Content Indexing and Search* section that is available in the *Version Settings* page.

Administration Interface

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.

Oxygen XML Editor/Author Integration


An add-on is 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.

Deploying the Oxygen Feedback Comments Component

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 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.
5. [Optional] To configure Oxygen Feedback as an external search engine check the **Enable content indexing** option.
6. Click **Continue**.
7. 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 and continue with these steps:
 - i. Copy the generated HTML fragment and click **Finish**.
 - ii. In *Oxygen XML Editor/Author*, open the **Configure Transformation Scenario(s)** dialog box.
 - iii. Select and duplicate the **DITA Map WebHelp Responsive** scenario.
 - iv. Go to the **Feedback** tab.
 - v. 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. Copy the generated HTML fragment and click **Finish**.
 - ii. Create an XML file (for example, `feedback-install.xml`) with the generated installation fragment.
 - iii. 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
```
8. [Optional] If you want the **Oxygen Feedback** comments component to fill the entire page width, contribute a custom CSS file (use the `args.css` parameter to reference it) that contains the following style rule:


```
div.footer {  
  float: none;  
}
```

For more details about **Oxygen Feedback**, how to configure settings, moderate comments, view statistics, and much more, see the [Oxygen Feedback user guide](#).

Also, to see a demonstration of **Oxygen Feedback** being integrated into WebHelp Responsive output, watch our Webinar: [DITA Publishing and Feedback with Oxygen Tools](#).

4.

Developer Reference

The section is designed for developers to provide advanced information about the Oxygen XML WebHelp Responsive plugin. The information in this section will help you to extend or customize the output and provide an overview of the plugin architecture.

The WebHelp Responsive plugin is a [DITA-OT](#) plugin that provides the ability to transform DITA documentation to HTML format. It extends the [DITA to HTML5 plugin](#) by using its XSLT stylesheets and ANT targets.

Related Information:

[DITA to HTML5 DITA-OT plugin](#)

Overview of WebHelp DITA-OT Processing Stages

The WebHelp Responsive plugin inherits the multi-stage processing mode from the [DITA-OT \(on page 212\)](#) publishing engine. Each stage in the process examines some or all of the content. Some stages result in temporary files that are used in later steps, while other stages result in updated copies of the DITA content. Most of the processing takes place in a temporary working directory, and the source files themselves are never modified.

The most important steps (*Ant* targets) in the WebHelp Responsive transformation process are:

whr-init

Creates a set of initializations required by the next processing steps such as: initialize the plugin Java CLASSPATH, load the [Oxygen Publishing Template](#), or set the default values for various properties.

preprocess

This is a [step](#) defined in the [DITA-OT](#) processor representing a set of sub-steps that typically runs at the beginning of every [DITA-OT](#) transformation. Each step or stage corresponds to an Ant target in the build pipeline; the `preprocess` target calls the entire set of steps.

whr-detect-lang

Detects the documentation language by looking into the DITA map file. If not detected, it uses the value of the `default.language` parameter.

whr-collect-indexterms

Collects the index terms from DITA topics and write them in [WebHelp Output Directory/index.xml](#). The `index.xml` file is used later by the `whr-create-indexterms-`

page step to generate the [index terms HTML page \(on page 101\)](#) ([WebHelp Output Directory/indexTerms.html](#)).

whr-create-props-file

Serializes the transformation parameters in XML and JS formats so they can be used in the next XSLT processing steps or JavaScript.

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.

whr-toc-xml

Generates the `toc.xml` file in the temporary directory.

whr-nav-links

Generates the navigational links for all DITA topics such as menu, table of contents, or breadcrumb links.

whr-context-help-map

Generates the `context-help-map.xml` file in the output folder. This file is used by the [Context-Sensitive WebHelp Responsive system \(on page 31\)](#).

whr-sitemap

Generates the `sitemap.xml` file in the output folder. This file is used for [Search Engine Optimization \(on page 170\)](#).

whr-copy-resources

Copies all the resources (logo, favicon, JavaScript files, CSS files, etc.) that are needed by the WebHelp transformation to the output folder.

whr-create-topic-pages

Generates an HTML file for each DITA topic.

Implementation is done by running an XSLT transformation that processes the [topic layout page \(on page 93\)](#) with `DITA-OT-DIR/plugins/com.oxygenxml.webhelp.responsive/templates/xsl/dita2webhelp/dita2webhelp.xsl` XSLT file for all DITA topics. You can extend this step by using the `com.oxygenxml.webhelp.xsl.dita2webhelp (on page 119)` extension point.

whr-create-main-page

Generates the [WebHelp main page \(on page 88\)](#) (`index.html`) in the output folder.

Implementation is done by running an XSLT transformation that processes the [main layout page \(on page 88\)](#) with the `DITA-OT-DIR/plugins/com.oxygenxml.webhelp.responsive/templates/xsl/mainFiles/`

`createMainPage.xsl` XSLT file. You can extend this step by using the `com.oxygenxml.webhelp.xsl.createMainPage` (on page 119) extension point.

whr-create-search-page

Generates the [WebHelp search results page](#) (on page 98) (`search.html`) in the output folder.

Implementation is done by running an XSLT transformation that processes the [search results page](#) (on page 98) with `DITA-OT-DIR/plugins/com.oxygenxml.webhelp.responsive/templates/xsl/mainFiles/createSearchPage.xsl` XSLT file. You can extend this step by using the `com.oxygenxml.webhelp.xsl.createSearchPage` (on page 119) extension point.

whr-create-indexterms-page

Generates the [WebHelp index terms HTML page](#) (on page 101) (`indexTerms.html`) in the output folder.

Implementation is done by running an XSLT transformation that transforms the [WebHelp Output Directory/index.xml](#) generated by the `whr-collect-indexterms` step with `DITA-OT-DIR/plugins/com.oxygenxml.webhelp.responsive/templates/xsl/mainFiles/createIndextermsPage.xsl` XSLT file. You can extend this step by using the `com.oxygenxml.webhelp.xsl.createIndexTermsPage` (on page 120) extension point.

whr-search-index

Processes the generated HTML (for all DITA topics) to generate an index file. This index is used to implement the WebHelp search function.



Note:

The WebHelp Responsive plugin uses the XSLT stylesheets from the [DITA-OT HTML5](#) plugin.

Related Information:

[DITA-OT Preprocessing](#)

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.

**Tip:**

You can start creating publishing templates by using the **Oxygen Styles Basket**. <https://styles.oxygenxml.com>

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 8. Oxygen Publishing Template Package (WebHelp Responsive)

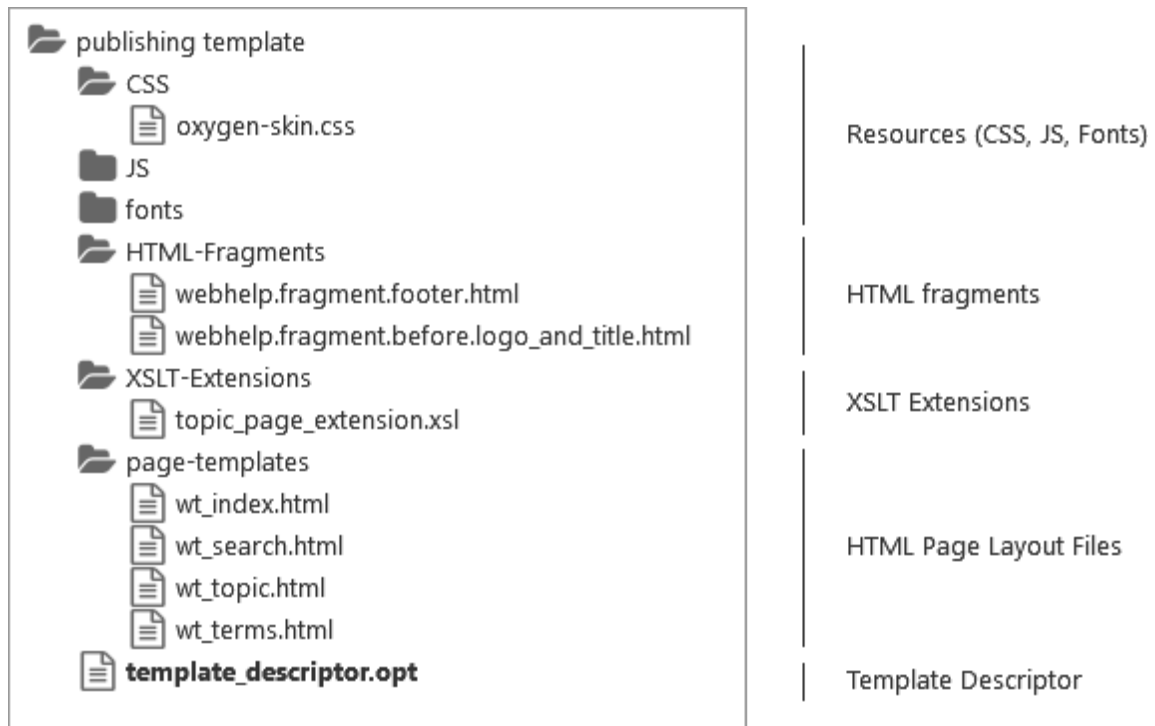


Figure 9. Oxygen Publishing Template Package (PDF)



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 \(on page 122\)](#)

[How to Edit a Packed Publishing Template \(on page 125\)](#)


[How to Add a Publishing Template to the Publishing Templates Gallery \(on page 125\)](#)

[How to Share a Publishing Template \(on page \)](#)

Publishing Templates Gallery

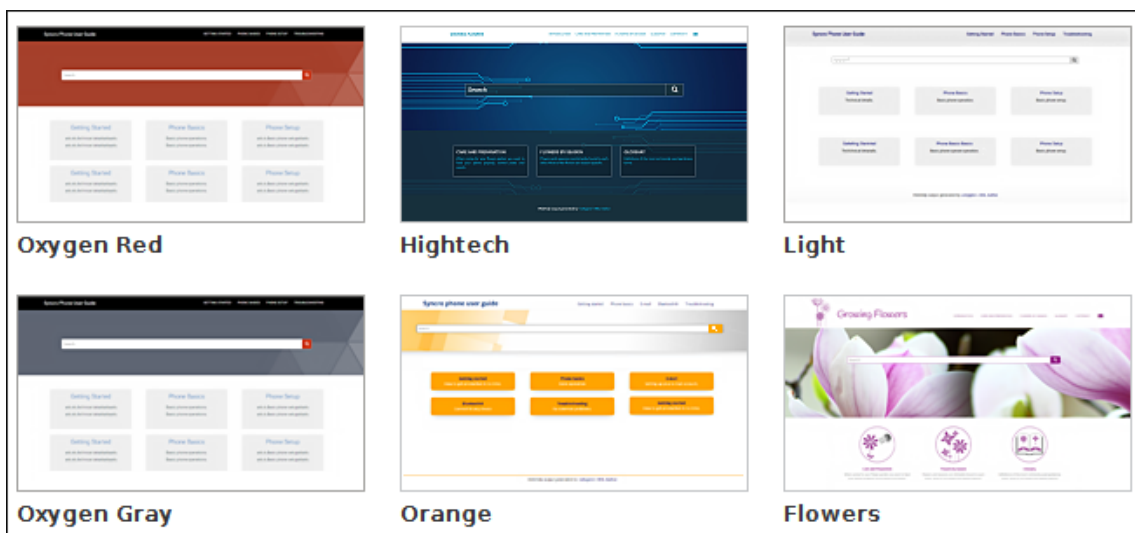
Oxygen XML WebHelp Responsive plugin comes bundled with a variety of built-in templates. You can use one of them to publish your documentation or as a starting point for a new publishing template.

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.

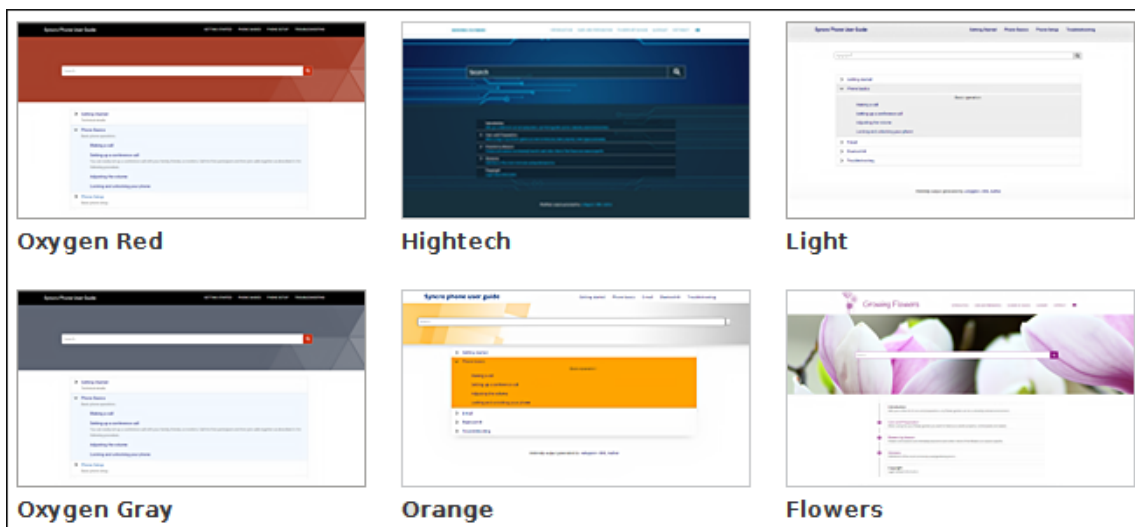
Tiles Templates

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*). 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 \(on page 74\)](#), [template preview image \(on page 74\)](#), [resources \(on page 75\)](#) (such as CSS, JS, fonts, logos), [transformation parameters \(on page 76\)](#), [HTML fragment extensions \(on page 78\)](#) (used to add fragments to placeholders), [XSLT extensions \(on page 78\)](#), or [HTML page layout files \(on page 87\)](#).

```

<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>
  </webhelp>
</publishing-template>

```

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.



Warning:

All paths set in the `@file` attribute must be relative.

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.
- **Additional Resources (graphics, JS, fonts, folders)** - For other resources (such as images referenced in CSS, JavaScript, fonts, entire folders, 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 10. Template Resources Mapping



Related Information:

[How to Add a Favicon in WebHelp Systems \(on page 150\)](#)

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 \(on page 104\)](#) 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) and adding it to the templates gallery (on page 125), when you select the template in the transformation scenario dialog box in **Oxygen XML Editor/Author**, 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 can include one or more [supported XSLT extension points](#) (*on page 118*). 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 a full list of the supported extension points, see: [XSLT-Import and XSLT-Parameter Extension Points](#) (*on page 118*).



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 Placeholders

The HTML pages contain component placeholders that can be used to insert custom HTML fragments either by specifying a **well-formed** XHTML fragment or referencing a path to a file that contains a **well-formed** XHTML fragment (for details on how the file or fragment needs to be constructed, see [How to Insert Custom HTML Content](#) (*on page 134*)).

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" />

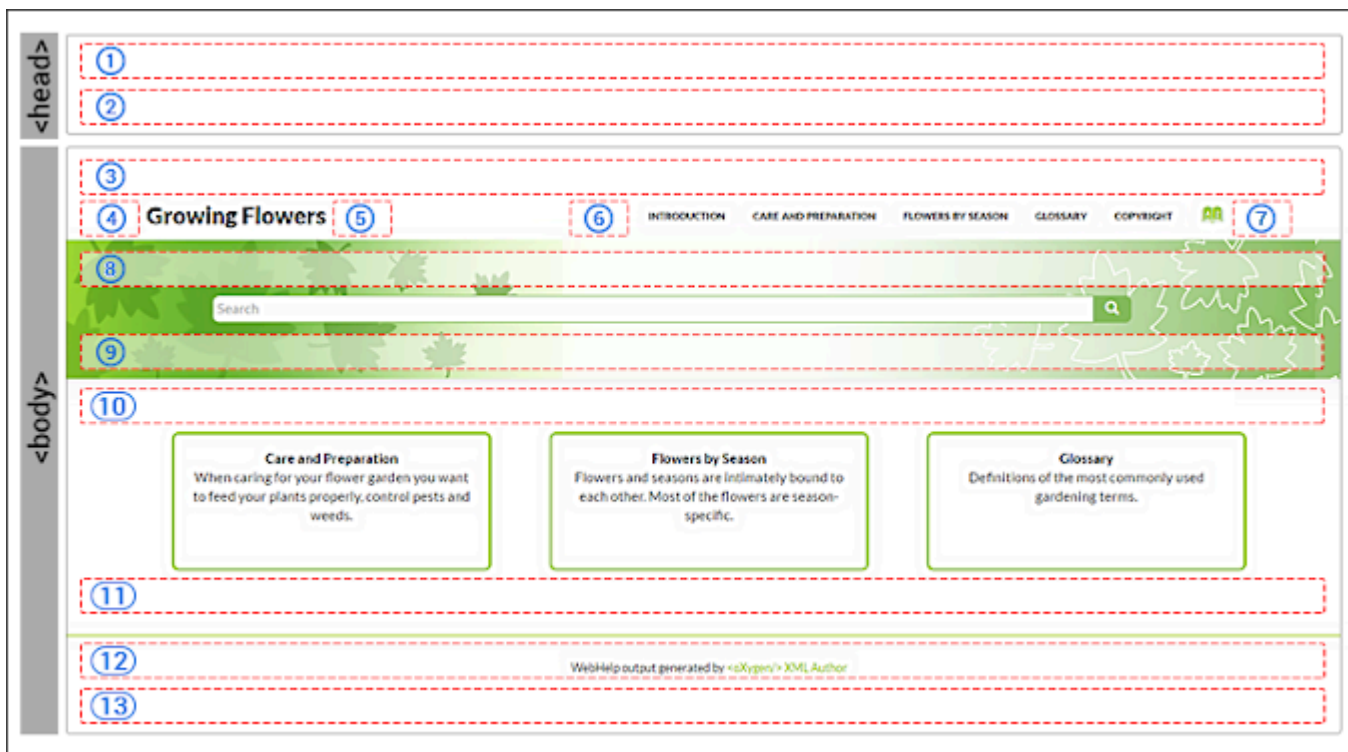
<fragment
  file="html-fragments/webhelp_fragment_footer.html"
  placeholder="webhelp.fragment.footer" />
</html-fragments>

```

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. Some of the placeholder parameters are global and can be used in all type of pages (*main page*, *topic page*, *search results page*, *index terms page*), while others are applicable for certain type of pages. The following diagram illustrates the predefined placeholders that are global (can be used in any of the types of pages).

