



Using DITA

Quick start guide to G&D's DITA future

Version 1.0

Printed on 24th November 2015



Helmut Scherzer (Editor)

Giesecke & Devrient / IBM
Prinzregentenstr. 159
81677 Munich
PO Box 80 07 29

Fon: +49 89 4119 2084

Mobile: +49 174 313 9891

mail: helmut.scherzer@gi-de.com

ID No.100.25003.230
© Copyright 2014 by
Giesecke & Devrient
Prinzregentenstr. 159
81677 Munich
PO Box 80 07 29

This document as well as the information or material contained is copyrighted. Any use not explicitly permitted by copyright law requires prior ^{test}consent of Giesecke & Devrient GmbH. This applies to any reproduction, revision, translation, storage on microfilm as well as its import and processing in electronical systems, in particular.

Trademarks

ezRead[™] is a registered trademark of Giesecke Devrient GmbH, München.

Java[™] is a registered trademark of Oracle system

IBM[™] is a registered trademark of IBM Corp.

Table of Contents

Tables	6
Figures	7
1 Installing the DITA environment	9
1.1 Installing the command line interface	10
1.2 Installing the DITA toolchain	11
1.3 oxygen installation	12
1.4 oxygen configuration	14
1.5 AHF installation	17
1.6 AHF configuration	17
1.7 Installing a plugin	17
1.8 ezRead Installation	17
2 Processing the DITA-OT	19
2.1 Using the command line	19
2.2 Running DITA with command line	20
2.3 Configuring oxygen for DITA-OT	21
2.3.1 Additional parameters	30
2.3.1.1 Default figure link text	30
2.4 Running oxygen scenarios	31
2.5 How DITA is processed...	33
2.6 PDF post processing	33
3 Important Topics	35
3.1 Using figures	35
3.2 Basic figure aspects	35
3.2.1 image-width	36
3.2.2 Wrap text around figures	37
3.3 Working with the Glossary	37
3.3.1 Creating a local glossary from a master list	38
3.4 Working with equations	38
4 DITA-OT extensions	39
4.1 Extensions on the paragraph.	39
4.2 Extensions on section	40
4.3 oxygen Annotations	41
4.4 Extensions to tables	42
4.4.1 Repeat table header	42
4.5 Extensions to fig/image	43
4.5.1 Creating Figures From Visio	43
4.5.2 Links within graphics	46
4.5.3 Tryout figures	49
4.6 Extension to links	52
4.6.1 Linking figures and tables	53
4.6.2 Trying out links	54

4.6.3 ezRead auto-linking	55
4.7 Extensions to Notes	56
4.8 Extensions to lists	57
4.9 Extensions to Mini-Toc	58
4.10 New page	59
4.11 Keep Lines together	60
4.12 Extension on the title element	60
5 Managing the front page	61
5.1 First page layout	61
5.2 Company Logo	61
5.3 Security Class	62
5.4 Watermark	63
5.5 Front picture	64
5.6 Second page layout	65
6 Author's support	66
6.1 Creating a Bibliography	66
6.1.1 Creating a Master Bibliography	66
6.1.2 Referencing external documents (Bibliography)	67
6.1.3 Generating a Local Bibliography.	68
6.1.4 Ignore Lists	69
6.1.5 Creating the oxygen scenario	69
6.1.6 Repair Master Bibliography	71
6.2 Creating a Glossary	71
7 MS-Word Docx 2 Dita conversion	72
7.1 Installing Docx2Dita	72
7.1.1 Style mapping	72
7.1.2 Create docx2dita scenario(s)	72
7.2 Running the docx2dita conversion	76
7.2.1 Prepare the DOCX for conversion	76
7.2.2 Converting docx2dita from oxygen GUI	78
7.2.3 Converting from the command line	79
7.2.4 Post Processing	80
8 Docbook to Dita conversion	81
9 CHM output	82
9.1 Installing CHM features	82
9.2 Producing CHM files	87
9.3 Changing CSM styles	88
9.3.1 CHM Font definition	88
9.4 Changes to the DITA-OT	89
9.5 Understanding the CHM process	91
10 Programming Stylesheets	97
10.1 Stylesheet Example	97
10.2 Steps to the first stylesheet	98

10.2.1 DGI conversion 99

11 Other Tools 101

11.1 Plant UML 101

12 Contribute to DITA-OT 102

12.1 Overview 102

12.2 GitHub 102

13 Starting with DITA 103

13.1 Todo's in this document 103

13.2 Why using a concept? 104

13.3 Why using a topic? 104

13.4 Why using a task? 104

13.5 Why using a reference? 104

14 Exercises 106

14.1 Paragraphs 106

14.2 Lists 106

14.3 Figures 107

14.4 Links 107

14.5 Notes 107

14.6 Index 108

14.7 Index Test 108

14.8 Creating the front page 109

14.9 Changing system variables 109

14.10 Extensions to tables 110

Appendix A - Bibliography 112

Glossary 113

Index 114

Tables

Table 1: ANT parameters	26
Table 2: Example using compressed rows	42
Table 3: Row colors	43
Table 4: Target 1	52
Table 5: Table in note	55
Table 6: Table in list	55
Table 7: Table in normal text flow	55
Table 8: Bibliography	66
Table 9: test table	108
Table 10: Bibliography	112

Figures

Figure 1: Finding the command prompt for installation	10
Figure 2: Copy command prompt to desktop	11
Figure 3: Copy the topic to the desktop	11
Figure 4: Fonts settings	15
Figure 5: Locate the DITA-OT	16
Figure 6: Disabling network requests	17
Figure 7: Setting the external DITA-OT (Example)	22
Figure 8: Duplicate the existing DITA Map PDF transformation scenario	23
Figure 9: Basic settings	24
Figure 10: Filter Settings	25
Figure 11: Parameters settings	26
Figure 12: Library settings	28
Figure 13: Output settings	29
Figure 14: Edit the args.figurelink.style	30
Figure 15: Empty xref options	31
Figure 16: Progress log during oxygen-scenario	32
Figure 17: Result window	32
Figure 18: Log window context dialog after right mouse click	33
Figure 19: Dita Process	33
Figure 20: Test figure	35
Figure 21: Test figure	36
Figure 22: Testing a formula (export as SVG)	38
Figure 23: Preview references in oxygen	41
Figure 24: Selecting Margins	44
Figure 25: Setting margins	45
Figure 26: Set Page width	45
Figure 27: Set "Fit to Drawing"	45
Figure 28: Set "Fit to Drawing"	46
Figure 29: Set "Fit to Drawing"	46
Figure 30: Plain figure - no width specified, the frame cannot be determined from the image size	49
Figure 31: Plain figure, image:width=50mm	49
Figure 32: fig:expanse=page, image:placement=break, width=50mm (ignored by expanse=page)	50
Figure 33: fig:expanse=column, image:align=right, width=50mm, placement=break	50
Figure 34: image:outputclass=page, placement=break	50
Figure 35: image:outputclass=flow, align=right, width=50mm, placement=break	50
Figure 36: image:width=50mm, align=left, placement=break	51
Figure 37: image:align=right, width=50mm, placement=break	51
Figure 38: image:align=right, width=50mm, but placement=inline (inline does not obey 'align')	51
Figure 39: Image with #hstarget1..4	52
Figure 40: Configuring mini-TOC in runtime parameters	59
Figure 41: Front page definition with logos	62
Figure 42: Security Class definition in Author mode	63
Figure 43: Watermark specification in the front page	64
Figure 44: Front picture definition in oxygen author mode	64
Figure 45: Creating a local bibliography	68
Figure 46: New scenario	70
Figure 47: Select scenario type	70
Figure 48: Create Scenario	71
Figure 49: All scenarios view	73
Figure 50: Open oxygen scenarios	73
Figure 51: Change the scenario options	74
Figure 52: Edit Parameters	75
Figure 53: Edit output parameters	76
Figure 54: Select the 'styles' context menu	77

Figure 55: Apply styles	78
Figure 56: Style Dialog	78
Figure 57: New transformation	82
Figure 58: Type selection	83
Figure 59: Parameters tab	84
Figure 60: Filters settings	85
Figure 61: Advanced tab	86
Figure 62: Output specification	87

1 Installing the DITA environment

1.1 Installing the command line interface	10
1.2 Installing the DITA toolchain	11
1.3 oxygen installation	12
1.4 oxygen configuration	14
1.5 AHF installation	17
1.6 AHF configuration	17
1.7 Installing a plugin	17
1.8 ezRead Installation	17

How to install everything to print the first PDF from DITA

It takes more than downloading the DITA-OT in order to start with DITA. Following the next chapters, however, will make it easy to install.

The suggested installation process suggests some default directories. The associated batch files assume these directories.



Note: If possible, do not change the suggested directories, this will make any service and maintenance easier.