Figure 11. Global Predefined Placeholders Diagram



1. Header (on page 80)
2. After Header (on page 80)
3. Before Body (on page 80)
4. Before Logo and Title (on page 80)
5. After Logo and Title (on page 80)
6. Before Top Menu (on page 80)
7. After Top Menu (on page 80)
8. Before Search Input (on page 80)
9. After Search Input (on page 80)
10. Before Main Content (on page 80)

- 11. After Main Content (*on page 80*)
- 12. Footer (*on page 80*)
- 13. After Body (*on page 80*)

Global Placeholder Parameters

The following placeholder parameters can be used in any of the type of pages (*main page, topic page, search results page, index terms page*). The parameter values can be either a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment:

- **1 = webhelp.fragment.head** - Displays the specified XHTML fragment in the header section.
- **2 = webhelp.fragment.after.header** - Displays the specified XHTML fragment after the header section.
- **3 = webhelp.fragment.before.body** - Displays the specified XHTML fragment before the body.
- **4 = webhelp.fragment.before.logo_and_title** - Displays the specified XHTML fragment before the logo and title.
- **5 = webhelp.fragment.after.logo_and_title** - Displays the specified XHTML fragment after the logo and title.
- **6 = webhelp.fragment.before.top_menu** - Displays the specified XHTML fragment before the top menu.
- **7 = webhelp.fragment.after.top_menu** - Displays the specified XHTML fragment after the top menu.
- **8 = webhelp.fragment.before.search.input** - Displays the specified XHTML fragment before the search input component.
- **9 = webhelp.fragment.after.search.input** - Displays the specified XHTML fragment after the search input component.
- **10 = webhelp.fragment.before.main.content.area** - Displays the specified XHTML fragment before the main content area
- **11 = webhelp.fragment.after.main.content.area** - Displays the specified XHTML fragment after the main content area.
- **12 = webhelp.fragment.footer** - Displays the specified XHTML fragment in the footer section.
- **13 = webhelp.fragment.after.body** - Displays the specified XHTML fragment after the body.

Main Page Placeholder Parameters

The following placeholder parameters can be used in the *main page*. The parameter values can be either a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment:

- **webhelp.fragment.head.main.page** - Displays the specified XHTML fragment in the header section.
- **webhelp.fragment.after.header.main.page** - Displays the specified XHTML fragment after the header section.
- **webhelp.fragment.before.body.main.page** - Displays the specified XHTML fragment before the body.
- **webhelp.fragment.before.search.input.main.page** - Displays the specified XHTML fragment before the search input component.
- **webhelp.fragment.after.search.input.main.page** - Displays the specified XHTML fragment after the search input component.

- **webhelp.fragment.before.main.content.area.main.page** - Displays the specified XHTML fragment before the main content area
- **webhelp.fragment.after.main.content.area.main.page** - Displays the specified XHTML fragment after the main content area.
- **webhelp.fragment.after.body.main.page** - Displays the specified XHTML fragment after the body.
- **webhelp.fragment.before.toc_or_tiles** - Displays the specified XHTML fragment before the main table of contents or tiles component on the main page.
- **webhelp.fragment.after.toc_or_tiles** - Displays the specified XHTML fragment after the main table of contents or tiles component on the main page.

Topic Page Placeholder Parameters

The following placeholder parameters can be used in the *topic page*. The parameter values can be either a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment:

- **webhelp.fragment.head.topic.page** - Displays the specified XHTML fragment in the header section.
- **webhelp.fragment.after.header.topic.page** - Displays the specified XHTML fragment after the header section.
- **webhelp.fragment.before.body.topic.page** - Displays the specified XHTML fragment before the body.
- **webhelp.fragment.before.search.input.topic.page** - Displays the specified XHTML fragment before the search input component.
- **webhelp.fragment.after.search.input.topic.page** - Displays the specified XHTML fragment after the search input component.
- **webhelp.fragment.before.main.content.area.topic.page** - Displays the specified XHTML fragment before the main content area
- **webhelp.fragment.after.main.content.area.topic.page** - Displays the specified XHTML fragment after the main content area.
- **webhelp.fragment.after.body.topic.page** - Displays the specified XHTML fragment after the body.
- **webhelp.fragment.before.topic.toolbar** - Displays the specified XHTML fragment before the toolbar buttons above the topic content in the topic page.
- **webhelp.fragment.after.topic.toolbar** - Displays the specified XHTML fragment after the toolbar buttons above the topic content in the topic page.
- **webhelp.fragment.before.topic.breadcrumb** - Displays the specified XHTML fragment before the breadcrumb component in the topic page.
- **webhelp.fragment.after.topic.breadcrumb** - Displays the specified XHTML fragment after the breadcrumb component in the topic page.
- **webhelp.fragment.before.publication.toc** - Displays the specified XHTML fragment before the publication's table of contents component in the topic page.
- **webhelp.fragment.after.publication.toc** - Displays the specified XHTML fragment after the publication's table of contents component in the topic page.
- **webhelp.fragment.before.topic.content** - Displays the specified XHTML fragment before the topic's main content in the topic page.

- **webhelp.fragment.after.topic.content** - Displays the specified XHTML fragment after the topic's main content in the topic page.
- **webhelp.fragment.before.feedback** - Displays the specified XHTML fragment before the **Oxygen Feedback** commenting component in the topic page.
- **webhelp.fragment.after.feedback** - Displays the specified XHTML fragment after the **Oxygen Feedback** commenting component in the topic page.
- **webhelp.fragment.before.topic.toc** - Displays the specified XHTML fragment before the topic's table of contents component in the topic page.
- **webhelp.fragment.after.topic.toc** - Displays the specified XHTML fragment after the topic's table of contents component in the topic page.

Search Results Page Placeholder Parameters

The following placeholder parameters can be used in the *search results page*. The parameter values can be either a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment:

- **webhelp.fragment.head.search.page** - Displays the specified XHTML fragment in the header section.
- **webhelp.fragment.after.header.search.page** - Displays the specified XHTML fragment after the header section.
- **webhelp.fragment.before.body.search.page** - Displays the specified XHTML fragment before the body.
- **webhelp.fragment.before.search.input.search.page** - Displays the specified XHTML fragment before the search input component.
- **webhelp.fragment.after.search.input.search.page** - Displays the specified XHTML fragment after the search input component.
- **webhelp.fragment.before.main.content.area.search.page** - Displays the specified XHTML fragment before the main content area
- **webhelp.fragment.after.main.content.area.search.page** - Displays the specified XHTML fragment after the main content area.
- **webhelp.fragment.after.body.search.page** - Displays the specified XHTML fragment after the body.
- **webhelp.google.search.script** - Replaces the search input component with a Google search component.

Index Terms Page Placeholder Parameters

The following placeholder parameters can be used in the *search results page*. The parameter values can be either a **well-formed** XHTML fragment or a path to a file that contains a **well-formed** XHTML fragment:

- **webhelp.fragment.head.terms.page** - Displays the specified XHTML fragment in the header section.
- **webhelp.fragment.after.header.terms.page** - Displays the specified XHTML fragment after the header section.
- **webhelp.fragment.before.body.terms.page** - Displays the specified XHTML fragment before the body.
- **webhelp.fragment.before.search.input.terms.page** - Displays the specified XHTML fragment before the search input component.

- **webhelp.fragment.after.search.input.terms.page** - Displays the specified XHTML fragment after the search input component.
- **webhelp.fragment.before.main.content.area.terms.page** - Displays the specified XHTML fragment before the main content area
- **webhelp.fragment.after.main.content.area.terms.page** - Displays the specified XHTML fragment after the main content area.
- **webhelp.fragment.after.body.terms.page** - Displays the specified XHTML fragment after the body.

Using String Values in Placeholder Parameter Values

If you use strings for values of HTML fragment placeholder parameter values, the string values are written to files in the transformation's temporary directory. The values of the associated parameters reference the paths of the temporary files. This means that the HTML fragments will have a uniform processing in the *WebHelp's XSLT Module*.

Example:

Suppose the placeholder parameter has the following string value:

```
# String value
webhelp.fragment.welcome = <p>This is an HTML paragraph.</p>
```

A new file that contains the parameter's value is created:

```
[temp-dir]/whr-html-fragments/webhelp_fragment_welcome.xml
```

```
<p>This is an HTML paragraph.</p>
```

The parameter's value then becomes:

```
# Absolute file path as value
webhelp.fragment.welcome= [temp-dir]/whr-html-fragments/webhelp_fragment_welcome.xml
```

Related Information:

[How to Insert Custom HTML Content \(on page 134\)](#)

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.xml`:

```
<!-- 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.xml` 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.

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>
  </webhelp>
</publishing-template>

```

```

</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>

```

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. Instead, use any of the other methods described in [Publishing Template Package Contents for WebHelp Responsive Customizations \(on page 71\)](#).

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](http://www.oxygenxml.com/webhelp/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 \(on page 212\)](#).

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>

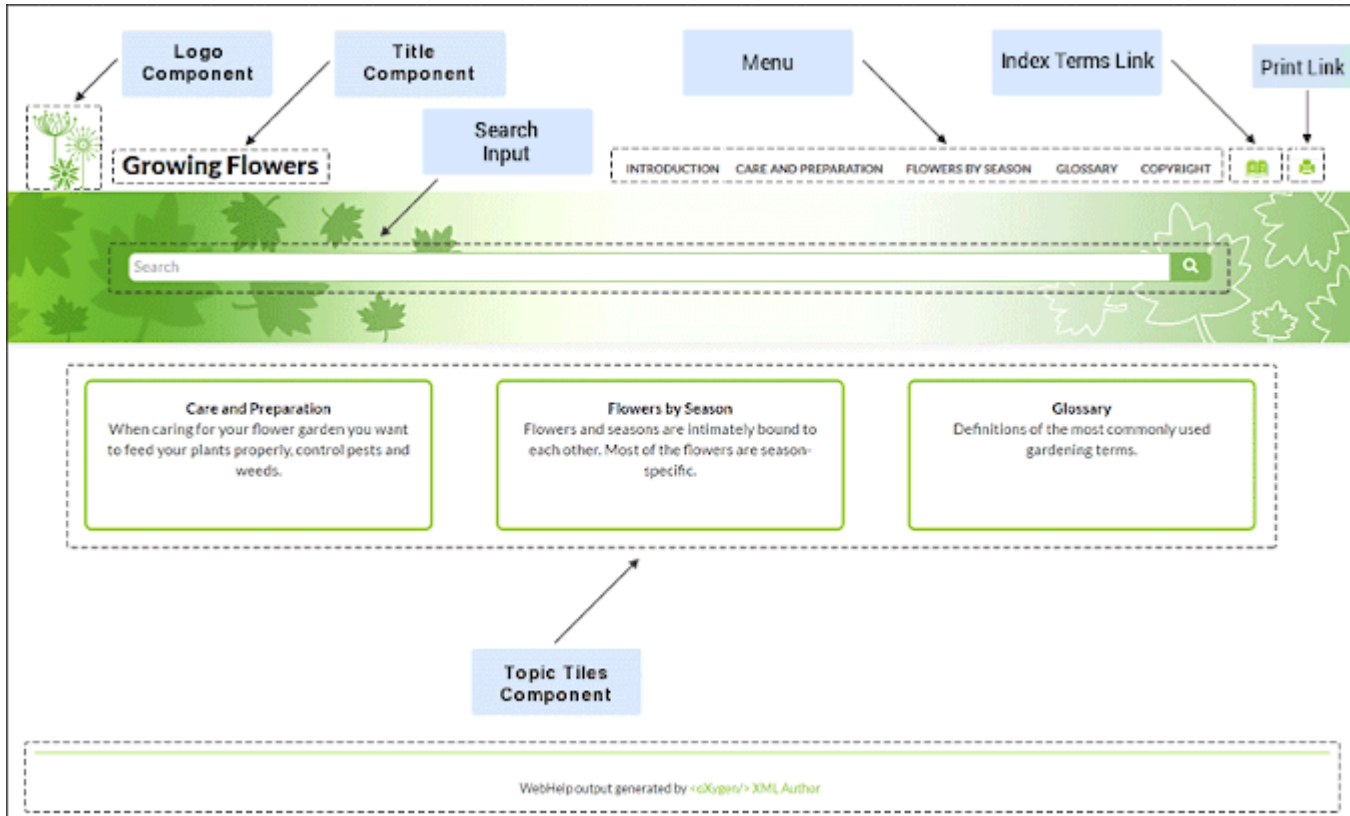
```

Main Page

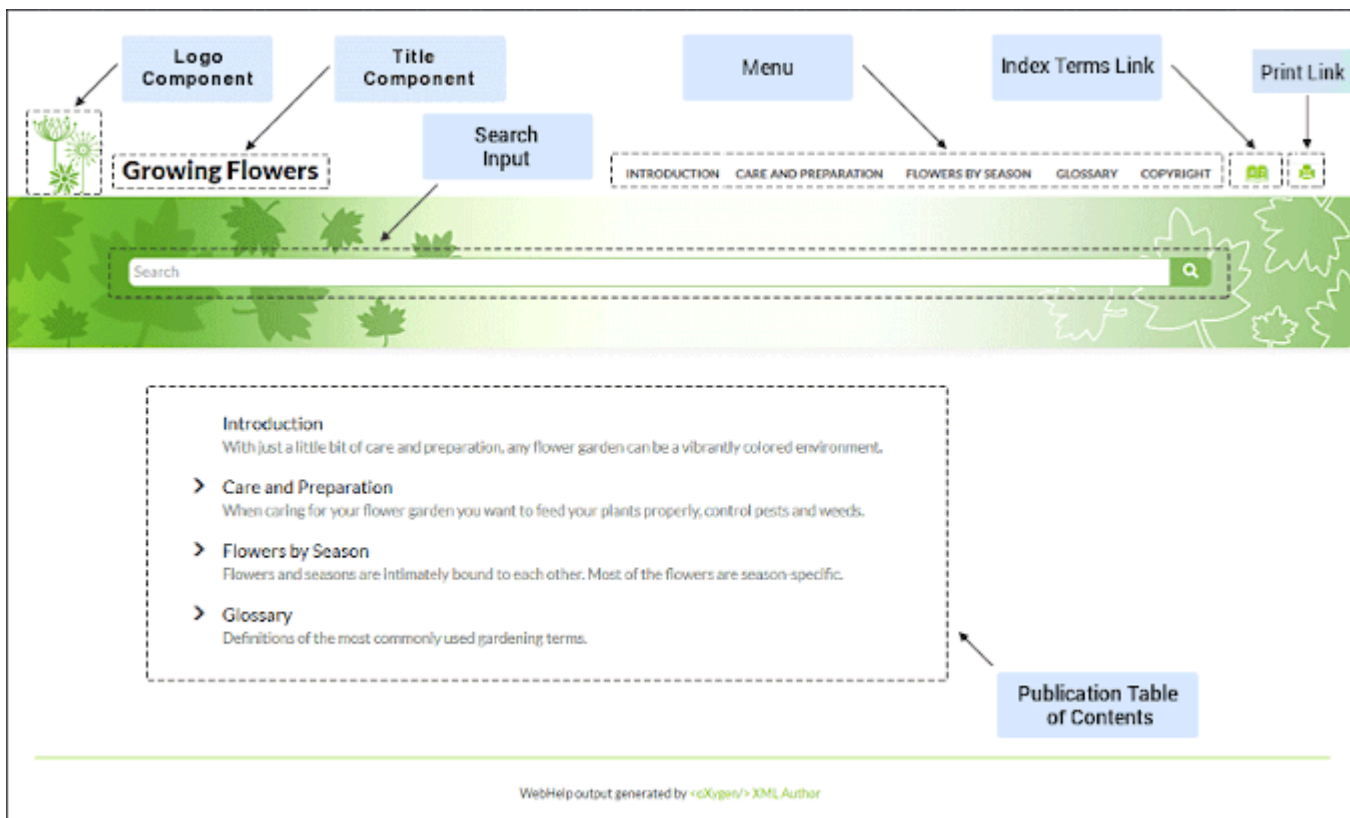
The *Main Page* is the home page generated in the WebHelp Responsive output. The name of the HTML file that defines this page is `wt_index.html` and it is located in the following directory: [DITA-OT-DIR/plugins/com.oxygenxml.webhelp.responsive/oxygen-webhelp/page-templates](http://www.oxygenxml.com/webhelp/plugins/com.oxygenxml.webhelp.responsive/oxygen-webhelp/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 12. Examples of Main Page Components for a Tiles Style of Layout



1. Publication Logo (on page 90)
2. Publication Title (on page 90)
3. Search Input (on page 91)
4. Main Menu (on page 91)
5. Index Terms Link (on page 92)
6. Topic Tiles (on page 91)
7. Print Link (on page 91)

Figure 13. Examples of Main Page Components for a Tree Style of Layout

1. Publication Logo (on page 90)
2. Publication Title (on page 90)
3. Search Input (on page 91)
4. Main Menu (on page 91)
5. Index Terms Link (on page 92)
6. Table of Contents (on page 92)
7. Print Link (on page 91)

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 \(on page 143\)](#).

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_shordesc/>
  </div>
</whc:webhelp_tile>
```

For information about customizing the tiles, see [How to Configure the Tiles on the WebHelp Responsive Main Page \(on page 147\)](#).

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 component 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/>
```

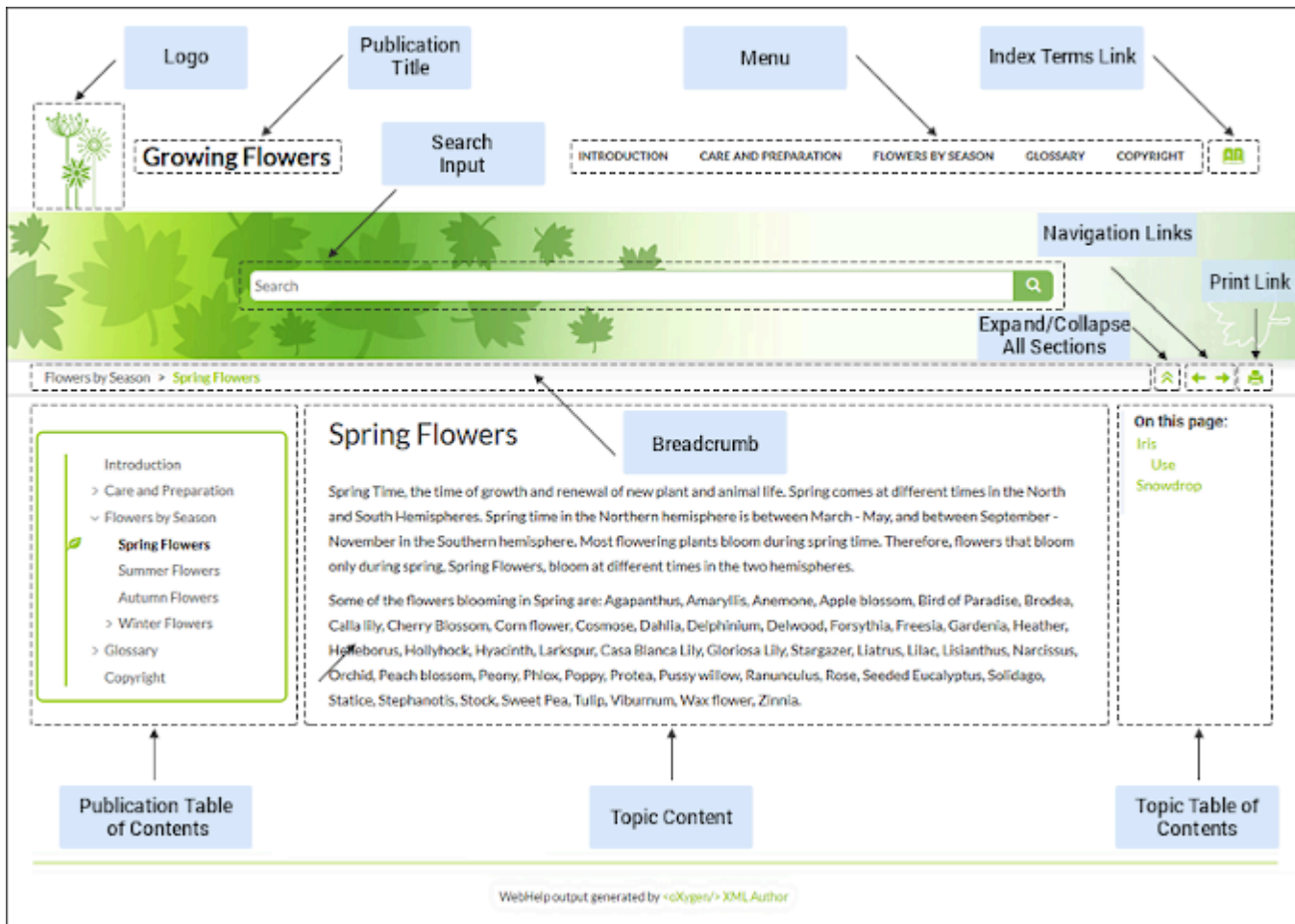
In the output, you will find a link to the skin resources.

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 14. Examples of Topic Page Components



1. Publication Logo (on page 95)
2. Publication Title (on page 94)
3. Search Input (on page 95)
4. Main Menu (on page 97)
5. Index Terms Link (on page 97)
6. Expand/Collapse All Sections (on page 97)
7. Navigation Links (on page 95)
8. Print Link (on page 96)
9. Breadcrumb (on page 95)
10. Publication Table of Contents (on page 96)
11. Topic Content (on page 96)
12. Topic Table of Contents (on page 96)

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 component 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 component 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 component will contain links to the topics referenced in the DITA map. It also includes an expand/collapse button (either `<` to collapse or the `>` to expand).

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 component will contain links to the sections within the current topic. It also includes an expand/collapse button (either `>` to collapse or the `<` to expand).

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 \(on page 143\)](#).

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 component 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/>
```

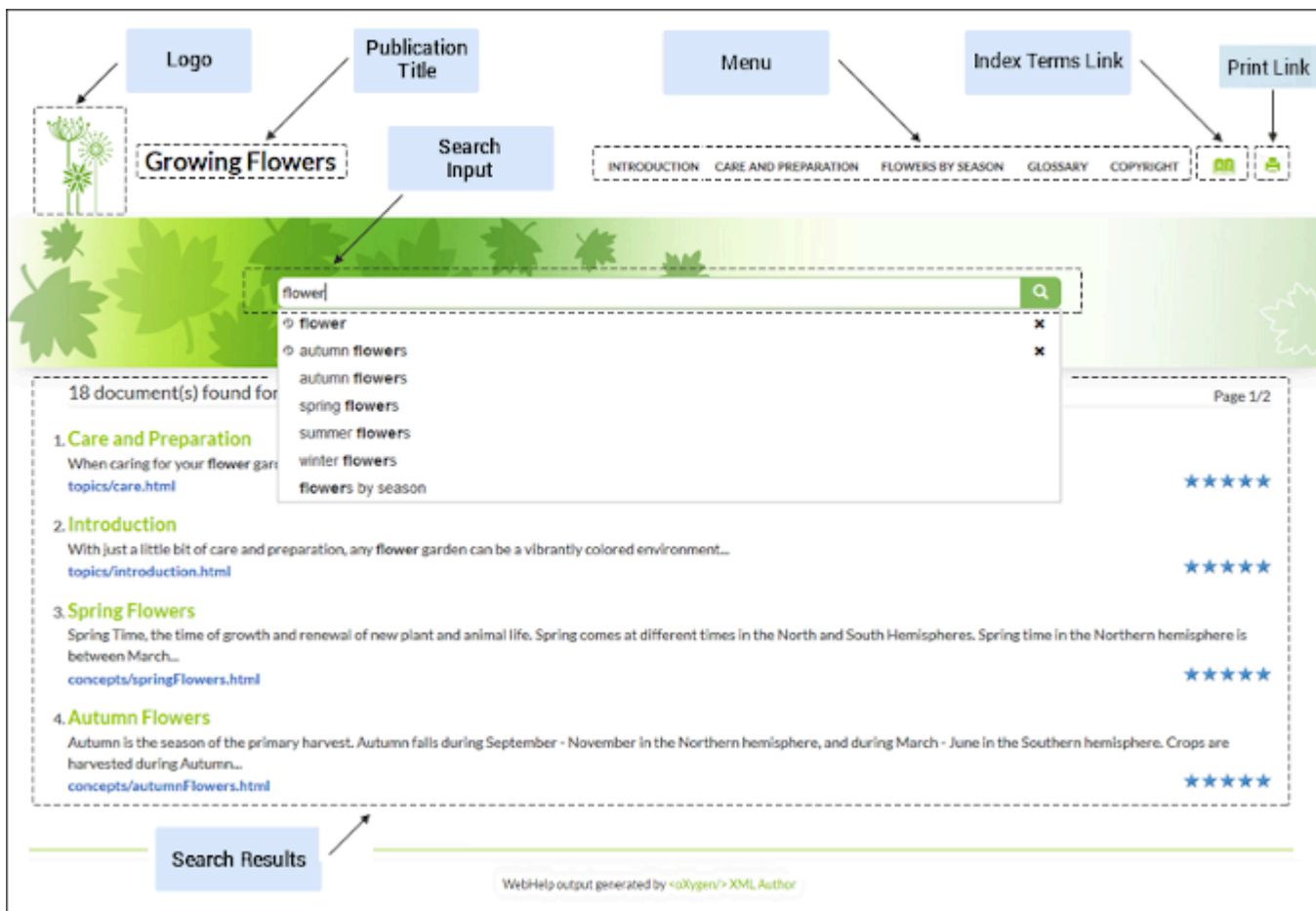
In the output, you will find a link to the skin resources.

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 15. Examples of Search Results Page Components



1. Publication Logo (on page 99)
2. Publication Title (on page 99)
3. Search Input (on page 100)
4. Main Menu (on page 100)
5. Index Terms Link (on page 101)
6. Search Results (on page 100)
7. Print Link (on page 100)

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 \(on page 143\)](#).

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 component 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.

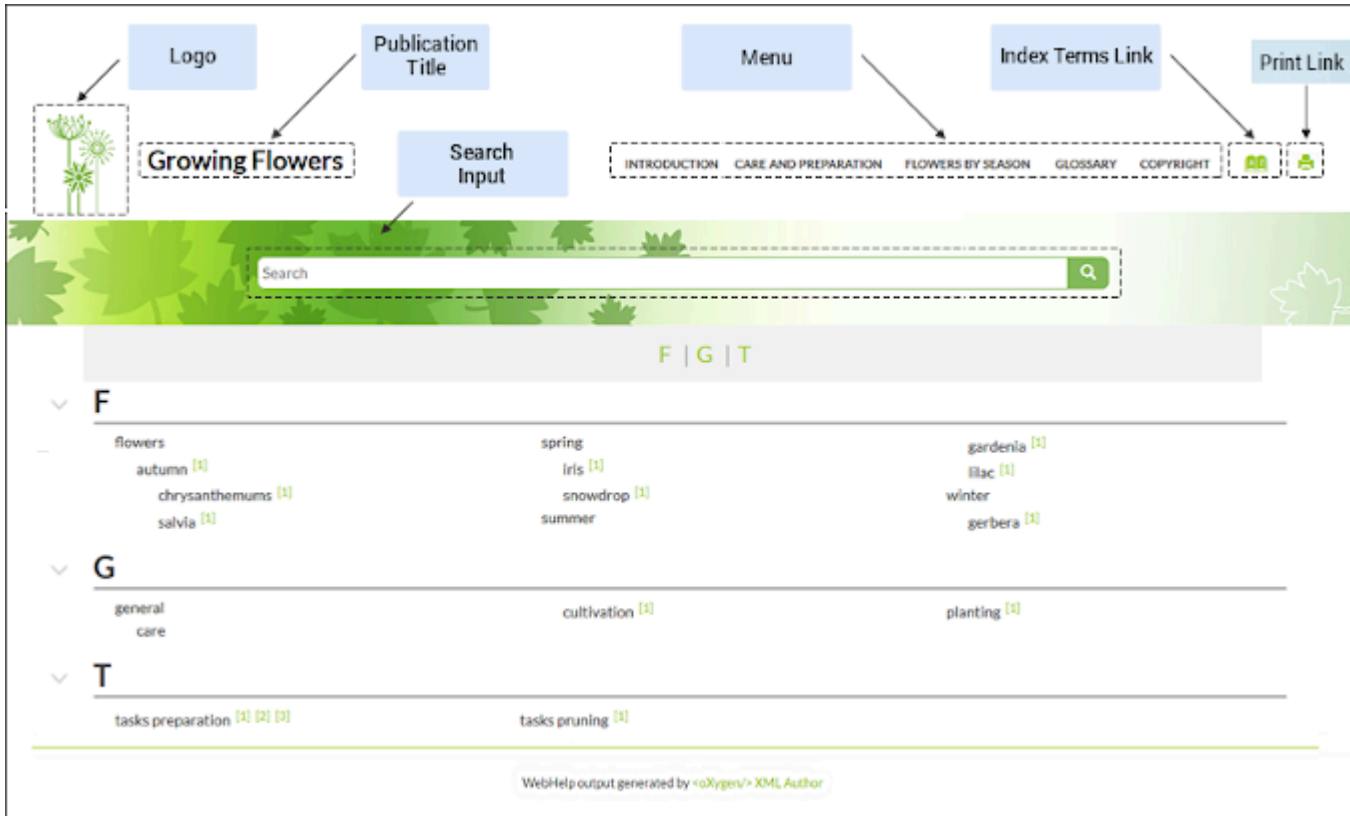
Index Terms Page

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 16. Example of Index Terms Page Components



1. Publication Logo (on page 102)
2. Publication Title (on page 102)
3. Search Input (on page 103)
4. Main Menu (on page 103)
5. Index Terms Link (webhelp_indexterms_link) (on page 103)
6. Print Link (on page 103)

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 \(on page 143\)](#).

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 component 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.

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.

Context Sensitive Help Parameter

webhelp.csh.disable.topicID.fallback

Specifies whether or not topic ID *fallbacks* are enabled when computing the mapping of context sensitive help and `resourceid` information is not available. Possible values are **false** (default) and **true**.

HTML Fragment Extension Parameters

webhelp.enable.html.fragments.cleanup

Enables or disables the automatic conversion of HTML fragments to well-formed XML. If set to **true** (default), the transformation automatically converts non-well-formed HTML content to a well-formed XML equivalent. If set to **false**, the transformation will fail if at least one HTML fragment is not well-formed.

webhelp.enable.scroll.to.search.term

Specifies whether or not the page should scroll to the first search term when opening the search results page. Possible values are **no** (default) and **true**.

webhelp.fragment.after.body

This parameter can be used to display a given XHTML fragment after the body in all types of pages. 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.body.main.page

This parameter can be used to display a given XHTML fragment after the body 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.body.search.page

This parameter can be used to display a given XHTML fragment after the body in the search results 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.body.terms.page

This parameter can be used to display a given XHTML fragment after the body in the index terms 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.body.topic.page

This parameter can be used to display a given XHTML fragment after the body in the topic 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.feedback

This parameter can be used to display a given XHTML fragment after the **Oxygen Feedback** commenting component in the topic 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.header

This parameter can be used to display a given XHTML fragment after the header section in all types of pages. 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.header.main.page

This parameter can be used to display a given XHTML fragment after the header section 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.header.search.page

This parameter can be used to display a given XHTML fragment after the header section in the search results 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.header.terms.page

This parameter can be used to display a given XHTML fragment after the header section in the index terms 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.header.topic.page

This parameter can be used to display a given XHTML fragment after the header section in the topic 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.logo_and_title

This parameter can be used to display a given XHTML fragment after the logo and title in all types of pages. 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.content.area

This parameter can be used to display a given XHTML fragment after the main content section in all types of pages. 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.content.area.main.page

This parameter can be used to display a given XHTML fragment after the main content section 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.main.content.area.topic.page

This parameter can be used to display a given XHTML fragment after the main content section in the topic 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.main.page.search (deprecated)

This parameter is deprecated. Use `webhelp.fragment.after.search.input.main.page` instead.

webhelp.fragment.after.publication.toc

This parameter can be used to display a given XHTML fragment before the publication's table of contents component in the topic 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.search.input

This parameter can be used to display a given XHTML fragment after the search field in all types of pages. 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.search.input.main.page

This parameter can be used to display a given XHTML fragment after the search field in all 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.search.input.search.page

This parameter can be used to display a given XHTML fragment after the search field in all the search results 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.search.input.terms.page

This parameter can be used to display a given XHTML fragment after the search field in all the index terms 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.search.input.topic.page

This parameter can be used to display a given XHTML fragment after the search field in all the topic 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.toc_or_tiles

This parameter can be used to display 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

This parameter can be used to display a given XHTML fragment after the top menu in all types of pages. 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.topic.breadcrumb

This parameter can be used to display a given XHTML fragment after the breadcrumb component in the topic 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.topic.content

This parameter can be used to display a given XHTML fragment after the topic's content in the topic 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.topic.toc

This parameter can be used to display a given XHTML fragment after the topic's table of contents component in the topic 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.topic.toolbar

This parameter can be used to display a given XHTML fragment after the toolbar buttons above the topic content in the topic 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.body

This parameter can be used to display a given XHTML fragment before the page body in all types of pages. 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.main.page

This parameter can be used to display a given XHTML fragment before the page body 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.body.search.page

This parameter can be used to display a given XHTML fragment before the page body in the search results 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.body.terms.page

This parameter can be used to display a given XHTML fragment before the page body in the index terms 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.body.topic.page

This parameter can be used to display a given XHTML fragment before the page body in the topic 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.feedback

This parameter can be used to display a given XHTML fragment before the **Oxygen Feedback** commenting component in the topic 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.logo_and_title

This parameter can be used to display 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.content.area

This parameter can be used to display a given XHTML fragment before the main content section in all types of pages. 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.content.area.main.page

This parameter can be used to display a given XHTML fragment before the main content section 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.main.content.area.search.page

This parameter can be used to display a given XHTML fragment before the main content section in the search results 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.main.content.area.terms.page

This parameter can be used to display a given XHTML fragment before the main content section in the index terms 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.main.content.area.topic.page

This parameter can be used to display a given XHTML fragment before the main content section in the topic 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.main.page.search (deprecated)

This parameter is deprecated. Use `webhelp.fragment.before.search.input.main.page` instead.

webhelp.fragment.before.publication.toc

This parameter can be used to display a given XHTML fragment before the publication's table of contents component in the topic 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.search.input

This parameter can be used to display a given XHTML fragment before the search field in all types of pages. 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.search.input.main.page

This parameter can be used to display a given XHTML fragment before the search field 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.search.input.search.page

This parameter can be used to display a given XHTML fragment before the search field in the search results 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.search.input.terms.page

This parameter can be used to display a given XHTML fragment before the search field in the index terms 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.search.input.topic.page

This parameter can be used to display a given XHTML fragment before the search field in the topic 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.toc_or_tiles

This parameter can be used to display 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

This parameter can be used to display a given XHTML fragment before the top menu in all types of pages. 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.topic.breadcrumb

This parameter can be used to display a given XHTML fragment before the breadcrumb component in the topic 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.topic.content

This parameter can be used to display a given XHTML fragment before the topic's content in the topic 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.topic.toc

This parameter can be used to display a given XHTML fragment before the topic's table of contents component in the topic 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.topic.toolbar

This parameter can be used to display a given XHTML fragment before the toolbar buttons above the topic content in the topic 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.custom.search.engine.results

This parameter can be used to replace the search results area with custom XHTML content. The value of the parameter is the path to an XHTML file that contains your custom content.

webhelp.fragment.custom.search.engine.script

This parameter can be used to replace WebHelp's built-in search engine with your own custom search engine. The value of the parameter is the path to an XHTML file that contains the scripts required for your custom search engine to run.

webhelp.fragment.footer

This parameter can be used to display a given XHTML fragment as the page footer in all types of pages. 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 WebHelp Responsive plugin (do not use it with a trial license).

webhelp.fragment.head

This parameter can be used to display a given XHTML fragment in the header section in all types of pages. 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.head.main.page

This parameter can be used to display a given XHTML fragment in the header section 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.head.search.page

This parameter can be used to display a given XHTML fragment in the header section in the search results 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.head.terms.page

This parameter can be used to display a given XHTML fragment in the header section in the index terms 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.head.topic.page

This parameter can be used to display a given XHTML fragment in the header section in the topic 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.welcome

This parameter can be used to display 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.enable.sticky.header

Controls whether or not the header section will remain *sticky* in the output. Possible values are **yes** (default) or **no**.

webhelp.enable.sticky.publication.toc

Controls whether or not the publication table of contents will remain *sticky* in the output.