Overview

In general the DITA Toolkit installation requires the following major steps

Install batches

There are a couple of batch files to help processing in various situations.

As an example ... to create a new DITA file, there is a batch file `newDita.bat` which copies a working set of a DITAMAP into the current directory. This template can be used and processed immediately without further editing.

Install XML editor (oxygen)

To write DITA files, an XML editor is required. The present toolchain supports the `oxygen` editor which is very powerful. As it comes with a license (cost approx. \$500) it is possible to install a quite powerful free XML-editor `SernaFree`.

Send a request to Helmut Scherzer to obtain this alternative.

Install AHF formatter

To create PDF files from DITA(XML) input a special formatter is required. This is another quite expensive investment which should be done as server license, in case that many authors need install DITA. We use the [Antenna House formatter](#) in contrast to the also popular [XEP formatter](#) which, however, entirely works JAVA based - which is why we prefer the binary coded AHF.

For single users, a stand-alone AHF (named license) is available for about [\\$1250](#)

Install the DITA-OT

The DITA-OT (Dita Open Toolkit) is a open-source set of stylesheets and build files in order to create different output formats from DITA sources. The PDF formatter (AHF) is only required because the available PDF formatter in the DITA-OT is very poor in features - it does not satisfy modern documentation's needs.

Install reference documentation

Together with the programs, a set of reference documentation is available, one of that is the DITA template which is a ready to go document as a start.

Install ezRead environment

The ezRead [ezRead#1] environment is a powerful tool in order to refer to chapters of external PDF files. It enhances the Adobe Acrobat (licensed product) with more than 30 new very powerful functions which allow the preparation and addressing of external documentation - even if it comes as a SECURE document.

1.1 Installing the command line interface

Whether to use GUI or command line?

The DITA installation is based on batch files and has to be done through the **command line interface**. Of course a batch file can also be launched from the Windows Explorer but if something goes wrong, the error information is not available because the command line window disappears after the batch execution.

The command line can be found in **START** → **All Programs** → **Accessories** → **Command Prompt**

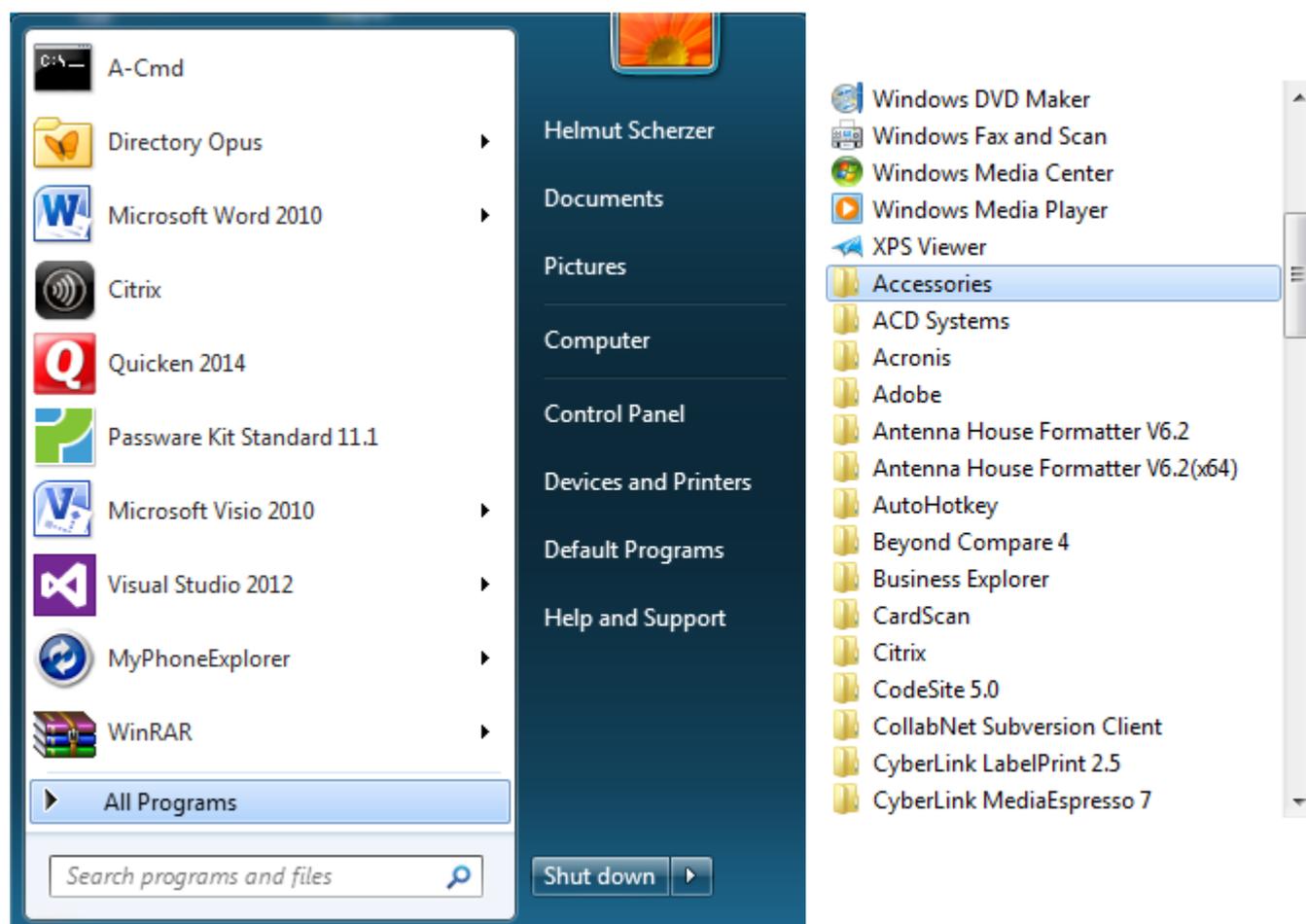


Figure 1: Finding the command prompt for installation

In the **Accessories folder you will find the command prompt**

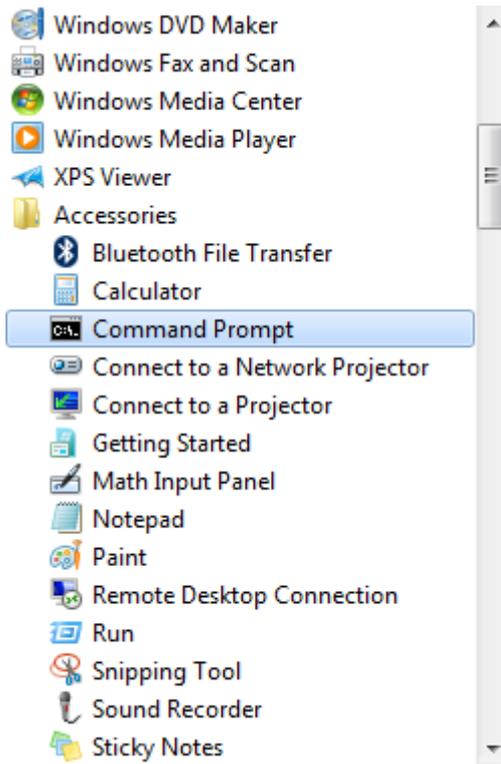


Figure 2: Copy command prompt to desktop

Use the **right mouse key** to *drag* the command icon  to your desktop. Answer the following dialog panel with **Copy**

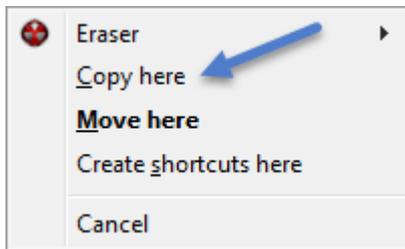


Figure 3: Copy the topic to the desktop

This installs a permanent command prompt on you System. I recommend this since the command line is an essential tool for further processing.

1.2 Installing the DITA toolchain

The DITA-Toolchain installation comprises several steps

1. Copy `01_Extract.Bat` and `InstallDitaTools.rar` to you local `C:\ProgramData\Install`
2. Run `01_Extract.BAT`



Note: What it does ... it will extract the installation files in the `C:\ProgramData\Install` and copy several files to their final directories.

3. The `01_Extract.Bat` will automatically launch `02_InstEnv.bat` which performs several settings to make the system work properly.



Note: What it does ... it will copy some files to their correct places and it will in particular set your local environment variables.

4. If oxygen is part of the package, install [Oxygen](#) from the `C:\ProgramData\Install\01_oxygen` directory.



Warning: You shall install exactly to `C:\ProgramData\oxy17` because the default installation will typically install in `C:\Program Files` which in many industrial PC's can only be fully accessed using administrator rights.

As [Oxygen](#) stores much default data (e.g. the default DITA-OT) in its installation directory, you would not be able extend the DITA-OT unless you are administrator.

`C:\ProgramData` (or the path that you see in the environment variable `%ProgramData%` is a place where the current user is allowed to write and it is therefore a good place to put your installation.