Possible values are **yes** (default) or **no**.

webhelp.enable.sticky.topic.toc

Controls whether or not the topic table of contents will remain *sticky* in the output. Possible values are **yes** (default) or **no**.

webhelp.figure.title.placement

Controls the placement of the title for figures (relative to the image). Possible values include **top** (default) and **bottom**.

webhelp.labels.generation.mode

Controls whether or not labels are generated in the output. These labels are useful because users can easily search for topics with the same label by simply clicking on the label presented in the output. Possible values are:

- **keywords-label** - Generates labels for each defined `<keyword>` element that has the `@outputclass` attribute value set to **label**.
- **keywords** - Generates labels for each defined `<keyword>` element. If the topic contains `<keyword>` elements with the `@outputclass` attribute value set to **label**, then only these elements will have labels generated for them in the output.
- **disable** - Disables the generation of labels in the Webhelp Responsive output.

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.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**
- **right** (default)
- **top**
- **bottom**
- **hidden** - The tooltip will not be displayed.

webhelp.rellinks.group.mode

Specifies the related links grouping mode. All links can be grouped into a single "Related Information" heading or links can be grouped by their target type (topic, task, or concept). Allowed values: **single-group** (default) or **group-by-type**.

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 terms page will be displayed in the output. The default value is `yes` (meaning the index terms icon is displayed). If set to `false`, the index terms icon is not displayed in the output and the index terms page is not generated.

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.skip.main.page.generation

If set to `true`, the default main page is not generated in the output. The default value is `false`.

webhelp.table.title.placement

Controls the placement of the title for tables. Possible values include **top** (default) and **bottom**.

webhelp.top.menu.activated.on.click

When this parameter is activated (set to `yes`), clicking an item in the top menu will expand the submenu (if available). You can then click on a submenu item to open the item (topic). You can click outside the menu or press **ESC** to hide the menu. When set to `no` (default), hovering over a menu item displays the menu content.

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.enable.search.autocomplete

Specifies if the *Autocomplete* feature is enabled in the WebHelp search text field. The default value is `yes`.

webhelp.google.search.results

A file path that specifies the location of a well-formed XHTML file containing the Google Custom Search Engine element `gcse:searchresults-only`. You can use all supported attributes for this

element. It is recommended to set the `@linkTarget` attribute to `frm` for frameless (*iframe*) version of WebHelp or to `contentWin` for the frameset version of WebHelp. The default value for this attribute is `blank` and the search results will be loaded in a new window. If this parameter is not specified, the following code will be used `<gcse:searchresults-only linkTarget="frm"></gcse:searchresults-only>`.

webhelp.google.search.script

A file path that specifies the location of a well-formed XHTML file containing the Custom Search Engine script from Google.

webhelp.search.default.operator

Makes it possible to change the default operator for the search engine. Possible values are `and`, `or` (default). If set to `and` while the search query is WORD1 WORD2, the search engine only returns results for topics that contain both WORD1 and WORD2. If set to `or` and the search query is WORD1 WORD2, the search engine returns results for topics that contain either WORD1 or WORD2.

webhelp.search.enable.pagination

Specifies whether or not search results will be displayed on multiple pages. Allowed values are `yes` or `no`.

webhelp.search.index.elements.to.exclude

Specifies a list of HTML elements that will not be indexed by the search engine. The value of the `@class` attribute can be used to exclude specific HTML elements from indexing. For example, the `div.not-indexed` value will not index all `<div>` elements that have a `@class` attribute with the value of `not-indexed`. Use a comma separator to specify more than one element.

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.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.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.search.stop.words.include

Specifies a list of words that will be ignored by the search engine. Use a comma separator to specify more than one word.

webhelp.sitemap.base.url

Base URL for all the `<loc>` elements in the generated `sitemap.xml` file. If this parameter is specified, the `loc` element will contain the value of this parameter plus the relative path to the page. If this parameter is not specified, the `loc` element will only contain the relative path of the page (the relative file path from the `@href` attribute of a `<topicref>` element from the *DITA map*, appended to this base URL value).

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`.

Publishing Speedup Parameters

parallel

A common parameter with other transformation types. When set to **true** (default value is **false**), the publishing pre-processing stages are run in parallel slightly improving the publishing time.

store-type

A common parameter with other transformation types. When set to **memory**, the processing stages use internal memory to store temporarily processed documents, thus decreasing the publishing time but slightly increasing the amount of internal memory used for the process. When publishing on Windows, setting this parameter can decrease the publishing times by about one-third.



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.

Parameters for Adding a Link to PDF Documentation in WebHelp Responsive Output

The following transformation parameters can be used to generate a PDF link component in the WebHelp Responsive output (for example, it could link to the PDF equivalent of the documentation):

webhelp.pdf.link.url

Specifies the target URL for the PDF link component.

webhelp.pdf.link.text

Specifies the text for the PDF link component.

webhelp.pdf.link.icon.path

Specifies the path or URL of the image icon to be used for the PDF link component. If not specified, a default icon is used.

webhelp.pdf.link.anchor.enabled

Specifies whether or not the current topic ID should be appended as the name destination at the end of the PDF link. Allowed values are: **yes** (default) and **no**.

webhelp.show.pdf.link

Specifies whether or not the PDF link component is shown in the WebHelp Responsive output. Allowed values are: **yes** (default) and **no**.

Related information

[Generating WebHelp Responsive Output \(on page 7\)](#)

[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 an XSLT extension point. The extension point will only affect the transformations that use the particular template.



Important:

While the publishing templates only support referencing one extension point at a time, you can use `xslt:include` or `xslt:import` to aggregate multiple modules.

For a specific example of how to use an extension in a publishing template, see: [How to Use XSLT Extension Points from a Publishing Template \(on page 183\)](#).

Example:

```

<publishing-template>
  ...
  <webhelp>
    ...
    <xslt>
      <extension
        id="com.oxygenxml.webhelp.xsl.createMainPage"
        file="xsl/customMainPage.xsl" />
    </xslt>
  </webhelp>
</publishing-template>

```

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 supported:

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 212)* 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`.

com.oxygenxml.webhelp.xsl.contextHelpMap

Extension point to override the XSLT stylesheet (`contextHelpMapDita.xsl`) that generates the context sensitive help mapping. The path to this stylesheet is: `DITA-OT-DIR\plugins\com.oxygenxml.webhelp.responsive\xsl\contextHelp
\contextHelpMapDita.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 119*).

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 119*).

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 119*).

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 119*).

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 120*).

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 120*).

com.oxygenxml.webhelp.xsl.contextHelpMap.param

Use this extension point to pass parameters to the stylesheet specified using the **com.oxygenxml.webhelp.xsl.contextHelpMap** extension point (*on page 120*).

Related Information:

[DITA-OT] [XSLT-Import Extension Points](#)

[DITA-OT] [XSLT-Parameter Extension Points](#)

5.

Customizing WebHelp Responsive Output

Oxygen XML WebHelp Responsive 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.

For an in-depth look at WebHelp Responsive features and some customization tips, watch our Webinar: [DITA Publishing and Feedback with Oxygen Tools](#).

Working with Publishing Templates

An *Oxygen Publishing Template (on page 213)* 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 \(on page 71\)](#)

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 Publishing Template Starting from Scratch

To create a new *Oxygen Publishing Template (on page 213)*, follow these steps:


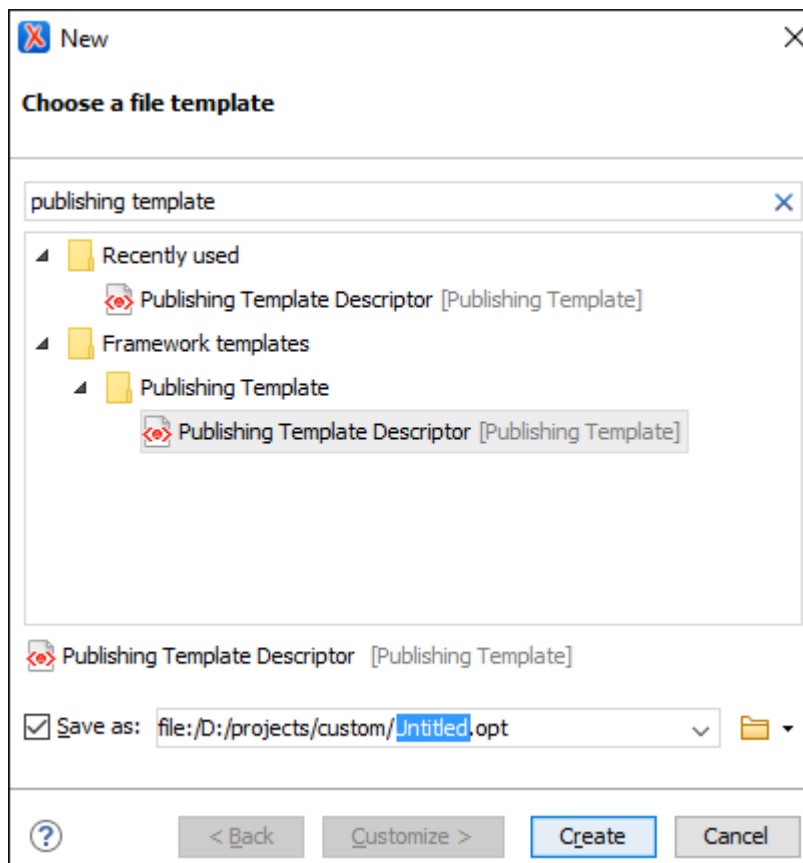
1. Create a folder that will contain all the template files.
2. In **Oxygen XML Editor/Author**, open the new document wizard (use **File > New** or the  **New** toolbar button), then choose the **Publishing Template Descriptor** template.

Figure 17. 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. See: [Publishing Template Package Contents for WebHelp Responsive Customizations \(on page 71\)](#).

Creating a Publishing Template Starting from an Existing Template

If you are using a **DITA Map WebHelp Responsive** or **DITA Map PDF - based on HTML5 & CSS** transformation, the easiest way to create a new *Oxygen Publishing Template (on page 213)* 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 87)* 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. See: [Publishing Template Package Contents for WebHelp Responsive Customizations \(on page 71\)](#).

For more information about creating and customizing publishing templates, watch our video demonstration:

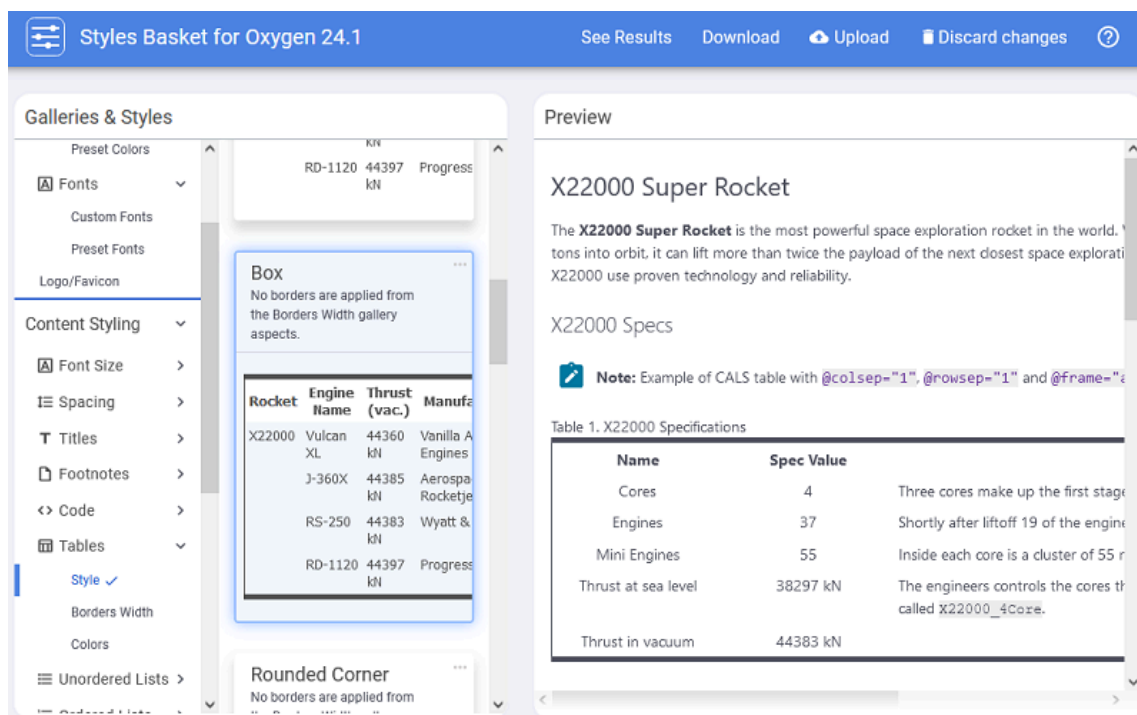
<https://www.youtube.com/embed/zNmXfKWwO8>

Creating a Publishing Template Using the Oxygen Styles Basket

Another way to create an *Oxygen Publishing Template (on page 213)* is to use the **Oxygen Styles Basket**. This tool is a handy free-to-use web-based visual tool that helps you create your own *Publishing Template Package* to customize your **DITA Map WebHelp Responsive** transformation scenarios.

It is based on galleries that you can visit to pick styling aspects to create a custom look and feel. Various different types of styles can be selected (such as fonts, tables, lists, spacing, code) and all changes can be seen in the **Preview** pane. You can also click the **See Results** button to generate a preview of either WebHelp or PDF output.

It is possible to **Download** the current template or **Upload** a previously generated template for further customization.

Figure 18. Oxygen Styles Basket Interface

Resources

For more information about the **Oxygen Styles Basket**, see the following resources:

- **Video: Introducing the New Oxygen Styles Basket**
- **Webinar: Using Oxygen Styles Basket to Create CSS Customization from Scratch**

Related information

[Publishing Template Package Contents for WebHelp Responsive Customizations \(on page 71\)](#)

How to Edit a Packed Publishing Template

To edit an existing *Oxygen Publishing Template (on page 213)* 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 **Project** view.
3. Using the **Project** 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.

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 template descriptor file](#) or [multiple descriptor files](#) ([on page 72](#)). 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 104](#)). 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/macOS:

```
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 104) and `webhelp.publishing.template.descriptor` (on page 104) parameters.

The `webhelp.publishing.template` (on page 104) parameter specifies the path to the ZIP archive (or root folder) while the `webhelp.publishing.template.descriptor` (on page 104) 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/macOS:**

```
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.

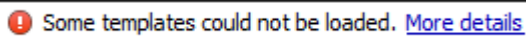
How to Share a Publishing Template


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 \(on page 72\)](#) 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:



 Some templates could not be loaded. [More details](#)

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 \(on page 72\)](#).

Also, if a template could be loaded, but certain elements could not be found in the [descriptor file \(on page 72\)](#), 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 \(on page 74\)](#) cannot be found.

Converting Old Templates to Newer Versions

WebHelp templates that were created in older versions of Oxygen XML WebHelp Responsive plugin can be converted to the Publishing Template format that was introduced in Oxygen XML WebHelp Responsive plugin version 20.0. This section contains several procedures for converting old templates depending on the version they were created in.

Convert Version 24.1 Publishing Templates to Version 25

If you have a custom Publishing Template that was created in Oxygen XML WebHelp Responsive plugin version 24.1, the following conversion procedure is required for the template to be compatible with Oxygen XML WebHelp Responsive plugin version 25.0:

1. In the **Project** view, add the root directory for your custom Publishing Template (you can use a linked folder 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 v25**, 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 25.0.

Related information

[Convert Version 23 Publishing Templates to Version 24 \(on page 129\)](#)

[Convert Version 22 Publishing Templates to Version 23 \(on page 130\)](#)

[Convert Version 21 Publishing Templates to Version 22 \(on page 130\)](#)

[Convert Version 20 Publishing Templates to Version 21 \(on page 131\)](#)

Convert Version 24.0 Publishing Templates to Version 24.1

If you have a custom Publishing Template that was created in Oxygen XML WebHelp Responsive plugin version 24.0, the following conversion procedure is required for the template to be compatible with Oxygen XML WebHelp Responsive plugin version 24.1:

1. In the **Project** view, add the root directory for your custom Publishing Template (you can use a linked folder 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 v24.1**, 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 24.1.

Related information

[Convert Version 23 Publishing Templates to Version 24 \(on page 129\)](#)

[Convert Version 22 Publishing Templates to Version 23 \(on page 130\)](#)

[Convert Version 21 Publishing Templates to Version 22 \(on page 130\)](#)

[Convert Version 20 Publishing Templates to Version 21 \(on page 131\)](#)

Convert Version 23 Publishing Templates to Version 24

If you have a custom Publishing Template that was created in Oxygen XML WebHelp Responsive plugin version 23.0 or 23.1, the following conversion procedure is required for the template to be compatible with Oxygen XML WebHelp Responsive plugin version 24:

1. In the **Project** view, add the root directory for your custom Publishing Template (you can use a linked folder 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 v24**, 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 24.

Related information

[Convert Version 24.0 Publishing Templates to Version 24.1 \(on page 129\)](#)

[Convert Version 22 Publishing Templates to Version 23 \(on page 130\)](#)

[Convert Version 21 Publishing Templates to Version 22 \(on page 130\)](#)

[Convert Version 20 Publishing Templates to Version 21 \(on page 131\)](#)

Convert Version 22 Publishing Templates to Version 23

If you have a custom Publishing Template that was created in Oxygen XML WebHelp Responsive plugin version 22.0 or 22.1, it is not necessary to convert it to version 23 because there were no structural changes made for the [HTML layout files \(on page 87\)](#) between the two versions.

Related information

[Convert Version 24.0 Publishing Templates to Version 24.1 \(on page 129\)](#)

[Convert Version 23 Publishing Templates to Version 24 \(on page 129\)](#)

[Convert Version 21 Publishing Templates to Version 22 \(on page 130\)](#)

[Convert Version 20 Publishing Templates to Version 21 \(on page 131\)](#)

Convert Version 21 Publishing Templates to Version 22

If you have a custom Publishing Template that was created in Oxygen XML WebHelp Responsive plugin version 21.0 or 21.1, the following conversion procedure is required for the template to be compatible with Oxygen XML WebHelp Responsive plugin version 22:

1. In the **Project** view, add the root directory for your custom Publishing Template (you can use a linked folder 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 24.0 Publishing Templates to Version 24.1 \(on page 129\)](#)

[Convert Version 23 Publishing Templates to Version 24 \(on page 129\)](#)

[Convert Version 22 Publishing Templates to Version 23 \(on page 130\)](#)

[Convert Version 20 Publishing Templates to Version 21 \(on page 131\)](#)

Convert Version 20 Publishing Templates to Version 21

If you have a custom Publishing Template that was created in Oxygen XML WebHelp Responsive plugin version 20.0 or 20.1, the following conversion procedure is required for the template to be compatible with Oxygen XML WebHelp Responsive plugin version 21.0 or 21.1:

1. In the **Project** view, add the root directory for your custom Publishing Template (you can use a linked folder 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. [Convert Version 20 Publishing Templates to Version 21 \(on page 131\)](#)
4. In the **XML Refactoring** dialog box, scroll to the **Publishing Template** section and select **Migrate HTML Page Layout Files to v21**, then click **Next**.
5. The **Scope** should be left as **Selected project resources**.

6. 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.
7. 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 24.0 Publishing Templates to Version 24.1 \(on page 129\)](#)

[Convert Version 23 Publishing Templates to Version 24 \(on page 129\)](#)

[Convert Version 22 Publishing Templates to Version 23 \(on page 130\)](#)

[Convert Version 21 Publishing Templates to Version 22 \(on page 130\)](#)

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 213)* 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 macOS, 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 changes 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 133) or the `args.css` parameter (on page 134).



Fastpath:

Regenerating the output to see the changes made in the CSS is not required. Instead, you can directly edit the files in *WebHelp Output Directory/oxygen-webhelp/template* and reload the page in your browser. Once you obtained the desired output, simply copy the stylesheet back to your publishing template folder.

Referencing the CSS Using a Publishing Template

1. If you have not already created a Publishing Template, see *How to Create a Publishing Template* (on page 72).
2. Using the **Project** 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 72) 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 Custom HTML Content

You can add custom HTML content in the WebHelp Responsive output by inserting it in a well-formed XML file (or specifying it in a **well-formed** XHTML fragment) that will be referenced in the transformation (either from an *Oxygen Publishing Template (on page 213)* or using one of the [HTML fragment placeholder parameters \(on page 105\)](#)). 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 content of the file is not *XML Well-formed*, the transformation will automatically convert non-well-formed HTML content to a well-formed XML equivalent (assuming the `webhelp.enable.html.fragments.cleanup` transformation parameter is set to **true**).

For example, if the HTML content includes several `<script>` or `<link>` elements, the XML fragment would have multiple root elements and to make it well-formed, it would be wrapped 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.



Note:

The converted fragments are stored in a file located in the `whr-html-fragments` subfolder of the transformation's temporary directory.

**Tip:**

If you do not want the transformation to automatically convert non-well-formed content into well-formed XML content, you can set the `webhelp.enable.html.fragments.cleanup` transformation parameter to **false**. This will instead cause the transformation to fail if at least one HTML fragment is not well-formed.

- **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 \(on page 68\)](#), 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 \(on page 192\)](#).

- **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 (e.g. &, <) that break the well-formedness of the XML fragment, it is important to place the content inside an XML comment.

Otherwise, the WebHelp transformation automatically wraps inline JavaScript or CSS content in an XML comment. Also, if the commented content contains constructs that are not allowed in an XML comment, those constructs are escaped.

[Important] XML comment tags (both the start and end tags) must be on lines by themselves. If they are on the same line as any of the script's content, it will likely result in a JavaScript error.

```
<script type="text/javascript">
  <!--
    /* Include JavaScript code here. */

    function myFunction() {
      return true;
    }
  -->
</script>
```

Using WebHelp Macros

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">
  var outDirPath = '<whc:macro value="{path(oxygen-webhelp-output-dir)}"'
  xmlns:whc="http://www.oxygenxml.com/webhelp/components"/>' ;
```



```
console.log("The output directory path is:", outDirPath);
</script>
```

**Note:**

When using the `<whc:macro>` element, you should also include the `xmlns:whc="http://www.oxygenxml.com/webhelp/components"` namespace declaration for the `whc` prefix. This is necessary for the XML fragment to be well-formed.

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 \(on page 122\)](#).
2. Insert the HTML content in a file that is XML well-formed (for example, `custom-html.xml`).
3. Using the **Project** view, copy your custom XML file 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 \(on page 72\)](#) 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.xml"
    placeholder="webhelp.fragment.head"/>
  </fragment>
</html-fragments>
</webhelp>
</publishing-template>
```

**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 \(on page 78\)](#).

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 \(on page 78\)](#).

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 Placeholders \(on page 78\)](#)

[Publishing Template Package Contents for WebHelp Responsive Customizations \(on page 71\)](#)

How to Change Numbering Styles for Ordered Lists

Ordered lists (``) 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 140) or the `args.css` parameter (on page 141).

Referencing the Custom CSS from a Publishing Template

1. If you have not already created a Publishing Template, see [How to Create a Publishing Template](#) (on page 72).
2. Using the **Project** 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 72) 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 Add Syntax Highlights for Codeblocks in the Output

Syntax Highlighting makes it easier to read the semantics of the structured content by displaying each type of code (language) in different colors and fonts. The application provides the ability to add syntax highlights in codeblocks for DITA to PDF or HTML-based output through the use of the `@outputclass` attribute and a variety of predefined values are available.

To provide syntax highlighting in the codeblocks that appear in the output, add the `@outputclass` attribute on the `<codeblock>` element and set its value to one of the predefined language values. The **Content Completion Assistant** offers a list of the possible values when adding the `@outputclass` attribute in **Text** mode but there are also two simple ways to set the value in **Author** mode:

- Select the `<codeblock>` element in the editor and in the **Attributes** view, click on the **Value** cell for the `@outputclass` attribute and select one of the predefined values (for example, `language-xml`).
- Select the `<codeblock>` element in the editor and use the **Alt + Enter** keyboard shortcut to open the in-place attributes editor window. Then select one of the predefined values from the **Value** drop-down menu.

The predefined values that can be selected are:

- language-json
- language-yaml
- language-xml
- language-bourne
- language-c
- language-cmd
- language-cpp
- language-csharp
- language-css
- language-dtd
- language-ini
- language-java

- language-javascript
- language-lua
- language-perl
- language-powershell
- language-php
- language-python
- language-ruby
- language-sql
- language-xquery

**Attention:**

It is recommended that you do not add inline elements in the codeblocks when using this `@outputclass` attribute, as it may lead to improper highlighting.

**Tip:**

Starting with version 24.0, the language values can also be set without using the `language-` prefix.

Example:

The following codeblock with the `@outputclass` set as `language-css`:

```
<codeblock outputclass="language-css" id="codeblock_1">@page preface-page {
  background-color:silver;
  @top-center{
    content: "Custom Preface Header";
  }
}
*[class ~= "topic/topic"][@topicrefclass ~= "bookmap/preface"] {
  page: preface-page;
}</codeblock>
```

would like this in WebHelp output:

```
@page preface-page {
  background-color:silver;
  @top-center{
    content: "Custom Preface Header";
  }
}
*[class ~= "topic/topic"][@topicrefclass ~= "bookmap/preface"] {
  page: preface-page;
}
```

How to Show or Hide Navigation Links in Topic Pages

The [topic pages \(on page 20\)](#) 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 \(on page 212\)](#) 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 Editor/Author**.
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 133) or the `args.css` parameter (on page 134).

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 68), follow this procedure:

1. If you have not already created a Publishing Template, see *How to Create a Publishing Template* (on page 68).
2. Open the *template descriptor file* (on page 72) 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 \(on page 78\)](#) 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 \(on page 68\)](#), follow this procedure:

1. If you have not already created a Publishing Template, see [How to Create a Publishing Template \(on page 68\)](#).
2. Open the [template descriptor file \(on page 72\)](#) 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

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 \(on page 78\)](#) 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 \(on page 68\)](#), follow this procedure:

1. If you have not already created a Publishing Template, see [How to Create a Publishing Template \(on page 68\)](#).
2. Open the [template descriptor file \(on page 72\)](#) associated with your publishing template and add the `webhelp.fragment.footer` parameter in the *html-fragments* section with its value set to a path of a file that contains well-formed XHTML content.

```
<publishing-template>
...
<webhelp>
...
<html-fragments>
  <fragment file="html/footer.xhtml" placeholder="webhelp.fragment.footer" />
</html-fragments>
</webhelp>
```

**Important:**

This parameter should only be used if you are using a valid, purchased license of Oxygen XML WebHelp Responsive 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

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 212). 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 Editor/Author**.
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 \(on page 133\)](#) or the `args.css` parameter [\(on page 134\)](#).

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 Editor/Author**.
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 133) or the `args.css` parameter (on page 134).

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 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 68), follow this procedure:

1. If you have not already created a Publishing Template, see [How to Create a Publishing Template \(on page 71\)](#).
2. Open the [template descriptor file \(on page 72\)](#) associated with your publishing template and add the `<logo>` element in the `<resources>` section and set the `@file` attribute value 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 `@target-url` attribute.

```

<publishing-template>
...
<webhelp>
...
<resources>
  <logo
    file="images/logo.png"
    target-url="http://www.example.com"
    alt="Alternate text for the logo image"/>
  </resources>
</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 68)*, follow this procedure:

1. If you have not already created a Publishing Template, see *Working with Publishing Templates (on page 122)*.
2. Open the *template descriptor file (on page 72)* associated with your publishing template and add the `<favicon>` element in the *resources* section. The path to the image is relative to the template root folder.

```
<publishing-template>
...
<webhelp>
...
<resources>
...
<favicon file="images/favicon.png" />
```

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 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>
```


- 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>
```

- 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, the majority of modern browsers have native support to render **MathML** equations embedded in the **HTML** code. If your browser that does not have support for MathML, **MathJax** is a solution to properly view MathML equations embedded in **HTML** content in a variety of browsers.

If you have DITA 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" id="MathJax-script" async
  src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.0.0/es5/latest?tex-mml-cthtml.js">
</script>
```

Result: The equation should now be properly rendered in the WebHelp output for other browsers.

Related information

[How to Insert Custom HTML Content \(on page 134\)](#)

[Getting Started with MathJax Components](#)

Searching the Output

This section contains topics that explain how to use some of the search features in WebHelp Responsive output.

Built-in JS Based Search Engine Customizations

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 following values 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

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.js`. By default, its value is **yes**, which means the *Kuromoji* analyzer is enabled by default. To hide the warning message, set the value of that parameter to **no** using either



of the methods listed below. When it 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 68)*, follow this procedure:

1. If you have not already created a Publishing Template, see *How to Create a Publishing Template (on page)*.
2. Open the *template descriptor file (on page 72)* 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 212)* 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.

How to Implement a Custom Search Filter

It is possible to implement a custom search filter (search input component) in your WebHelp Responsive output. The search input component is where users enter search queries to locate certain content within the WebHelp output.

To integrate a custom search filter, follow these steps:

1. If you have not already created a Publishing Template, see [How to Create a Publishing Template \(on page 154\)](#).
2. Create the following items in the folder that contains your publishing descriptor file (the `.opt` file):
 - A folder named `js`.
 - A folder named `fragments`.
3. In the `js` folder, create a file named `search-filter.js`.
4. As a starting point, you can copy the following content to the `search-filter.js` file:

```
/**
 * Object that implements the methods required by WebHelp to run a search filter.
 */
function CustomSearchFilter() {

    /**
     * Method required to run the search filter in webhelp. It is called when the users
     * executes the query in the search page.
     *
     * @param {WebHelpAPI.SearchResult} searchResult The search result for the executed
     query.
     *
     * @return A list of WebHelpAPI.SearchResult objects
     */
    this.filterResults = function (searchResult) {
        // implement filter
        return filteredResults;
    }
}

// Set the Search Filter to WebHelp
WebHelpAPI.setCustomSearchFilter(new CustomSearchFilter());
...
```



Note:

See the [API Search Objects section \(on page 166\)](#) for details on how to create a `WebHelpAPI.SearchResult` object.

5. Implement your custom search filter.
6. In the **fragments** folder, create a file named **search-filter-script-fragment.xml**.
7. In the **search-filter-script-fragment.xml** file, define the scripts that are required for your custom search filter to run. For example:

```
<div>
  <script src="{oxygen-webhelp-template-dir}/js/search-filter.js"></script>
</div>
```

8. Copy the **js** folder to the output folder during the transformation process. For this, open the **.opt** file and add the following content in the **<resources>** section (see [Template Resources \(on page 75\)](#) for more details):

```
<fileset>
  <include name="js/**"/>
</fileset>
```

9. Set the transformation parameters needed to enable the custom search filter. For this, open the **.opt** file and add the following content inside the **<webhelp>** element:

```
<html-fragments>
  <fragment file="fragments/search-filter-script-fragment.xml"
    placeholder="webhelp.fragment.head.search.page"/>
</html-fragments>
```

10. Run the transformation with this publishing template selected.

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 \(on page 212\)](#) 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 76) 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.

Oxygen Feedback Search Engine

How to Configure Faceted Search in WebHelp Output

A *faceted search* is a powerful tool that allows users to refine search results by selecting filters or facets. *Facets* are predefined categories that are associated with search results. By selecting one or more facets, users can narrow down their search results to a specific category or set of categories.

Configure Oxygen Feedback as an External Search Engine

To enable faceted searches, you need to have a search engine that supports this functionality. The **Oxygen Feedback** search engine implements faceted searches and can be easily configured as a search engine for WebHelp, see [Adding Oxygen Feedback to WebHelp Responsive Documentation \(on page 63\)](#) for more details.



Attention:

The default search engine that comes embedded in the WebHelp Responsive output does not support faceted searches.

Defining Facets Using a DITA Subject Scheme Map

A **subject scheme map** can be used to define controlled values and subject definitions. *Subject definitions* are classifications and sub-classifications that compose a tree. Subject definitions provide semantics that can be used in conjunction with taxonomies and ontologies.

The `<subjectdef>` element is used to define both a subject category and a list of controlled values. The parent `<subjectdef>` element defines the category, and the children `<subjectdef>` elements define the controlled values.

The following example defines the "Operating system" category, with "Linux" and "Windows" sub-categories. The controlled values (facet values) are: "RedHat Linux", "SUSE Linux", "Windows 7", and "Windows 10".

```
<subjectScheme>
  ...
  <hasInstance>
    <subjectdef keys="os" navtitle="Operating system">
      <subjectdef keys="linux" navtitle="Linux">
        <subjectdef keys="redhat" navtitle="RedHat Linux"/>
        <subjectdef keys="suse" navtitle="SUSE Linux"/>
      </subjectdef>
      <subjectdef keys="windows" navtitle="Windows">
        <subjectdef keys="win7" navtitle="Windows 7"/>
        <subjectdef keys="win10" navtitle="Windows 10"/>
      </subjectdef>
    </subjectdef>
  </hasInstance>
  ...
</subjectScheme>
```

Associating Faceted Values With a Topic Using a DITA Classification Map

The **classification domain** provides elements that enable map authors to indicate information about the subject matter of DITA topics. The subjects are defined in subject scheme maps, and the subjects are referenced using the `@keyref` attribute.

The following example shows you how to associate a faceted value with a topic:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Classification Map//EN" "classifyMap.dtd">
<map>
  <title>Classification map</title>
  <topicref keyref="how-to-install-on-suse.dita">
    <topicsubject keyref="linux">
      <subjectref keyref="suse"/>
    </topicsubject>
  </topicref>
```

```
<topicref keyref="how-to-install-win7.dita">
  <topicsubject keyref="windows">
    <subjectref keyref="win7" />
  </topicsubject>
</topicref>
</map>
```

**Note:**

The facet information cascades into child `<topicref>` elements.

Refining the Search Results by Using Facets in the Search Page

The configured facets are displayed in the search page, allowing you to narrow down the results.

When a user selects a facet, the search results are updated to only include the topics that match the selected facets. If multiple facet values are selected from the same category/facet, the search results display all topics with at least one facet. On the other hand, if multiple facet values from distinct facets are selected, the search results display all topics with all selected facet values.

Related information

[Adding Oxygen Feedback to WebHelp Responsive Documentation \(on page 63\)](#)

How to Add Searchable Labels in WebHelp Output

It is possible to add *searchable labels* in WebHelp Responsive output that can be clicked to search for topics with that exact same label. Labels are textual words attached to a DITA topic that enables it to be easily found using the search function. These labels can help you organize your topics, making it more accessible to retrieve topics for a specific text.

Configure Oxygen Feedback as an External Search Engine

To enable *searchable labels*, you need to have a search engine that supports this functionality. The **Oxygen Feedback** search engine implements *searchable labels* and can be easily configured as a search engine for WebHelp. See [Adding Oxygen Feedback to WebHelp Responsive Documentation \(on page 63\)](#) for more details.

**Attention:**

The default search engine that comes embedded in the WebHelp Responsive output does not support searchable labels. It will simply perform a standard search using the content within the label.

How to Add Searchable Labels in a DITA Topic

The generation of *searchable labels* in the WebHelp Responsive output is activated by default. You need to insert the desired text (to be displayed in the label in the output) in a `<keyword>` element with an `@outputclass` attribute set to **label** within the prolog of each topic that you want to have that label displayed in the output.



Note:

You can right-click anywhere within the topic in **Author** mode and select **Insert > Insert Label** to quickly insert the needed structure in the prolog.

For example:

```
<prolog>
  <metadata>
    <keywords>
      <keyword outputclass="label">Customization</keyword>
    </keywords>
  </metadata>
</prolog>
```

This would add a label that contains the text "Customization" in the output for the particular topic. If the user clicks that label, the search engine will search for all topics that have this same label defined.

Transformation Parameters for Generating Searchable Labels

You can have more control over how the labels are generated in the WebHelp Responsive output by using the **webhelp.labels.generation.mode** transformation parameter. The possible values for this parameter are:

- **keywords-label** - Generates labels for each defined `<keyword>` element that has the `@outputclass` attribute value set to **label**.
- **keywords** - Generates labels for each defined `<keyword>` element. If the topic contains `<keyword>` elements with the `@outputclass` attribute value set to **label**, then only these elements will have labels generated for them in the output.
- **disable** - Disables the generation of labels in the WebHelp Responsive output.



Note:

The default value for the **webhelp.labels.generation.mode** transformation parameter is **keywords-label**.

Searchable Labels in WebHelp Responsive Output

The WebHelp Responsive transformation will generate a component that renders the text value of the `<keyword>` element. When the user clicks that component, they will be redirected to the search page with the search query populated for them and the search engine will display all topics that have the same text value defined in the prolog.

Custom Search Engine

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 68)*, 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 \(on page 72\)](#) 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.

Replacing the Search Engine Only

It is possible to replace the internal search engine that is used by **Oxygen XML WebHelp** by using a custom JavaScript file. This customization method allows you to replace the search engine but keep the search results presentation.

To replace WebHelp's internal search engine, follow this procedure:

1. If you have not already created a Publishing Template, see [How to Create a Publishing Template \(on page \)](#).
2. Create the following items in the folder that contains your publishing descriptor file (the `.opt` file):
 - A folder named `js`.
 - A folder named `fragments`.
3. In the `js` folder, create a file named `search-engine.js`.
4. As a starting point, you can copy the following content to the `search-engine.js` file:

```

/**
 * Object that implements the methods required by WebHelp to run a search engine.
 */
function CustomSearchEngine() {

```

```

/**
 * Method required to run the search engine in webhelp. Handler when the users
 * executes the query in the search page.
 *
 * @param {String} query          The search input string from the user.
 * @param {Function} successHandler Needs to be called if the search operation is executed
 *                                successfully. The parameter needs to have the type of
 *                                WebHelpAPI.SearchResult
 * @param {Function} errorHandler Needs to be called if the search operation fails to
 *                                execute successfully. It needs to have the type
 *                                of String.
 */
this.performSearchOperation = function(query, successHandler, errorHandler) {
    // implement search engine
    // const searchRestult = externalSearchEngine(query);

    // convert the result to WebHelpApi.SearchResult
    // const formattedResult = convert(searchRestult);

    // call successHanlder with the converted result.
    // successHandler(formattedResult)
}

/**
 * Method required to run the search engine in webhelp. Handler when the
 * page is changed in the search page.
 *
 * @param {Integer} pageToShow    The page to be dispalyed.
 * @param {Integer} maxItemsPerPage The maximum # of items that can be displayed on a page.
 * @param {String} query          The search input string from the user.
 * @param {Function} successHandler Needs to be called if the search operation is executed
 *                                successfully. The parameter needs to have the type of
 *                                WebHelpAPI.SearchResult
 * @param {Function} errorHandler Needs to be called if the search operation fails to
 *                                execute successfully. It needs to have the type
 *                                of String.
 */
this.onPageChangedHandler = function(pageToShow, maxItemsPerPage, query, successHandler,
errorHandler) {
    // implement search engine
    // const searchRestult = externalSearchEngine(pageToShot, maxItemsPerPage, query);

```

```

    // convert the result to WebHelpApi.SearchResult
    // const formattedResult = convert(searchResult);

    // call successHanlder with the converted result.
    // successHandler(formattedResult)
  }
}

// Set the Search Engine to WebHelp
WebHelpAPI.setCustomSearchEngine(new CustomSearchEngine());

```

**Note:**

See the [API Search Objects section \(on page 166\)](#) for details on how to convert your custom search engine results to `WebHelpAPI.SearchResult`.

5. Implement your search engine.
6. In the `fragments` folder, create a file named `search-engine-script-fragment.xml`.
7. In the `search-engine-script-fragment.xml` file, define the scripts that are required for your search engine to run. For example:

```

<div>
  <script src="{oxygen-webhelp-template-dir}/js/search-engine.js"></script>
</div>

```

8. Copy the `js` folder to the output folder during the transformation process. For this, open the `.opt` file and add the following content in the `<resources>` section (see [Template Resources \(on page 75\)](#) for more details):

```

<fileset>
  <include name="js/**"/>
</fileset>

```

9. Set the transformation parameters needed to enable the search filter. For this, open the `.opt` file and add the following content inside the `<webhelp>` element:

```

<html-fragments>
  <fragment file="fragments/search-engine-script-fragment.xml"
    placeholder="webhelp.fragment.head.search.page"/>
</html-fragments>

```

API Search Objects

To replace the WebHelp Search Engine, you will need to convert your custom search result into WebHelp API Objects that WebHelp will use to render your search result on the search page. To convert your custom search result, you will have to create the following objects:

1. `WebHelpAPI.SearchMeta` is a JavaScript object used to hold additional information for the search result. To create such an object, the following fields are required:
 - **String: `searchEngineName`** - The name of the search engine used to retrieve the search result.
 - **Integer: `totalSearchItems`** - The total number of search items the search engine returned.
 - **Integer: `currentPage`** - The current page to display.
 - **Integer: `maxItemsPerPage`** - The maximum number of items that can be displayed on a page.
 - **Integer: `totalPages`** - The number of total pages for the search result.
 - **String: `originalSearchExpression`** - The query string the user typed in the search input field.

```
conase searchMeta = new WebHelpAPI.SearchMeta(searchEngineName, totalSearchItems, currentPage,
maxItemsPerPage, totalPages, originalSearchExpression);
```

2. `WebHelpAPI.SearchDocument` is a JavaScript object used to hold the search result for a single topic/HTML page. To create such an object, the following fields are required:
 - **String: `linkLocation`** - The URL to the topic.
 - **String: `title`** - The topic title.
 - **String: `shortDescription`** - The topic short description.

```
const searchDocument = new WebHelpAPI.SearchDocument(linkLocation, title, shortDescription);
```

3. `WebHelpAPI.SearchResult` is a JavaScript object used to display the search results in the search page. To create such an object, the following fields are required:
 - **WebHelpAPI.SearchMeta: `searchMeta`** - Contains additional information for the search result.
 - **Array[WebHelpAPI.SearchDocument]: `documents`** - An array with the matching documents (HTML pages) for the search result.

```
conase searchMeta = new WebHelpAPI.SearchMeta(searchEngineName, totalSearchItems, currentPage,
maxItemsPerPage, totalPages, originalSearchExpression);
const searchDocument = new WebHelpAPI.SearchDocument(linkLocation, title, shortDescription);
const documents = [searchDocument]; // An array with one element.
const searchResult = new WebHelpAPI.SearchResult(searchMeta, documents);
```

Replacing the Search Engine and Results Presentation

It is possible to integrate a custom search engine and replace the search results area into your WebHelp Responsive output. This is done by using the following transformation parameters:

webhelp.fragment.custom.search.engine.results

This parameter can be used to replace the search results area with custom XHTML content. The value of the parameter is the path to an XHTML file that contains your custom content.

webhelp.fragment.custom.search.engine.script

This parameter can be used to replace WebHelp's built-in search engine with your own custom search engine. The value of the parameter is the path to an XHTML file that contains the scripts required for your custom search engine to run.

To integrate a custom search engine into your WebHelp Responsive output, follow these steps:

1. If you have not already created a Publishing Template, see [How to Create a Publishing Template \(on page \)](#).
2. Create the following items in the folder that contains your publishing descriptor file (the `.opt` file):
 - A file named `custom-search-results-fragment.xml`.
 - A file named `custom-search-script-fragment.xml`.
 - A folder named `js`.
3. In the `custom-search-results-fragment.xml` file, define the HTML structure that will be used as the search results area. For example:

```
<div id="cumstom-search-results">...</div>
```

**Note:**

The custom search engine script will need to find an HTML element from the HTML structure that will be used as the search results area and write the search results inside it. In this example, it is the `<div>` element with the id `custom-search-results`.

4. In the `js` folder, create a file named `custom-search.js`.
5. As a starting point, you can copy the following content to the `custom-search.js` file:

```
document.addEventListener('DOMContentLoaded', (event) => {
  const params = new URLSearchParams(window.location.search);
  const searchQuery = params.get('searchQuery');
  // Implement your custom search engine
  // Display the search results
});
```

**Important:**

The value entered by the user in the search page will be available in the URL's query parameters in a parameter named `searchQuery`.

**Attention:**

`URLSearchParams` is not supported on all browsers (it is used as an example). A list with the supported browsers can be found [here](#). A different solution should be used if you need to support other browsers.

6. Implement your custom search engine.

**Note:**

The search results should be pushed into the `<div>` element created earlier with the id `custom-search-results`.

7. In the `custom-search-script-fragment.xml` file, define the scripts that are required for your custom search engine to run. For example:


```
<div>
  <script src="{oxygen-webhelp-template-dir}/js/custom-search.js"></script>
</div>
```

- Copy the `js` folder to the output folder during the transformation process. For this, open the `.opt` file and add the following content in the `<resources>` section (see [Template Resources \(on page 75\)](#) for more details):

```
<fileset>
  <include name="js/**"/>
</fileset>
```

- Set the transformation parameters needed to enable the custom search engine. For this, open the `.opt` file and add the following content inside the `<webhelp>` element:

```
<html-fragments>
  <fragment file="custom-search-script-fragment.xml"
    placeholder="webhelp.fragment.custom.search.engine.script"/>
  <fragment file="custom-search-results-fragment.xml"
    placeholder="webhelp.fragment.custom.search.engine.results"/>
</html-fragments>
```

- Run the transformation with this publishing template selected.


Tip:

A sample publishing template that overrides WebHelp's default search engine is available to download [here](#). You can use it as a starting point for your customization.

How to Display Custom Title in Search Results

It is possible to display a custom title for topics in the search results page. This can be achieved by adding the `<searchtitle>` element inside the particular DITA topic (or within the topic reference in the DITA map). The `<searchtitle>` element is used to specify the title that is displayed by search tools that locate the topic. This is useful when the topic has a title that makes sense in the context of a single information set, but may be too general in a list of search results. If the `<searchtitle>` is specified, then the search results page will display the contents inside the `<searchtitle>` as the topic title.

For details about the `<searchtitle>` element (including an example), see <https://docs.oasis-open.org/dita/v1.2/os/spec/langref/searchtitle.html>.

How to Trigger a Search Query When WebHelp is Loaded

You can use the `searchQuery` URL parameter to perform a search operation when WebHelp is loaded. This opens the search results page with the specified search query processed. The URL should look something like this:

```
http://localhost/webhelp/search.html?searchQuery=deploying%20feedback
```

Configuring the Search Engine Optimization

A **DITA Map WebHelp** transformation scenario produces 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 web admins 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.



Important:

If the `webhelp.sitemap.base.url` parameter is specified, the `loc` element will contain the value of this parameter plus the relative path to the page. If the `webhelp.sitemap.base.url` parameter is not specified, the `loc` element will only contain the relative path of the page.

You can also set these additional parameters:

- **webhelp.sitemap.change.frequency** - Specifies how frequently the WebHelp pages are likely to change (accepted values are: `always`, `hourly`, `daily`, `weekly`, `monthly`, `yearly`, and `never`).
- **webhelp.sitemap.priority** - Specifies the priority of each page (a value ranging from 0.0 to 1.0).

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 hh:mm:ss`.
- **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.

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

Oxygen XML WebHelp Responsive plugin comes with support for the following built-in languages: English, French, German, Japanese, and Chinese. It is possible to edit existing localization strings or add a new language.

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 \(on page 154\)](#)

[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.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 \(on page 171\)](#) 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>
```

- In the `DITA-OT-DIR/bin` directory of the `DITA-OT`, run one of the following scripts, depending on your operating system:

- Windows: `DITA-OT-DIR/bin/dita.bat --install`
- Linux/macOS: `sh DITA-OT-DIR/bin/dita --install`

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:

- Create a `com.oxygenxml.webhelp.localization` plugin directory inside the `DITA-OT-DIR/plugins/` location.
- 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.responsive"/>

  <feature extension="dita.xsl.strings" file="webhelp-extension-strings.xml"/>
</plugin>
```

- 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>
```

- 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`.
- 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>
```

- 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/plugins/org.dita.base/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. In the [DITA-OT-DIR/bin](#) directory of the [DITA-OT](#), run one of the following scripts, depending on your operating system:

- Windows: `DITA-OT-DIR/bin/dita.bat --install`
- Linux/macOS: `sh DITA-OT-DIR/bin/dita --install`

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 \(on page 212\)](#) 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 \(on page 68\)](#), 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 72) associated with your publishing template.
5. Use one of the parameters that begin with `webhelp.fragment` (on page 78) 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 78)*. 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 X™ (formerly known as 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 X™ *Tweet* widget to your WebHelp Responsive output using an *Oxygen Publishing Template (on page 68)*, 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 \(on page 72\)](#) associated with your publishing template.
5. Use **one of the parameters that begin with `webhelp.fragment` (on page 78)** 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 X™ *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` \(on page 78\)](#). 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 68), 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 72) associated with your publishing template.
6. Use the `webhelp.fragment.after.body` parameter (on page 105) 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-XXXXXXX-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 105*) 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.

Ant Extensions for WebHelp Responsive

The WebHelp Responsive plugin provides extension points that allow you to implement custom Ant targets to perform additional operations before and after certain processing stages. The following extension points are available in WebHelp Responsive:

whr-init-pre

Runs a custom Ant target before the `whr-init` processing stage.

whr-init-post

Runs a custom Ant target after the `whr-init` processing stage.

whr-collect-indexterms-pre

Runs a custom Ant target before the `whr-collect-indexterms` processing stage.

whr-collect-indexterms-post

Runs a custom Ant target after the `whr-collect-indexterms` processing stage.

whr-toc-xml-pre

Runs a custom Ant target before the `whr-toc-xml` processing stage.

whr-toc-xml-post

Runs a custom Ant target after the `whr-toc-xml` processing stage.

whr-context-help-map-pre

Runs a custom Ant target before the `whr-context-help-map` processing stage.

whr-context-help-map-post

Runs a custom Ant target after the `whr-context-help-map` processing stage.

whr-sitemap-pre

Runs a custom Ant target before the `whr-sitemap` processing stage.

whr-sitemap-post

Runs a custom Ant target after the `whr-sitemap` processing stage.

whr-copy-resources-pre

Runs a custom Ant target before the `whr-copy-resources` processing stage.

whr-copy-resources-post

Runs a custom Ant target after the `whr-copy-resources` processing stage.

whr-create-topic-pages-pre

Runs a custom Ant target before the `whr-create-topic-pages` processing stage.

whr-create-topic-pages-post

Runs a custom Ant target after the `whr-create-topic-pages` processing stage.

whr-create-main-page-pre

Runs a custom Ant target before the `whr-create-main-page` processing stage.

whr-create-main-page-post

Runs a custom Ant target after the `whr-create-main-page` processing stage.

whr-create-search-page-pre

Runs a custom Ant target before the `whr-create-search-page` processing stage.

whr-create-search-page-post

Runs a custom Ant target after the `whr-create-search-page` processing stage.

whr-create-indexterms-page-pre

Runs a custom Ant target before the `whr-create-indexterms-page` processing stage.

whr-create-indexterms-page-post

Runs a custom Ant target after the `whr-create-indexterms-page` processing stage.

whr-search-index-pre

Runs a custom Ant target before the `whr-search-index` processing stage.

whr-search-index-post

Runs a custom Ant target after the `whr-search-index` processing stage.

To use Ant extension points for WebHelp Responsive, follow these steps:

1. In the `DITA-OT-DIR/plugins/` folder, create a folder for this plugin (for example, `com.oxygenxml.webhelp.responsive.custom.ant.extensions`).
2. Create a **plugin.xml** file (in the folder you created in step 1) that extends the WebHelp Responsive plugin and specifies an Ant extension point with your custom Ant project file that contains the new build targets. For example:

```
<plugin id="com.oxygenxml.webhelp.responsive.custom.ant.extensions">
  <require plugin="com.oxygenxml.webhelp.responsive"/>
  <feature extension="ant.import" file="custom_build_file.xml"/>
</plugin>
```

3. Create the **custom_build_file.xml** file (in the folder you created in step 1) that contains your custom Ant project implementing one or more extension points:

```
<project name="custom.ant.extensions.integrator" basedir=".">
  <target name="custom-whr-init-pre" extensionOf="whr-init-pre">
    <echo>Extension point that executes before whr-init</echo>
  </target>
  <target name="custom-whr-init-post" extensionOf="whr-init-post">
    <echo>Extension point that executes after whr-init</echo>
  </target>
</project>
```

4. Integrate the plugin into the DITA-OT. In the `DITA-OT-DIR/bin` directory of the DITA Open Toolkit, run one of the following scripts, depending on your operating system:

- Windows: `DITA-OT-DIR/bin/dita.bat --install`
- Linux/macOS: `sh DITA-OT-DIR/bin/dita --install`

5. Execute a DITA Map to WebHelp Responsive transformation script.

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 \(on page 213\)](#).



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

[XSLT-Import and XSLT-Parameter Extension Points \(on page 118\)](#)

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 \(on page 68\)](#).

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:

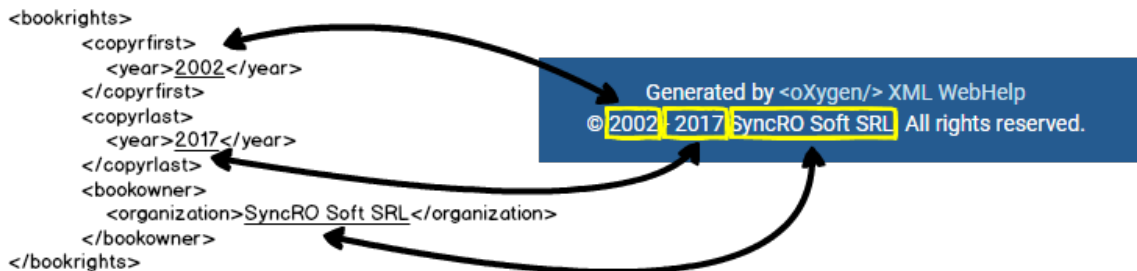
```
<bookrights>
  <copyrfirst>
    <year>2002</year>
  </copyrfirst>
  <copyrlast>
    <year>2017</year>
  </copyrlast>
  <bookowner>
    <organization>SyncRO Soft SRL</organization>
  </bookowner>
</bookrights>
```

```

</bookowner>
</bookrights>

```

Figure 19. 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 88) 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($toc/*:topicmeta/*:bookrights/*:copyfirst) and
                    exists($toc/*:topicmeta/*:bookrights/*:copylast)">
        <span class="copyright_years">
          &#xa9; <xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyfirst"/>
          - <xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copylast"/>
        </span>
      </xsl:when>

      <!-- Adds only the first year if last is not defined. -->
      <xsl:when test="exists($toc/*:topicmeta/*:bookrights/*:copyfirst)">
        <span class="copyright_years">
          &#xa9; <xsl:value-of select="$toc/*:topicmeta/*:bookrights/*:copyfirst"/>
        </span>
      </xsl:when>
    </xsl:choose>
  </div>

```



```

    </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 have not already created a Publishing Template, see [How to Create a Publishing Template \(on page \)](#).
2. Link the folder associated with the publishing template to your current project in the **Project** view.

Step Result: You should have the `custom_footer_template` folder linked in your project.

3. Using the **Project** view, create an `xslt` folder inside the project root folder.

Step Result: 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>
    </xsl:choose>
  </div>

```

```

<!-- 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 72) 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.