5. if the Antenna House Formatter (AHF) is part of the package, install the Antenna House Formatter to `C:\ProgramData\AHF`. You have to explicitly type this Path into the installation dialog - it will not be suggested by a drop-down list.
6. For AHF you will need ADMIN rights. Unfortunately for the normal user's PC, you have to elevate your rights (e.g. by utilities like "Forty-Two") .
7. Copy `newDita.Bat` from `C:\ProgramData\batch` to the Desktop. Later you will always copy this batch file to a potential directory that shall start a new DITA file.



Note: Do not make a shortcut - this won't work, because the batch file checks the directory in which it exists and you don't want to create new DITA documents on the desktop.

8. Install the `ezRead Tools` - they are essential if you use links from/to PDF (I highly recommend to do so). There is a documentation available under `C:\ProgramData\ezRead\Books` which explains the installation in [\[ezRead#1.1\]](#).
9. You are done ... to start with your first working DITA file
 - a. go to any directory of your choice
 - b. only if you use the GUI: copy `newDita.bat` into this directory, in the command line it is only important that you launch `newDita.bat` from the new directory that shall contain your next DITA project.
 - c. run the `newDita.bat`, it copies a reference DITA book to the present directory which is as complete as you can process it immediately.

1.3 oxygen installation

[Oxygen](#) should be installed according to the process described in

[oxygen installation path](#)

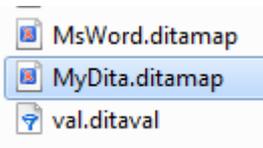


Attention: Do not forget to associate `.ditamap` with oxygen, this is not done automatically during [Oxygen](#) installation.

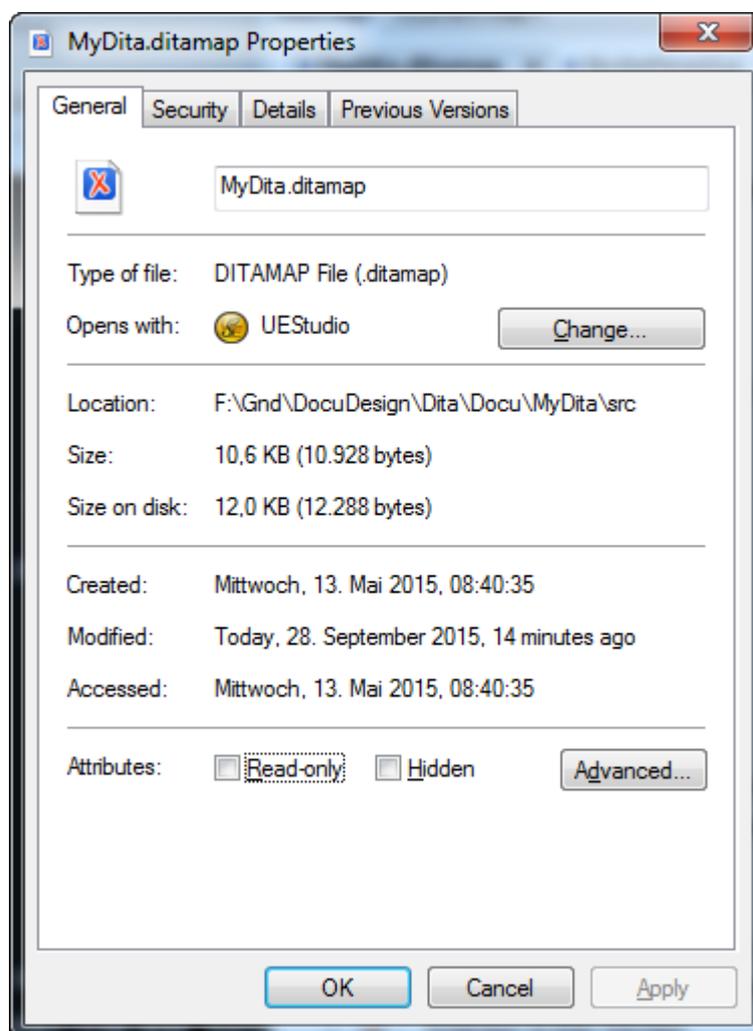
Associate file types (e.g. ditamap) to oxygen

To allow opening [Oxygen](#) if you double-click a DITA file (e.g. .ditamap or .dita) you need to associate the file type to [Oxygen](#).

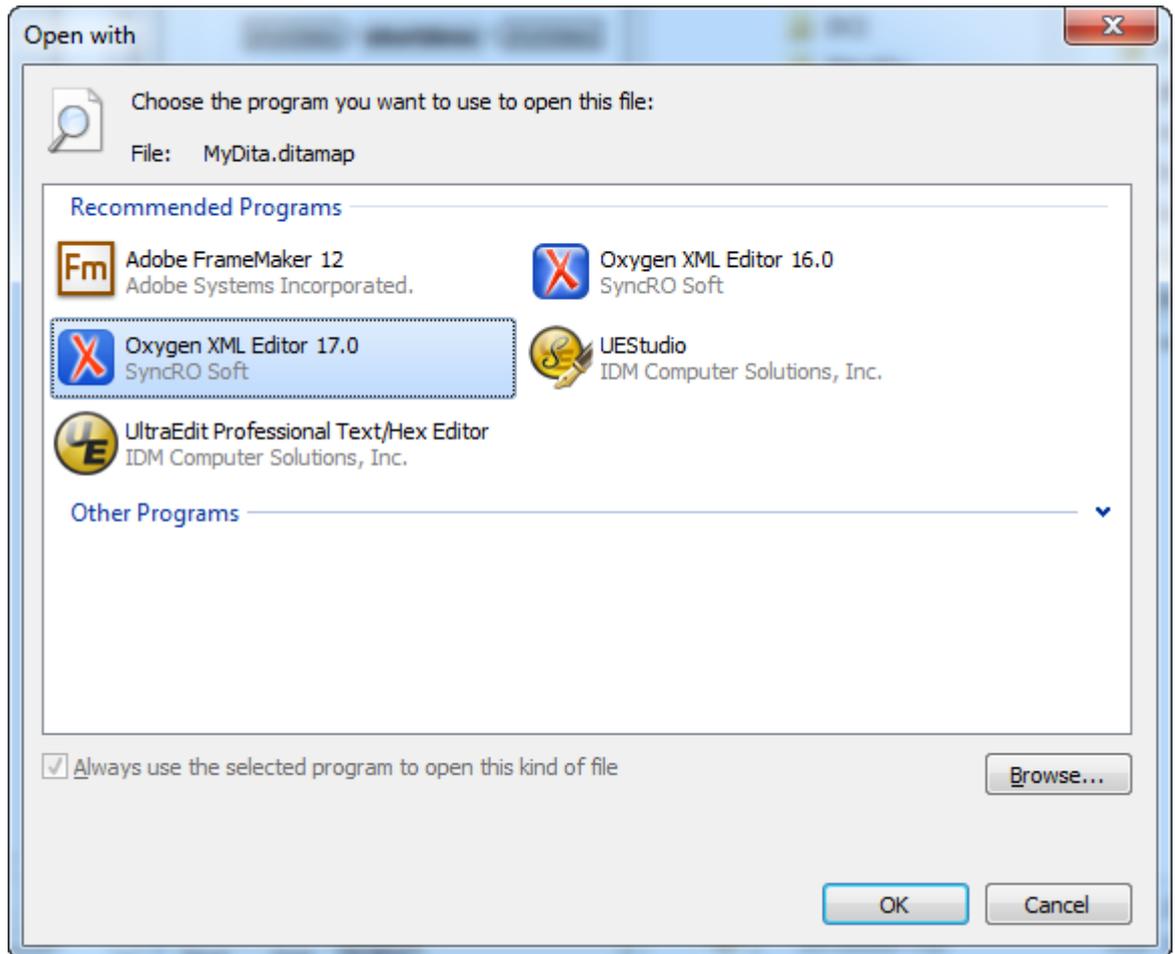
1. Open the Windows™ file explorer and navigate to any such file (e.g. .ditamap that should be associated).



2. Right click the file and select the **Properties** button
3. Click on **Change** in the next dialog to change file association



4. Select Oxygen XML Editor 17.0 in the example to associated [Oxygen](#) to the file type



Note: If **Oxygen** does not appear in the list, you can try the **Other Programs** or you can always use the **Browse** button to navigate to the oxygen installation path (e.g. C:\ProgramData\oxy17) and select **Oxygen.exe** from there.