```

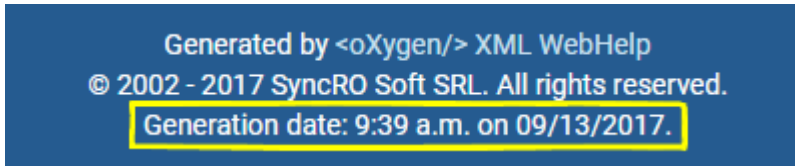
<publishing-template>
  ...
  <webhelp>
    ...
    <xslt>
      <extension
        file="xslt/customMainPage.xsl"
        id="com.oxygenxml.webhelp.xsl.createMainPage" />
    </xslt>
  </webhelp>
</publishing-template>

```

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 20. 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 72) 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" />
    </parameters>
  </webhelp>
</publishing-template>
```

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:

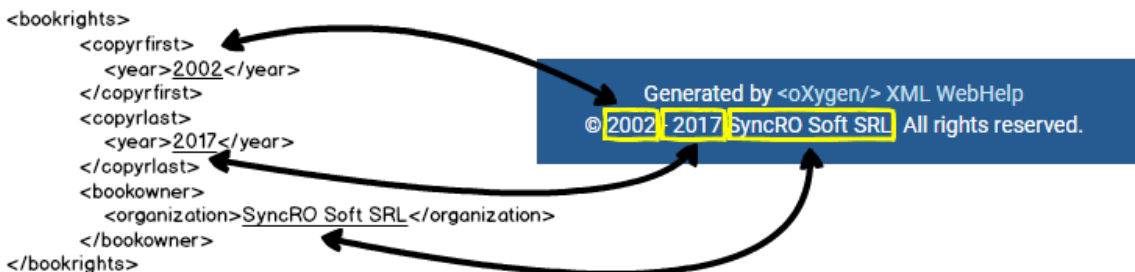
This customization is available as a GitHub project at: <https://github.com/oxygenxml/com.oxygenxml.webhelp.responsive.custom.footer>.

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 21. 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 88) 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 119). 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>

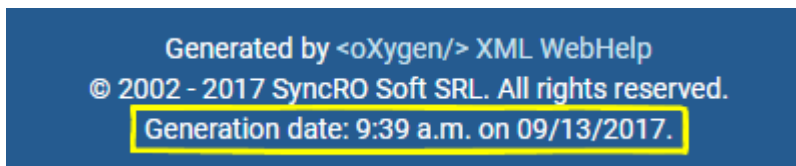
```

- In the `DITA-OT-DIR/bin` directory of the `DITA-OT`, run one of the following scripts, depending on your operating system:
 - Windows: `DITA-OT-DIR/bin/dita.bat --install`
 - Linux/macOS: `sh DITA-OT-DIR/bin/dita --install`
- Execute a DITA Map to WebHelp Responsive transformation script.

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 not this customization is active. In this case, you can use the `com.oxygenxml.webhelp.xsl.createMainPage.param` extension point ([on page 120](#)).

Figure 22. Generation Time Added in the WebHelp Footer



To add this functionality, follow these steps:

- Create a DITA-OT plugin structure by following the first 3 steps in the [procedure above \(on page 188\)](#).
- 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>

```

- 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. In the `DITA-OT-DIR/bin` directory of the `DITA-OT`, run one of the following scripts, depending on your operating system:
 - Windows: `DITA-OT-DIR/bin/dita.bat --install`
 - Linux/macOS: `sh DITA-OT-DIR/bin/dita --install`
6. Use the `webhelp.footer.add.generation.time` parameter in your DITA transformation script and specify the desired value (*yes* or *no*).
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 Copy Additional Resources to Output Directory

You can copy additional resources (such as graphics, JavaScript, CSS, entire folders, or other resources) to the output directory either by using an *Oxygen Publishing Template (on page 213)* or the `webhelp.custom.resources` parameter.

Copying Additional Resources to the Output Directory using a Publishing Template

1. If you have not already created a Publishing Template, see *How to Create a Publishing Template (on page)*.
2. Add a new `<fileset>` element in the `resources` section of the template descriptor file (*on page 75*).

```
<publishing-template>
  ...
  <webhelp>
    ...
    <resources>
      <fileset>
        <include name="custom-resources/**/*" />
        <exclude name="**/*.git" />
      </fileset>
    </resources>
  </webhelp>
</publishing-template>
```


**Note:**

Relative paths in the descriptor file are relative to the template root folder.

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 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 68)*, follow this procedure:

1. If you have not already created a Publishing Template, see *Working with Publishing Templates (on page 122)*.
2. Open the *template descriptor file (on page 72)* 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 & (e.g. `&tags-mode=no-tags`).

```

<publishing-template>
  ...
  <webhelp>
    ...
    <parameters>
      <parameter name="editlink.ditamap.edit.url"
                value="webdav-https://dav.box.com/dav/my.ditamap" />
    </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.ditamap.edit.url=webdav-https://dav.box.com/dav/my.ditamap
```

- **macOS/ Linux:**

```
dita -i /mySample.ditamap -f webhelp-responsive -Deditlink.ditamap.edit.url=webdav-https://dav.box.com/dav/my.ditamap
```

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 & (e.g.: *&tags-mode=no-tags*).
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 212)* (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 68)*, 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 72)* 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>
</publishing-template>
```

```
</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 View MathML Equations in HTML Output

By default, 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>
```

Alternate Method for DITA

For DITA documents, you can also use the following procedure:

1. Create an XML file that contains a script similar to the one shown in the example above.
2. Edit the DITA Map transformation scenario and open the **Parameters** tab.
3. Set the following parameter to point to the XML file created in step 1:
 - **WebHelp Responsive Systems** - Set the `webhelp.fragment.head` parameter to point to your XML file.
 - **WebHelp Classic Systems** - Set the `webhelp.head.script` parameter to point to your XML file.
 - **Any other type of HTML-based publishing** - Set the `args.hdf` parameter to point to your XML file.
4. Run the transformation scenario.

Result: The equation should now be properly rendered in other browsers, such as Edge, IE, or Chrome.

How to Disable Caching in WebHelp Responsive Output

In cases where a set of WebHelp Responsive 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 Responsive output, follow this procedure:

1. Create a new well-formed XML 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:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head>
    <meta http-equiv="Pragma" content="no-cache" />
    <meta http-equiv="Expires" content="-1" />
  </head>
</html>
```

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.
4. Click **OK** to save the changes to the transformation scenario.
5. Run the transformation scenario.

Result: Your additional content is included at the end of the `<head>` element of your output document.

How to Add a Link to PDF Documentation

It is possible to add a component in your WebHelp output that links to an external PDF resource. For example, it could link to the PDF equivalent of the documentation. This is achieved by configuring some transformation parameters and the link component is added in the header/breadcrumb stripe, next to the navigation links.

The transformation parameters used for generating a PDF link component in the WebHelp Responsive output are:

webhelp.pdf.link.url

Specifies the target URL for the PDF link component.

webhelp.pdf.link.text

Specifies the text for the PDF link component.

webhelp.pdf.link.icon.path

Specifies the path or URL of the image icon to be used for the PDF link component. If not specified, a default icon is used.

webhelp.show.pdf.link

Specifies whether or not the PDF link component is shown in the WebHelp Responsive output. Allowed values are: **yes** (default) and **no**.

webhelp.pdf.link.anchor.enabled

Specifies whether or not the current topic ID should be appended as the name destination at the end of the PDF link. Allowed values are: **yes** (default) and **no**.

How to Add a Custom Component for WebHelp Output

This topic explains how to use several customization methods to define and implement a custom component for WebHelp output pages.

Predefined components

The WebHelp output is based on a set of [HTML Page Layout Files \(on page 87\)](#) that define the default layout of the generated pages. Each layout file is made of a set of various components. Each component is described using an associated XML element that is processed at the generation time resulting in its associated component being included in the output pages.

Here are a few examples of predefined components: **Logo, Title, Menu, Search Input, Topics Tiles, Topic Breadcrumb, Topic Content, Publication Table of Contents**. A complete list with all the available components is available here: [Layout of the Responsive Page Types \(on page 16\)](#).

For example, the page component that is used to define the Search Input field in the WebHelp HTML pages is defined as follows:

```
<!-- Search form -->
<whc:webhelp_search_input class="navbar-form wh_topic_page_search search" role="form" />
```

At publishing time, the above component will be expanded into:

```
<div class=" wh_search_input navbar-form wh_topic_page_search search">
  <form id="searchForm" method="get" role="search" action="../search.html">
    <div>
      <input type="search" placeholder="Search "
        class="wh_search_textfield ui-autocomplete-input" id="textToSearch"
        name="searchQuery" aria-label="Search query" required="required"
        autocomplete="off" />
      <button type="submit" class="wh_search_button" aria-label="Search">
        <span class="search_input_text">Search</span>
      </button>
    </div>
  </form>
</div>
```

Customization Methods

The most common customization methods for the WebHelp Responsive output include:

- Apply custom CSS styles ([on page 132](#)) to change the default layout and styles.
- Insert additional HTML content ([on page 134](#)) using one of the available [HTML Fragment Placeholder parameters \(on page 78\)](#).
- Extend the default processing using [XSLT Extension Points \(on page 78\)](#).
- Configure available [Transformation Parameters \(on page 104\)](#).

Use Case: Custom Link Component

For the subsequent procedure, suppose you have a DITA project for a User Manual and you also have various video demonstrations available on your website that supplement the documentation. You may want to link a video demonstration for a particular feature to its associated DITA topic in the WebHelp output.

You could simply add a link somewhere in your DITA topic, but this approach would not be very suitable for a printable (PDF) version of your User Manual. Thus, you need to include the link to the associated video demonstration only in the WebHelp output of your User Manual (and not the PDF version).

One way to link a video with its associated topic is to include its URL in the metadata section. For example:

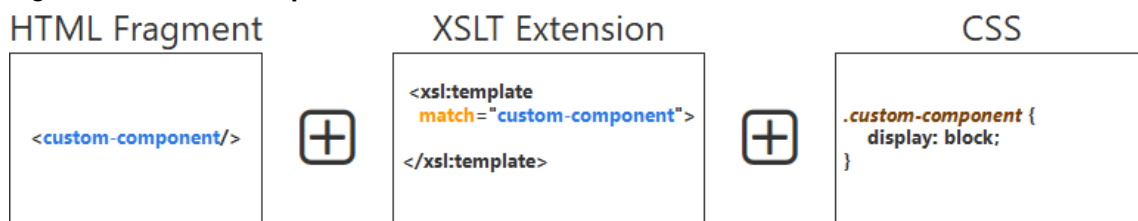
```
<prolog>
  <metadata>
    <othermeta name="video-link" content="https://www.youtube.com/watch?v=zNmXfKWXwO8" />
  </metadata>
</prolog>
```

Next, you need to instruct WebHelp to pick up the URL from the metadata and generate a link in a specific location of the HTML output page. You can achieve this by creating your own WebHelp custom component.

Creating a Custom Component

You can combine several of the available customization methods to define and implement your own WebHelp custom component.

Figure 23. Custom Component



To create a custom component that displays a link to the current topic's associated video tutorial, follow these steps:

1. Define your component. For example, it may have the following form:

```
<comp:video-link xmlns:comp="http://example.com/custom-components" />
```

The component is an XML element that belongs to a custom defined namespace.

2. Insert the component in your topic pages. To do this, you will have to save the associated XML element in an HTML Fragment file (for example, named `video-link-fragment.xml`).
3. Reference the HTML Fragment file in your current [Publishing Template's descriptor file \(on page 72\)](#) and associate it with an HTML Fragment placeholder that is available for the topic pages (`webhelp.fragment.before.topic.toolbar` in this case):

```
<html-fragments>
  <fragment file="component/html-fragment/video-link-fragment.xml"
    placeholder="webhelp.fragment.before.topic.toolbar" />
</html-fragments>
```



Note:

The HTML Fragment file is referenced using a path relative to the Publishing Template root directory.

4. Create a custom XSLT file that processes the custom component and picks up the video URL available in the current topic's metadata and generates a link to the page that contains the video:


```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:comp="http://example.com/custom-components"
  exclude-result-prefixes="xs comp"
  version="3.0">

  <!-- Custom component implementation -->
  <xsl:template match="comp:video-link" mode="copy_template">
    <xsl:param name="ditaot_topicContent" tunnel="yes"/>
    <!-- Look for a 'video-link' <meta> element in the current topic content -->
    <xsl:variable name="videoLinkMeta"
      select="$ditaot_topicContent//*:meta[@name='video-link']"/>
    <xsl:if test="exists($videoLinkMeta)">
      <div class="video-link-container">
        <a href="{ $videoLinkMeta[1]/@content }"
          class="video-link" target="_blank" aria-label="Video">
          <span>Video</span>
        </a>
      </div>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>

```

The HTML content generated for your component will look like this:

```

<div class="video-link-container">
  <a href="https://www.youtube.com/watch?v=zNmXfKWXwO8"
    class="video-link" target="_blank"
    aria-label="Video">
    <span>Video</span>
  </a>
</div>

```

5. Reference the above XSL file in your Publishing Template's descriptor file using the XSLT extension point associated with the XSL module that generates an HTML file for each DITA topic:

```

<xslt>
  <extension file="component/xsl/video-link-impl.xsl"
    id="com.oxygenxml.webhelp.xsl.dita2webhelp" />
</xslt>

```

6. Create a custom CSS file that contains the rules for styling the output for your component:

```

@import url('https://fonts.googleapis.com/icon?family=Material+Icons');

```

```

.video-link-container {
    display: flex;
    align-items: center;
    flex-grow: 10;
    justify-content: flex-end;
}

.video-link {
    display: flex;
    align-items: center;
    color: #fff !important;
}

.video-link:before {
    content: "smart_display";
    font-family: 'Material Icons';
    font-size: 20px;
    display: inline-block;
    word-wrap: normal;
    white-space: nowrap;
}

.video-link span {
    display: none;
}

.wh_right_tools {
    padding: 0;
}

```

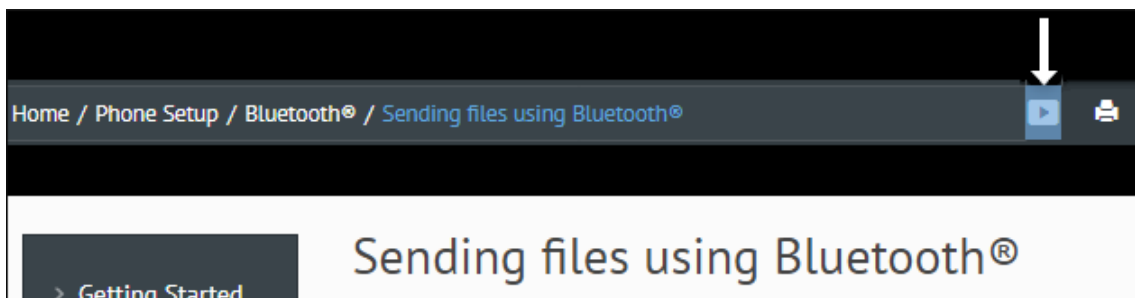
7. Reference the above CSS file in your Publishing Template's descriptor file:

```

<resources>
    <!-- .... -->
    <css file="component/css/video-link.css"/>
</resources>

```

Result: An icon that is a link to the video appears in the header stripe in the output page.

Figure 24. Custom Link to Video Component

Sample Publishing Template

A sample Publishing Template that contains all the above customizations is available here: <https://github.com/oxygenxml/oxygen-publishing-template-samples/tree/master/templates/video-link-custom-component>.

How to Generate Google Structured Data

It is possible to generate Google Structured Data (`<script>` elements that contain a JSON-LD object) in the DITA WebHelp Responsive output. Google uses this JSON-LD object to better understand the contents of the page and display special search results in a Google Search.



Tip:

For more details, see [Google Search Central: Understand how structured data works](#).

To generate Google Structured Data in WebHelp output, use the following transformation parameter:

google.structured.data

Specifies whether or not Google Structured Data will be generated in the output. If set to **yes**, the transformation automatically generates Google Structured Data for **Questions and Answers** topics, DITA Task topics, and from `<data>` elements found inside a topic that has the `@name="oxy:question"` construct. If set to **no** (default value), the transformation will not generate Google Structured Data.

Generating Google Structured Data for DITA Tasks Topics

When Google Structured Data is enabled, the DITA Task `<title>`, `<shordesc>`, and `<step>` elements are mapped to the `HowTo` JSON-LD object. For example, the following DITA Task topic:

```
<task id="task_id">
  <title>My task</title>
  <shordesc>Task description</shordesc>
  <steps>
    <step>
      <cmd>Step 1 content.</cmd>
    </step>
  </steps>
</task>
```

```

<step>
  <cmd>Step 2 content.</cmd>
</step>
</steps>
</task>

```

will generate the following structure in the output:

```

<script type="application/ld+json" id="jsonld-howto">
{
  "@context": "https://schema.org",
  "@type": "HowTo",
  "name": "My task",
  "description": "Task description",
  "supply": [],
  "tool": [],
  "step": [
    {
      "@type": "HowToStep",
      "text": "<span class=\"topic/ph task/cmd ph cmd\">Step 1 content.</span>"
    },
    {
      "@type": "HowToStep",
      "text": "<span class=\"topic/ph task/cmd ph cmd\">Step 2 content.</span>"
    }
  ]
}
</script>

```

Generating for Questions and Answers Topics

When Google Structured Data is enabled, the QA topic `<qagroup>` elements are mapped to the [FAQPage](#) JSON-LD object. For example, the following QA topic:

```

<qatopic id="qa_id">
  <title>Faq Page 1</title>
  <qabody>
    <qagroup>
      <question>What is a car engine?</question>
      <answer>The car engine is a device that uses fuel to create mechanical power that can
        turn the car's wheels.</answer>
    </qagroup>
  </qabody>
</qatopic>

```

will generate the following structure in the output:

```
<script type="application/ld+json" id="jsonld-faq">
{
  "@context": "https://schema.org",
  "@type": "FAQPage",
  "mainEntity": [
    {
      "@type": "Question",
      "name": "What is a car engine?",
      "acceptedAnswer": {
        "@type": "Answer",
        "text": "<div class=\"- topic/div qatopic/answer div answer\">The car engine is a
device that uses fuel to create mechanical power that can turn the car's wheels.</div>"
      }
    }
  ]
}
</script>
```

Generating from `data` elements found inside a topic

When Google Structured Data is enabled, the WebHelp Responsive transformation will map the `<data>` elements found inside a topic to a `FAQPage` JSON-LD object. There are 2 different use cases depending on where the `<data>` element is found in the document:

- In the `<prolog>` element. For example, this content:

```
<concept id="lawnmowerconcept">
  <title>Lawnmower</title>
  <shortdesc>The lawnmower is a machine used to cut grass in the yard.</shortdesc>
  <prolog>
    <metadata>
      <data name="oxy:question">What tools are necessary to cut the grass?</data>
    </metadata>
  </prolog>
  <conbody>
    <p>Lawnmowers can be electric, gas-powered, or manual.</p>
  </conbody>
</concept>
```

will generate the following structure in the output:

```
<script type="application/ld+json" id="jsonld-faq">
{
  "@context": "https://schema.org",
```

```

"@type": "FAQPage",
"mainEntity": [
  {
    "@type": "Question",
    "name": "What tools are necessary to cut the grass?",
    "acceptedAnswer": {
      "@type": "Answer",
      "text": "<div class=\"- topic/body concept/conbody body conbody\">
        <p class=\"- topic/shortdesc shortdesc\">The lawnmower is a machine
        used to cut grass in the yard.</p> <p class=\"- topic/p p\">Lawnmowers can be electric,
        gas-powered, or manual.</p> </div>"
    }
  }
]
}
</script>

```



Important:

The answer represents the HTML result of the entire content inside the topic.

- Inside the topic body elements. For example, content:

```

<topic id="concept-id">
  <title>Morning</title>
  <shortdesc>In the morning we have breakfast.</shortdesc>
  <body>
    <ul>
      <data name="oxy:question">What do people drink in the morning?</data>
      <li>Tea</li>
      <li>Milk</li>
    </ul>
  </body>
</topic>

```

will generate the following structure in the output:

```

<script type="application/ld+json" id="jsonld-faq">
  {
    "@context": "https://schema.org",
    "@type": "FAQPage",
    "mainEntity": [
      {
        "@type": "Question",
        "name": "What do people drink in the morning?",

```

```

    "acceptedAnswer": {
      "@type": "Answer",
      "text": "<div class=\"- topic/body body\"><ul class=\"- topic/ul ul\"></ul>
<li class=\"- topic/li li\">Tea</li> <li class=\"- topic/li li\">Milk</li> </div>"
    }
  }
]
}
</script>

```



Important:

The answer represents the HTML result of the entire block where the `<data>` element is located inside.

How to Group Related Links by Type

By default, all links from DITA relationship tables or related link elements within topics are grouped under one "Related information" heading:

```

Related information
  Target Topic
  Target Concept
  Target Task

```

It is possible to group the links by target type (topic type) by setting the `webhelp.rellinks.group.mode=group-by-type` parameter. The output will look like this:

```

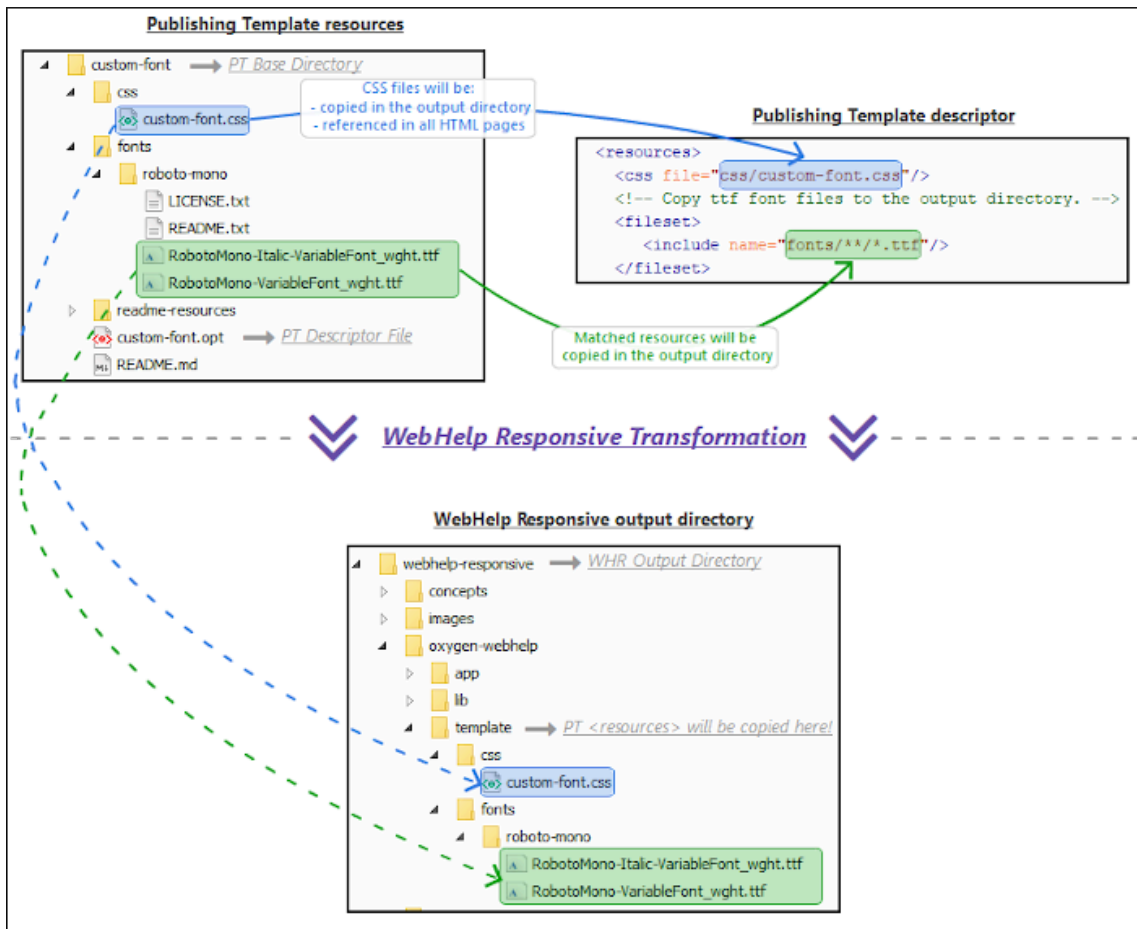
Related concepts
  Target Concept
Related tasks
  Target Task
Related information
  Target Topic

```

How to Use a Local Font in WebHelp Responsive Output

It is possible to use a local fonts in WebHelp Responsive output by copying the local font file to the output directory through a Publishing Template and referencing the font files using `@font-face` rules within a custom CSS.

Figure 25. Referencing Local Fonts in a Publishing Template



To use a local font in your WebHelp Responsive output, follow these steps:

1. If you have not already created a Publishing Template, see [How to Create a Publishing Template \(on page 198\)](#).
2. Add the local font files to the `fonts` folder within your Publishing Template directory structure. For example:

```
fonts/roboto-mono/RobotoMono-Italic-VariableFont_wght.ttf
fonts/roboto-mono/RobotoMono-VariableFont_wght.ttf
```

3. Configure WebHelp Responsive to copy the font file to the output directory. Define a `<fileset>` that matches the location of the font files in the `<resources>` section of your Publishing Template's descriptor file.

```
<resources>
  <!-- Copy ttf font files to the output directory. -->
  <fileset>
    <include name="fonts/**/*.*.ttf"/>
  </fileset>
</resources>
```

All the files matched by this fileset will be copied to the output directory. The additional resources will be copied in the following subfolder of the output directory:


```
{OUTPUT-DIR}/oxygen-webhelp/template/
```

4. Create a custom CSS file in your Publishing Template directory.

```
css/custom-font.css
```

5. Reference the CSS file in the `<resources>` section of the Publishing Template's descriptor file. This means that the CSS file will be referenced in each HTML page within the WebHelp Responsive output.

```
<resources>
  <css file="css/custom-font.css"/>
  <!-- ... -->
</resources>
```

6. Add `@font-face` definitions that reference the font files in your custom CSS file. The font files can be referenced using relative URLs since the CSS and the font files included in the Publishing Template package will be copied together in the output folder.

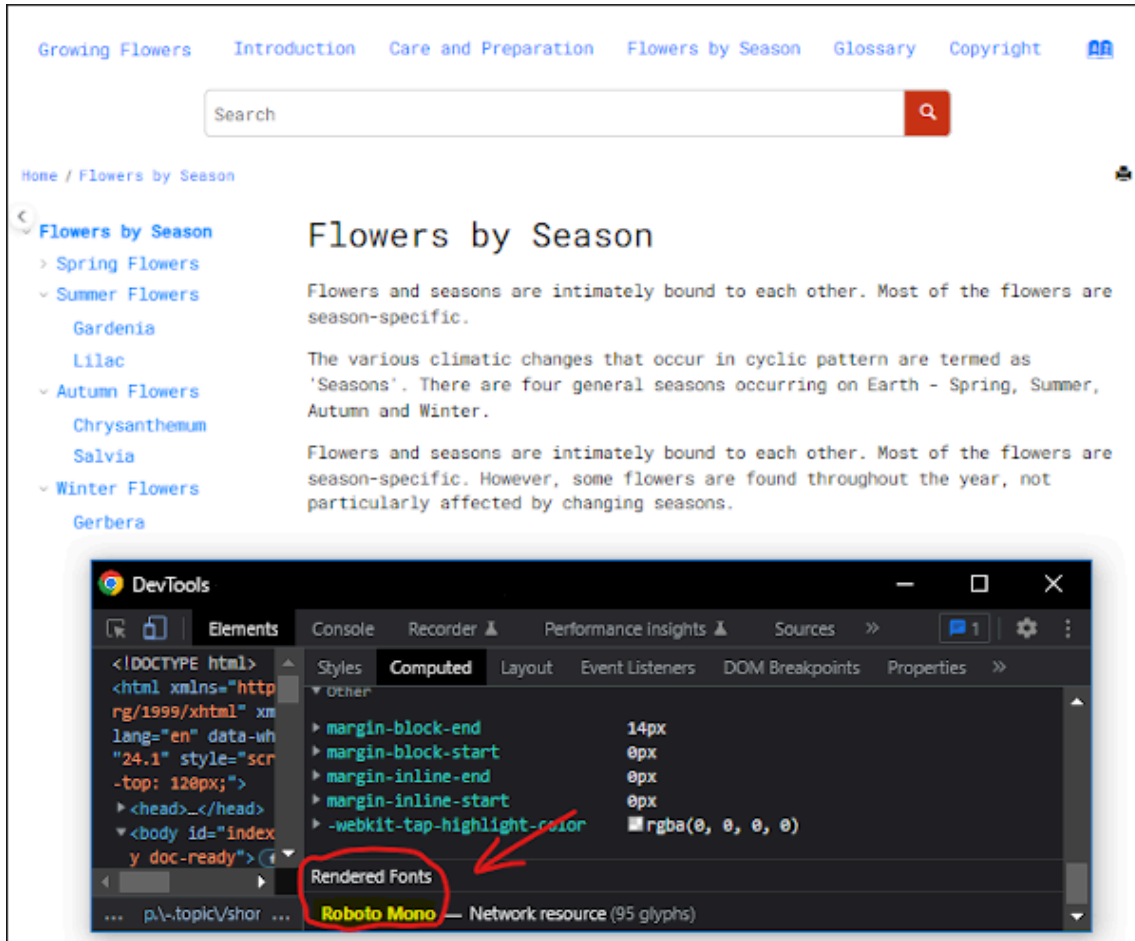
```
@font-face {
  font-family: 'Roboto Mono';
  font-style: normal;
  src: url('../fonts/roboto-mono/RobotoMono-VariableFont_wght.ttf') format('truetype');
}
@font-face {
  font-family: 'Roboto Mono';
  font-style: italic;
  src: url('../fonts/roboto-mono/RobotoMono-Italic-VariableFont_wght.ttf')
  format('truetype');
}
```

7. Add a CSS rule that applies the custom font on all elements.

```
body {
  font-family: 'Roboto Mono', sans-serif;
}
```

8. Run the transformation with the publishing template selected.

Figure 26. Output Example



How to Use JQuery in WebHelp Responsive Output

The JQuery library that comes bundled with WebHelp is accessible in the browser's global context so that developers have access to use it.

To use the JQuery library in your WebHelp Responsive output, follow these steps:

1. If you have not already created a Publishing Template, see [How to Create a Publishing Template](#).
2. Create the following items in the folder that contains your publishing template's descriptor file (the `.opt` file):
 - A folder named `js`
 - A folder named `fragments`
3. In the `js` folder, create a file named `custom.js`.
4. As a starting point, you can copy the following content to the `custom.js` file:

```
$(document).ready(function () {
    // Your JQuery code.
});
```

5. In the `fragments` folder, create a file named `jquery-scripts.html` with the following content:

```
<html>
  <script src="{oxygen-webhelp-template-dir}/js/custom.js" defer="defer"></script>
</html>
```

**Important:**

Make sure that the `@defer` attribute is present on the `<script>` element.

6. Copy the `js` folder to the output folder during the transformation process. For this, open the `.opt` file and add the following content in the `<resources>` section (see [Template Resources](#) for more details):

```
<fileset>
  ...
  <include name="js/**" />
  ...
</fileset>
```

7. Include the `jquery-scripts.html` file in your WebHelp Responsive output by opening the `.opt` file and add the following content inside the `<webhelp>` element:

```
<html-fragments>
  <fragment file="jquery-scripts.html" placeholder="webhelp.fragment.head"/>
</html-fragments>
```

8. Run the transformation with your publishing template selected.

6.

Glossary

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"`).

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.

DITA Map

A **DITA map** is a component of the DITA *framework (on page 213)* 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 212)* to generate a deliverable using an output type such as XHTML, PDF, HTML Help, or Eclipse Help.

DITA Open Toolkit

DITA Open Toolkit is an open-source publishing engine for content authored in the Darwin Information Typing Architecture. It is a vendor-independent, open-source implementation of the DITA standard, released under the [Apache License, Version 2.0](#).

The toolkit supports all versions of the [OASIS DITA specification](#), including 1.0, 1.1, 1.2, and 1.3.

DITA-OT

Related information

<http://www.dita-ot.org/>

DITA-OT-DIR

DITA_OT_DIR refers to the default directory for your DITA Open Toolkit distribution.

Framework

A **framework** refers to a package that contains resources and configuration information to provide ready-to-use support for a vocabulary or document type. A *framework* is associated to a document type according to a set of rules. It also includes a variety of settings that improve editing capabilities for its particular file type.

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.

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 213](#)) defines a set of effective key bindings.

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 to share with others. It represents the main method for customizing the *WebHelp Responsive* output.

Related Information:

[Publishing Template Package Contents for WebHelp Responsive Customizations \(on page 71\)](#)

Root Map

A **Root Map** (or main map) specifies a *DITA map* ([on page 212](#)) 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 213](#))).

WebHelp Output Directory

WebHelp_OUTPUT_DIR refers to the output directory where WebHelp transformation files will be generated.

When running the WebHelp transformation from a command line, the output directory can be specified using the `-o` or `--output` option.

Index

A

- Add link to PDF
 - 198
- Adding audio objects
 - 151
- Adding favicon
 - 150
- Adding logo
 - 149
- Adding videos
 - 151
- Adding welcome message
 - 145
- Author editing mode
 - MathML equations in HTML output
 - 196
- Automate Output with Jenkins
 - 10
- Automate Output with Travis CI
 - 12

B

- Built-in templates
 - 70

C

- Changing numbers for ordered lists
 - 140
- Changing scoring values in search results
 - 153
- Comments section
 - 63
- Context-sensitive help
 - 31
- Copy resources to output directory
 - 192
- Creating publishing templates
 - 122
- CSS styling
 - 132
- Custom templates
 - 70

- Customizing main page layout
 - 143
- Customizing output with CSS
 - 132
- Customizing output with HTML content
 - 134
- Customizing the footer
 - 146
- Customizing the menu
 - 143
- Customizing the tiles
 - 147

D

- DITA-OT process stages
 - 66

E

- Edit link to launch Web Author
 - 193
- Editing publishing templates
 - 125
- Excluding topics from search results
 - 157

F

- Facebook Like button
 - 174
- Flagging DITA content
 - 195

G

- Google Analytics
 - 179
- Google Search
 - 162

I

- Increase memory
 - 14
- Index terms layout page
 - 26
- Inserting HTML
 - 134

J

Jenkins integration	10		
L			
Localizing interface	171		
M			
Main layout page	17		
MathML equations in HTML output in Author mode	196		
MathML equations in WebHelp output	153		
Memory issues	14		
N			
Navigation links	143		
O			
Optimizing Japanese content indexing	154		
Optimizing search results	170		
Out of memory	14		
OutOfMemory	14		
Oxygen Styles Basket	122		
P			
PDF link	198		
Previous/Next arrows	143		
Publishing Template			
Adding to gallery	125		
Converting old templates	128		
Converting publishing templates to version 22	130		
Converting publishing templates to version 23	130		
Converting publishing templates to version 24	129		
Converting publishing templates to version 24.1	129		
Converting publishing templates to version 25	128		
Converting version 20 publishing templates to version 21	131		
Could not be loaded error message	128		
Creating	122		
Descriptor file	72		
Editing	125		
HTML fragments	78		
HTML page layout files	87		
Resources	75		
Running from a command line	126		
Sharing	127		
Transformation parameters	76		
Troubleshooting	128		
XSLT extensions	78		
Publishing Template package	71		
R			
Right-to-left languages	174		
S			
Search features	27, 28		

Search layout page

23

Search rules

27, 28

searchQuery parameter

169

Sharing publishing templates

127

Sharing templates

70

Syntax highlights in codeblocks

141

T

Topic layout page

20

Transformation parameters

104

Travis CI integration

12

Tweet button

176

X

XML documents

Author Mode editing

MathML equations in HTML output

196

XSLT Import extension points

118

XSLT Parameter extension points

118

XSLT-import

extension points

183

Copyright

Oxygen XML WebHelp Responsive plugin User Manual

Syncro Soft SRL.

Copyright © 2002-2023 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 WebHelp Responsive plugin End-User License Agreement, as well as information regarding support for this product, while under warranty, is available through the [Oxygen XML WebHelp Responsive plugin End-User License Agreement](#).

Terms and conditions: For the terms and conditions for using Oxygen XML WebHelp Responsive plugin, see [Oxygen XML WebHelp Responsive plugin End-User License Agreement](#).

Documentation: For the most current versions of documentation, see the [Oxygen XML WebHelp Responsive 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](#).