1.4 oxygen configuration

oxygen is configured through the **Options** → **Preferences** menu item. After installation, you should set the **Fonts**

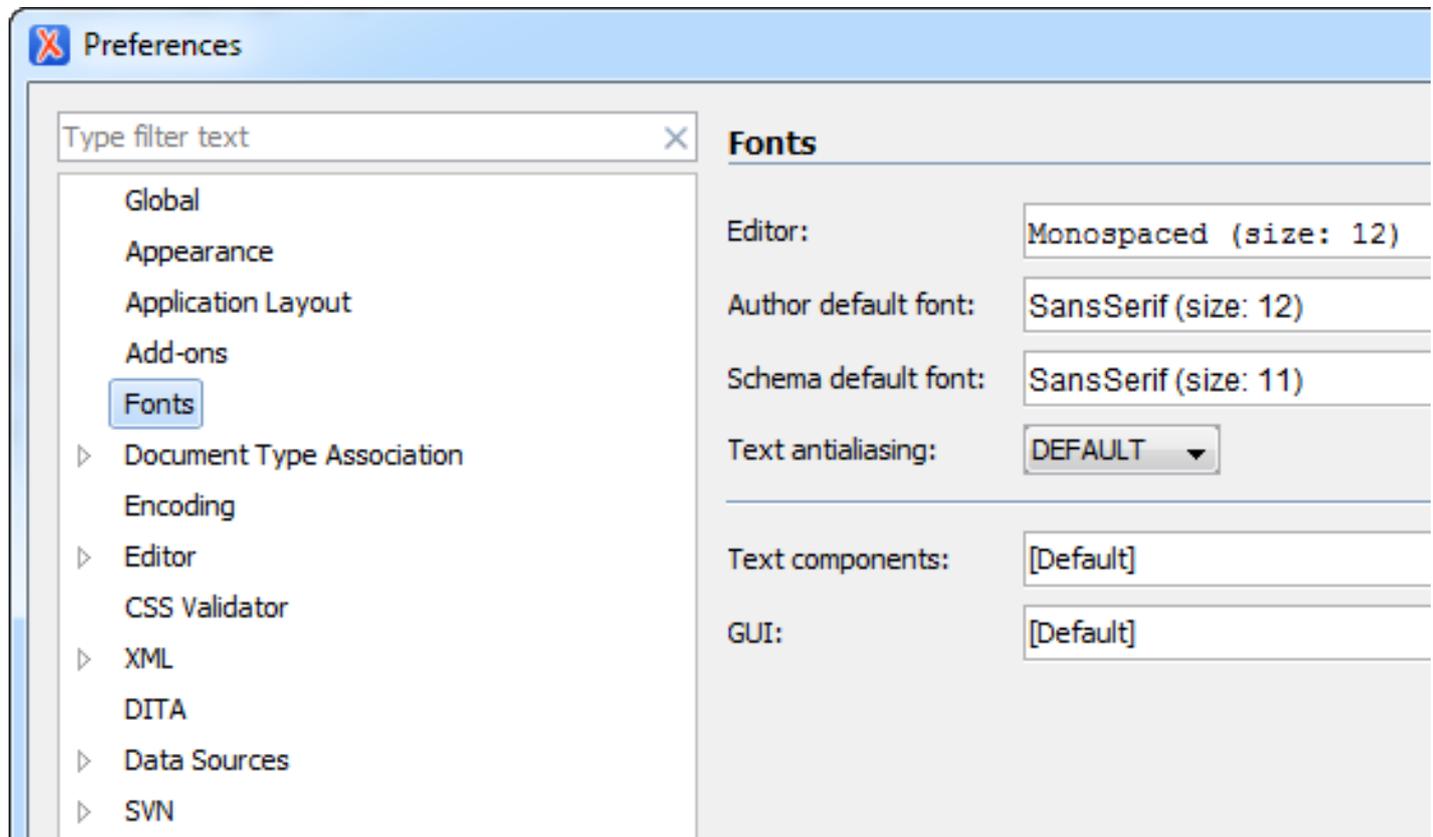


Figure 4: Fonts settings

oxygen should know where to find the DITA-OT, although we specify this again in the appropriate scenarios

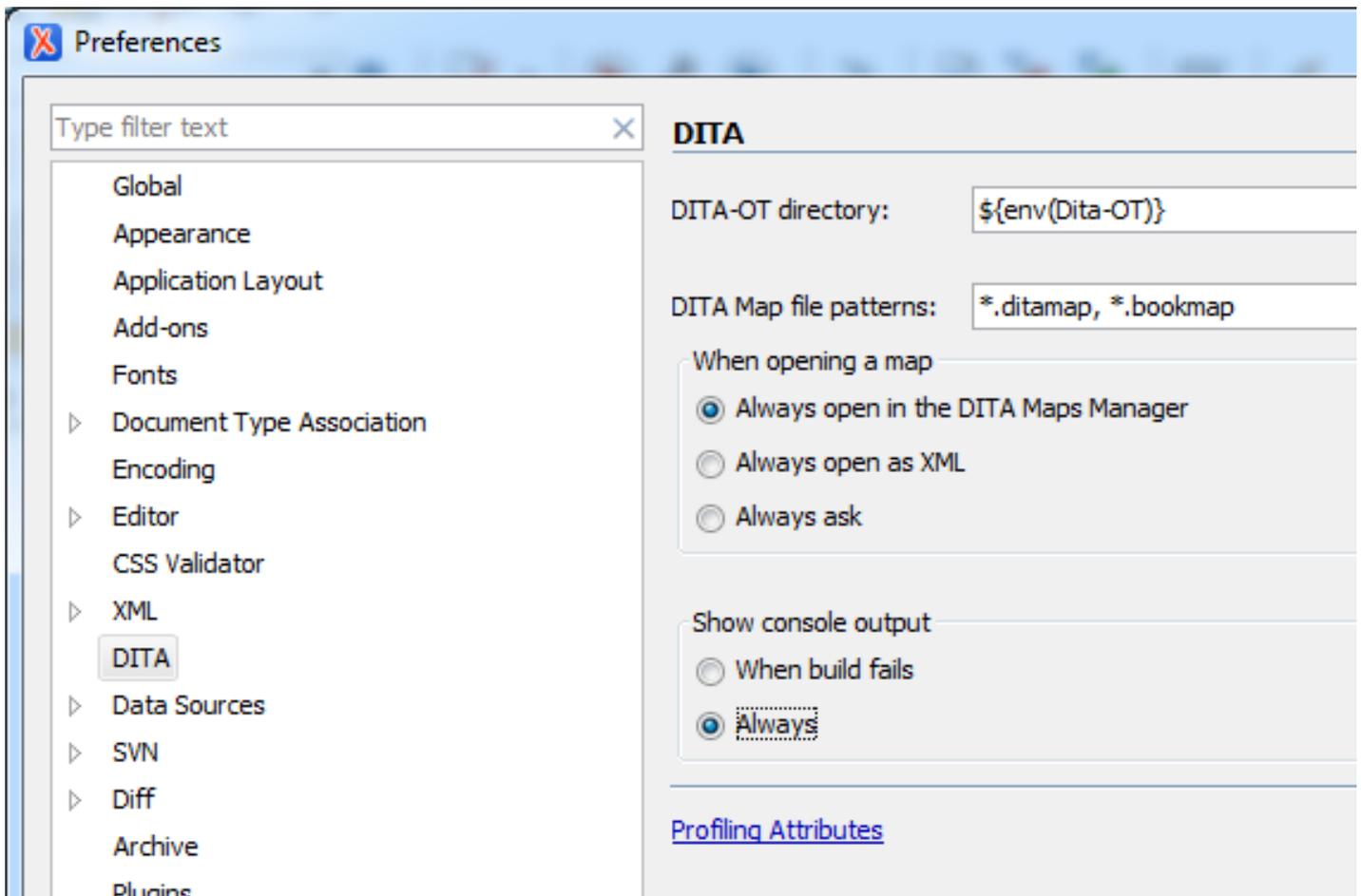


Figure 5: Locate the DITA-OT



Note: According to [Figure 2](#) set set the DITA-OT directory to the environment variable *Dita-OT* which we created during installation. This is quite practical as it is easier to change that environment variable (for other applications) than several changes in the scenarios and the oxygen editor.

proxy settings

To avoid the annoying proxy warnings when oxygen starts, you may disable oxygen's desire to contact the network.

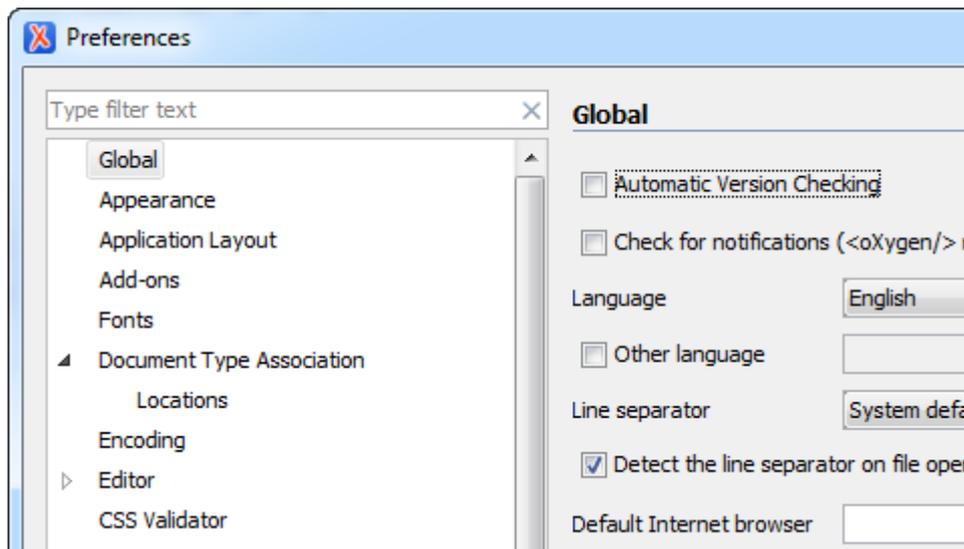


Figure 6: Disabling network requests

Disable the **Automatic Version Checking** and the **Check for notifications**.

1.5 AHF installation

1.6 AHF configuration

1.7 Installing a plugin

On a plain download and un-zip of the DITA-OT, there is no customized plugin that would enhance the basic features. If you need to create your own customized plug-in on the DITA-OT, you need to do several steps in order to integrate that plugin into the build process.



Attention: The ezRead installation provides with an already configured and customized plugin based on the very latest version of the DITA-OT. You do not need to do any further action beyond installation

The best reference for the customized plugin integration is [DtPrt#2]. To summarize what it requires to recognize a plugin simply is a `plugin.xml` installed directory under the new plugin directory. That plugin shall specify its own transfer type.

Issuing `ant -f integrator.xml` will search all such `plugin.xml` and and make them available for appropriate build files that specify the associated *transfer type*.

1.8 ezRead Installation

Read [\[ezRead#1\]](#) for all you ever wanted to know about ezRead installation.

2 Processing the DITA-OT

The following chapters explain what you need to know in order to process with the DITA-OT.

Topics	2.1 Using the command line	19
	2.2 Running DITA with command line	20
	2.3 Configuring oxygen for DITA-OT	21
	2.4 Running oxygen scenarios	31
	2.5 How DITA is processed... ..	33
	2.6 PDF post processing	33

2.1 Using the command line

Why command line?

LINUX users do hardly ask this question because for them it is too obvious that for many situations using command line input is much faster than finding the right button in a GUI application. On the other hand, if you use the command line on Windows operating system, your colleagues will start kidding on you since you are "back to the old DOS time". Yet the reasons to use the command line are as popular for Windows as they are for LINUX.

- If you know the commands, most system administration actions are much faster to realize than through a GUI
- Through commands and associated script files, you have much better control over the process by
 - creating log files
 - pause a process for the investigation of intermediate results
 - writing a test script
 - tailor process and sequences more powerful and faster than through programming a GUI

On the down-side, using the command line requires knowledge about the commands and the navigation to a file in a path with a nesting-depth of 10 levels can be tiring compared to the easyness of a (x-)windows based GUI.

To finalize such endless discussion ... You will be most powerful in your computing life, if you do both, each at its optimal application field.

The most important Windows™ commands

- md** make (=create) directory
- cd** change to directory, where you could use \`<name>` to start from the root or just the `<name>` to start from your local position.
 - Using `cd <partial name><tab>` will expand the name (when it is unique) or parse through the candidates on every `<tab>`
 - Using `<Shift><Tab>` will bring you back (revers direction candidates)

rd remove directory, using `rd <name> /s` will delete als subdirs. Use `rd <name> /s /y` will not even ask removes the directory

dir show the directory. Use option `/p` to pause on page, use

- `<name> = ..` is the parent directory
- `<name> = .` is the present directory
- `<name> = \<name>` starts from root



Note: The unix notation for the path using a slash "/" will also work in Windows™

copy <source> <target> copies a file from `<source>` to `<target>`

xcopy <source> <target> is a more powerful copy for files and directories

help <command name> helps you to understand the syntax of `cd`, `dir`, `rd` ...

2.2 Running DITA with command line

The first DITA file is easy to create. Once you have opened a command line, you may create any working directoy - in our example it shall be `F:\Work>`

To get this directory created do the following

```
C:>F:
F:>md Work
F:>cd Work
F:\Work>newdita full
F:\Work>cd book
F:\Work\book>r
```

and the first book ist produced.

The first steps simply create a work directory, some basic explanation is given in [The most important Windows commands](#)

Launching `newdita full` copies a reference book from `C:\ProgramData\Dita\RefDita\` to the present position (here `F:\Work>`)



Important: The `full` parameter is required to add the *command line batches* to the copy.

The `r.bat` is calling `build\CreatePdf.bat` which actually invokes the `book.ant` file to create the first PDF.

Get the log file If anything is wrong with your installation you can launch

```
F:\Work\book>lg
```

which shows the log file that was created. This file can be copied to some support instance (e.g. Helmut Scherzer) to give you advice what's wrong with the installation.

2.3 Configuring oxygen for DITA-OT

Oxygen XML Editor comes bundled with a DITA Open Toolkit, located in the `[OXYGEN_DIR]/frameworks/dita/DITA-OT` directory.

Starting with Oxygen XML Editor version 17, if you want to use the external DITA OT for all transformations and validations, you can open the **Options** → **Preferences** dialog box and go to the DITA page, where you can specify the DITA OT to be used.

Otherwise, to use an external DITA Open Toolkit, follow these steps:

1. Verify that your system variables are set correctly after your installation. The following is an example of a typical set of parameters relevant for further processing.

```
ANT_HOME=C:\ProgramData\Dita-OT2
ANT_OPTS=-Xmx1600m -Xms1600m
AXF_OPT=C:\ProgramData\ezRead\RefDita\settings\AHFSettings.xml
CLASSPATH=C:\ProgramData\Dita-OT2\lib\saxon.jar; \
           C:\ProgramData\Dita-OT2\lib\saxon-dom.jar; \
           C:\ProgramData\Dita-OT2\lib\xercesImpl.jar; \
           C:\ProgramData\Dita-OT2\lib\xml-apis.jar;
Dita-Input=MyDita
Dita-OT=C:\ProgramData\Dita-OT2
DitaLog=log
DitaLogFile=logahf.txt
DitaOutputDir=..\pdf
JAVA_HOME=C:\Program Files\Java\jdk1.7.0_25
ProjDocRel=C:\ProgramData\ezRead\Documentation
ProjGfx="..\gfx"
```

These variables are important if you use the command line interface. With the oxygen internal invocation some of the entries are directly entered into the oxygen dialogs (see below) so the remaining important system variables are:

```
ANT_HOME=C:\ProgramData\Dita-OT2
AXF_OPT=C:\ProgramData\ezRead\RefDita\settings\AHFSettings.xml
```

oxygen cannot interpret system variables in the parameter list, therefore the following system variables need to be explicitly coded in the Advanced tab

```
ProjDocRel=C:\ProgramData\ezRead\Documentation
ProjGfx="..\gfx"
```

The `ProjDocRel` setting [[AHF#relurl](#)] is used in order to make relative paths to the stub directory [[ezRead#7.1.6](#)]. Unfortunately the Antenna House Formatter creates URLs (`file://<path>`) notation to realize the links in the PDF. On click - this triggers the Internet Browser instead of opening the PDF directly. That behavior can be fixed with the `ezRead` conversion function [[ezRead#12.1.20.1](#)].

2. Edit your transformation scenarios and in the **Parameters tab** change the value for the `dita.dir` parameter to point to the new directory.

If your external Dita-OT is e.g. in `F:\Dita-OT2` then the setting should be as follows

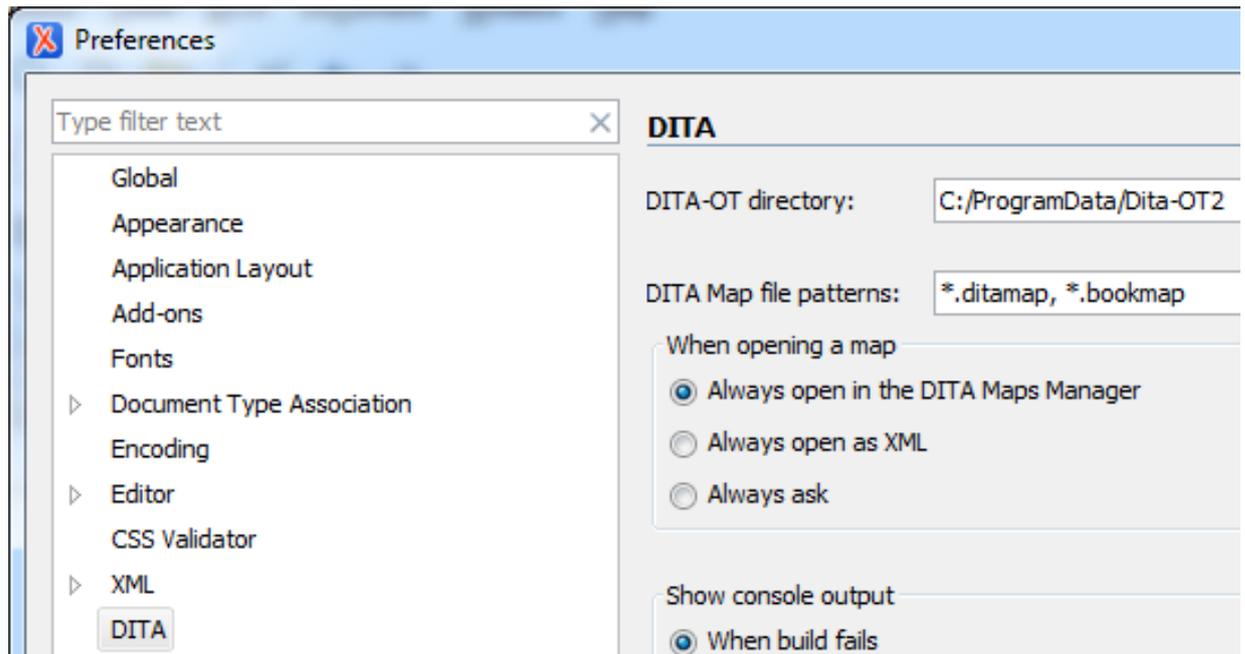


Figure 7: Setting the external DITA-OT (Example)

- 3. Copy an existing transformation scenario. Therefore select **DITA Maps** → **Configure Transformation scenarios** and the following panel will appear

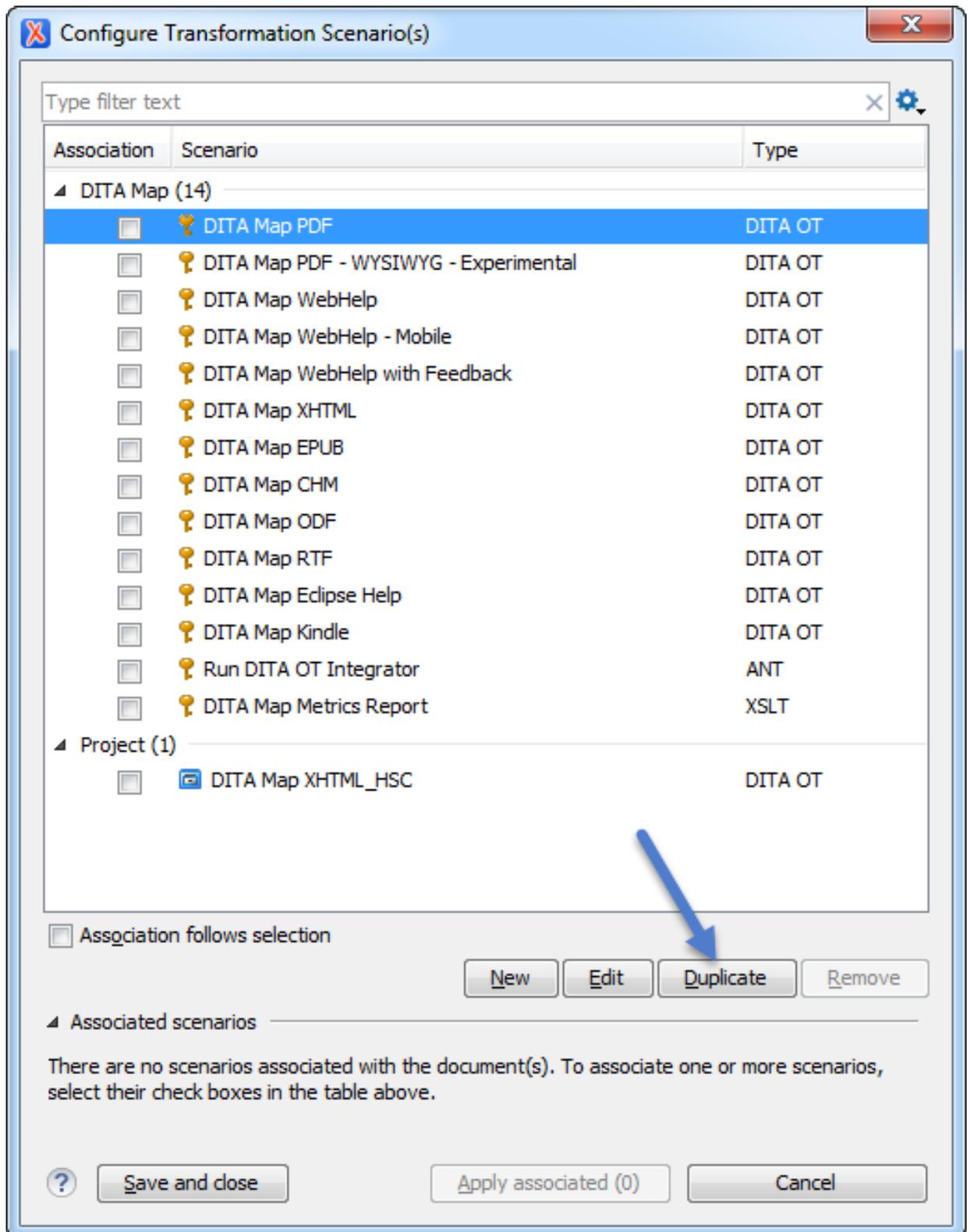


Figure 8: Duplicate the existing DITA Map PDF transformation scenario

After the duplication you will see the new transformation scenario

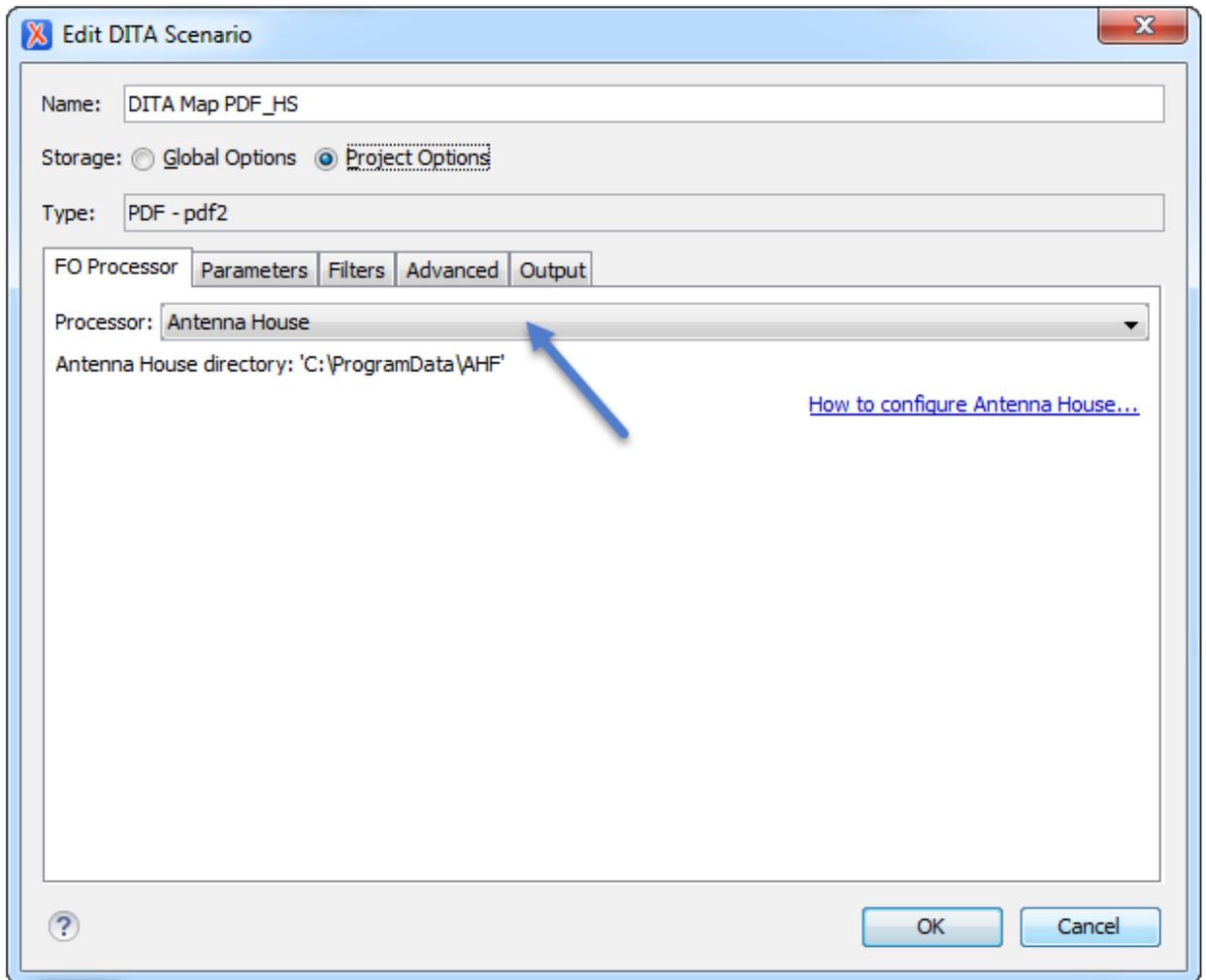


Figure 9: Basic settings

4. Assign a name to the transformation scenario (here `DITA Map PDF_DCI`)
5. Select the `Antenna House` processor as FO-processor
6. Switch to the **Filters** tab

7. Goto the Parameters tab

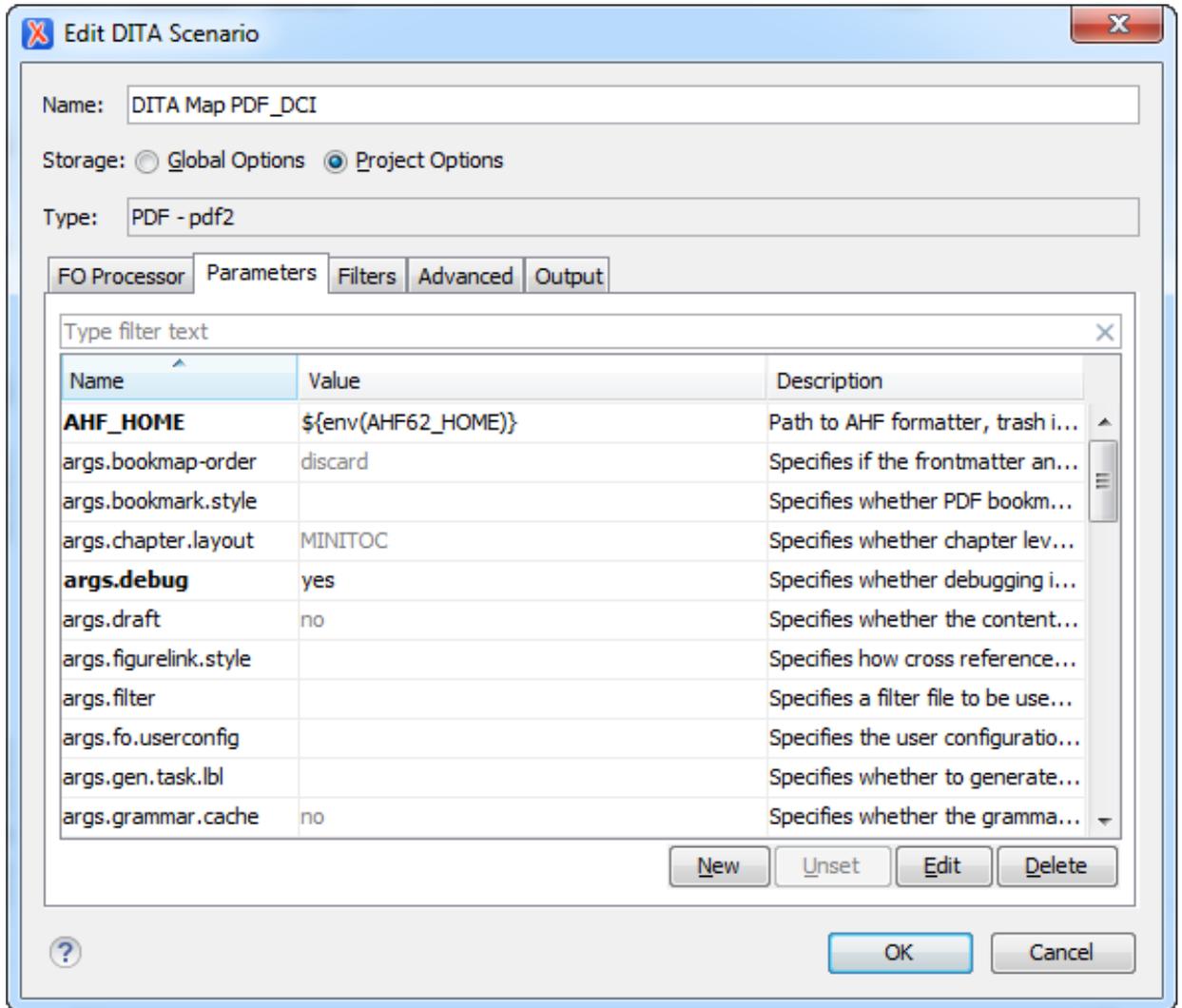


Figure 11: Parameters settings

8. Select each of the following parameters and **Edit** its parameters to the values suggested below.

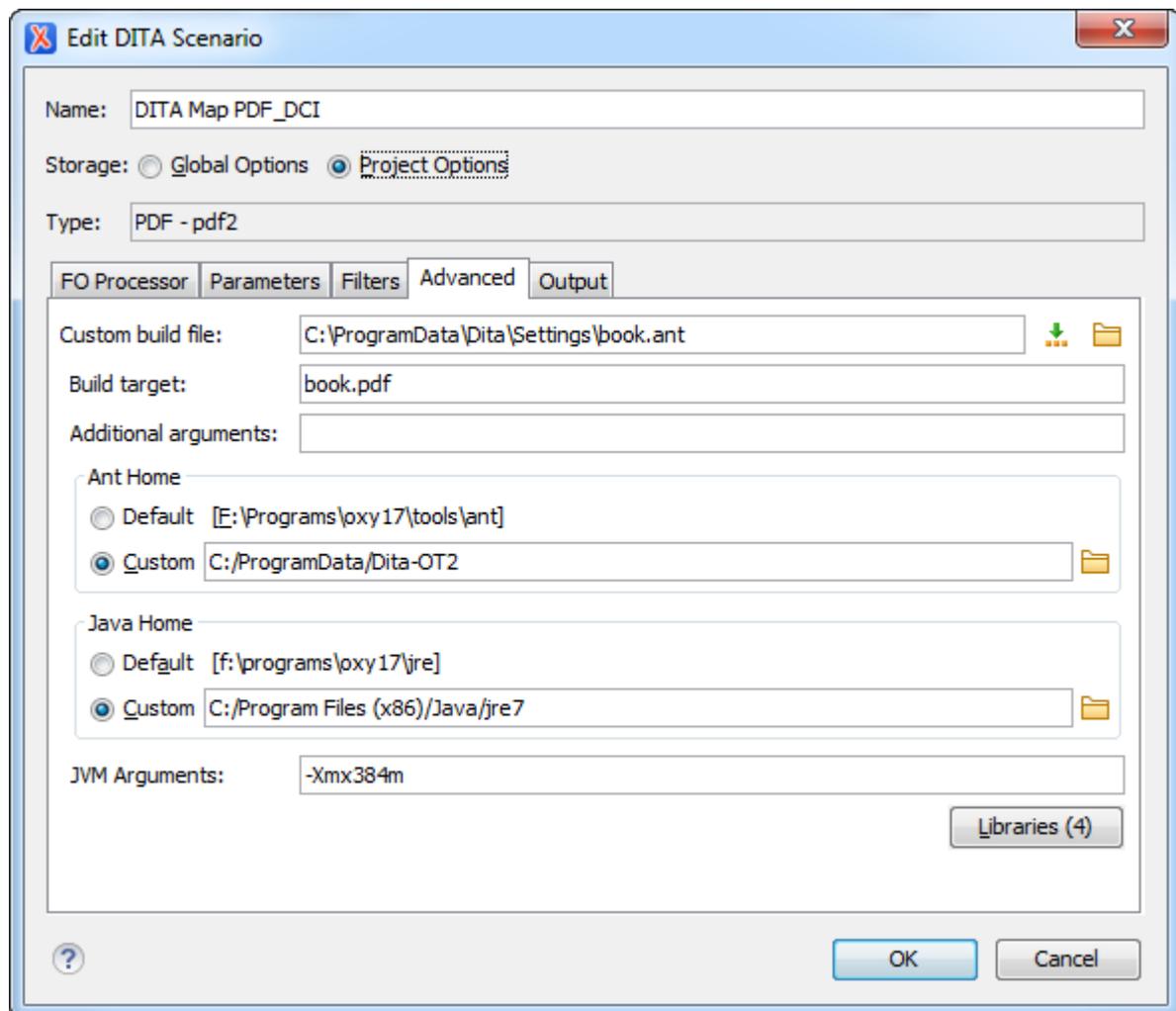
Table 1: ANT parameters

Parameter	Value
AHF_HOME	<code>\${env(AHF62_HOME)}</code>
 Note: You must create this entry with New	
args.debug	yes
args.input	<code>\${cf}</code>
args.logdir	<code>\${cfd}/../build/log</code>
args.rellinks	nofamily
clean.temp	no
customization.dir	<code>\${env(Dita-OT)}\plugins\com.ref1.pdf\cfg</code>
dita.dir	<code>\${env(Dita-OT)}</code>

Table 1: ANT parameters

Parameter	Value
dita.temp.dir	\${cfld}/../build/temp
env.AXF_OPT	\${env(AXF_OPT)}/AHFSettings.xml
 Note: You must create this entry with New	
gfxPath	\${cfld}/gfx
org.dita.pdf2.use-out-temp	true
pdf.formatter	ah
ProjDocRel	\${env(ProjDoc)}
retain.topic.fo	yes
transtype	custpdf

9. Go to the **Advanced** tab



No entries should be in the **Additional Parameters** field, all of them are already entered in the Parameters tab

10., click the **Libraries** button

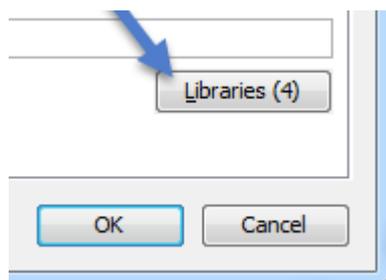
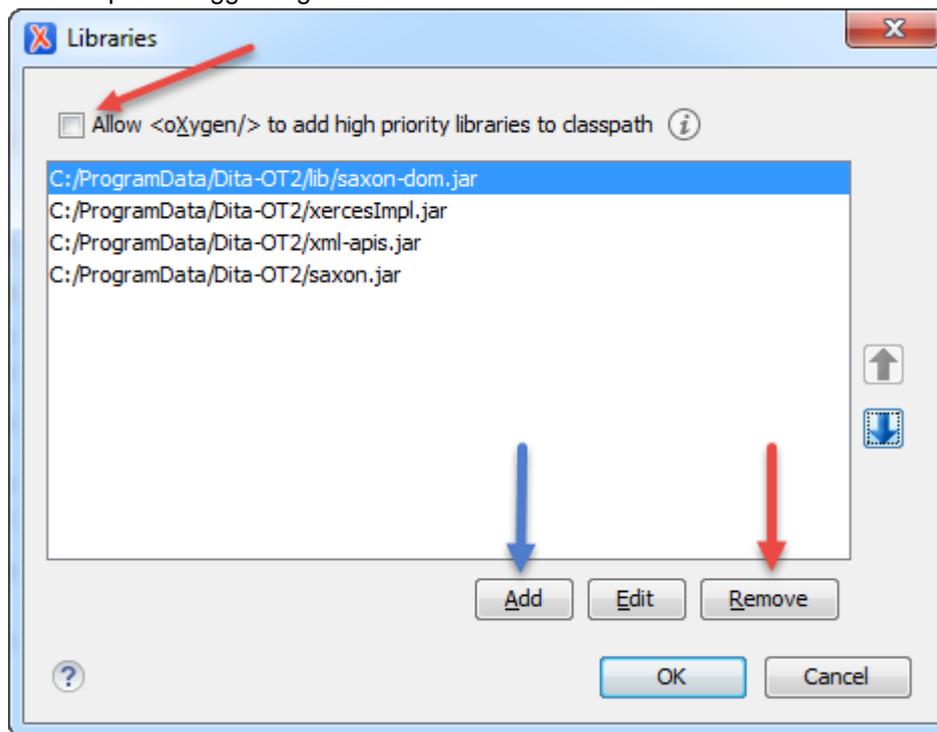


Figure 12: Library settings

and there will be a panel suggesting a lot of default



libraries.

- a. Uncheck Allow Oxygen to add high priority libraries to classpath.
- b. Select all existing libraries and **Remove** all of them.
- c. **Add** the libraries according to their position in your DITA-OT.



Note: For G&D, the location of the DITA-OT is in `C:\>ProgramData\Dita-OT2`. This is automatically implemented during **installation**.

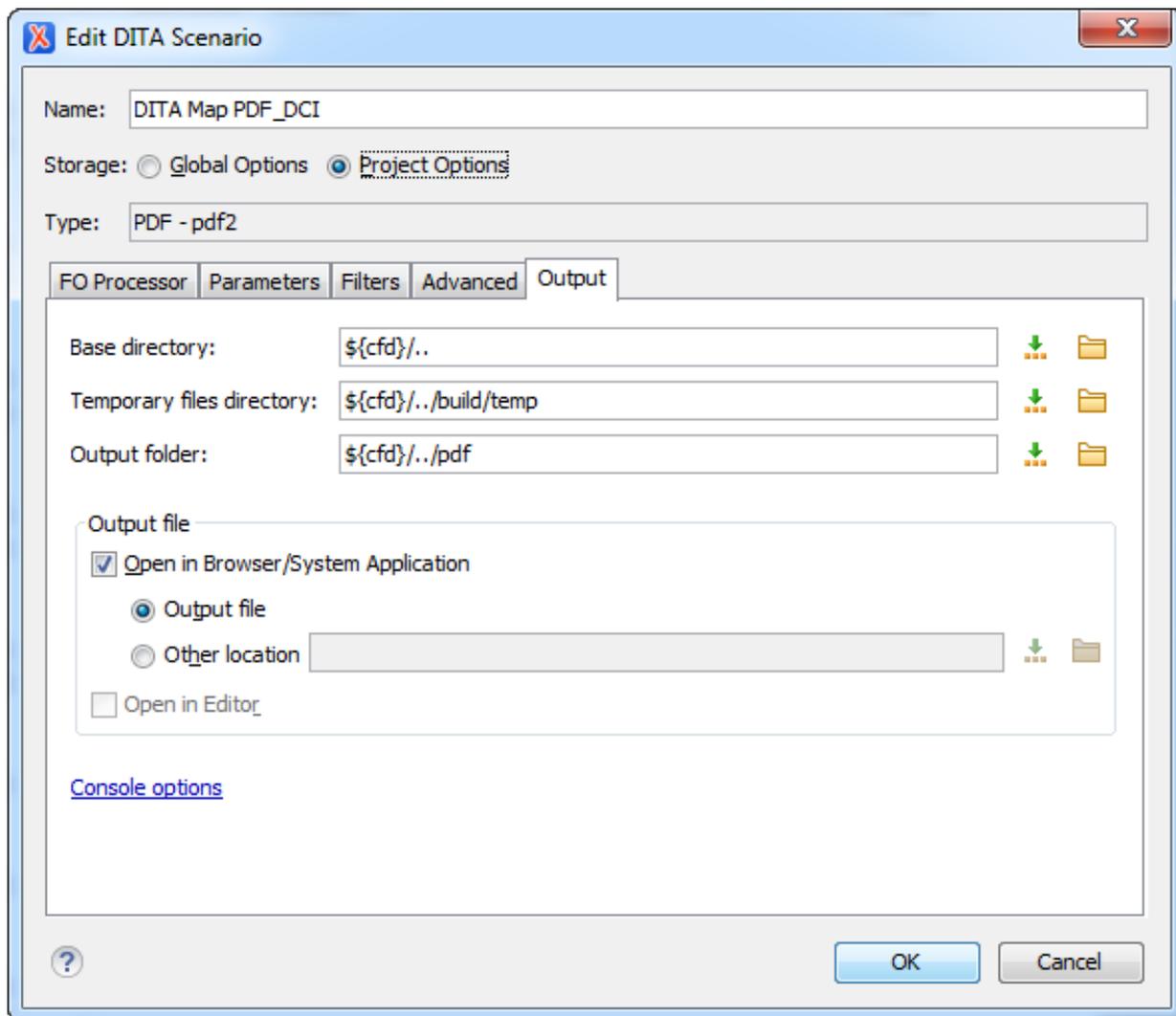
11. Go to the **Output** tab

Figure 13: Output settings

12. Enter the settings as shown in [Working with equations](#)

Important: The `${cfd}` directory points on the location of the `<document>.ditamap`. There are technical reasons (e.g. resolving graphic links) to maintain the directory structure as suggested here.



Tip: The original filepath is `${frameworksDir}/dita/DITA-OT`

Possible extensions

If there are also changes in the DTDs and you want to use the new versions for content completion and validation, go to the *Oxygen XML Editor preferences* in the *Document Type Association page*, edit the DITA and DITA Map document types and *modify the catalog* entry in the Catalogs tab to point to the custom catalog file `catalog-dita.xml`. You may consult [\[oxy17#8.6\]](#) for further information.

Related Information

<http://www.oxygenxml.com/dita/styleguide/webhelp-feedback/index.html>

2.3.1 Additional parameters

There are some other parameters which are no essential to rund the first build. However, the advanced user will be curious how far s(he) can go with configuration.

2.3.1.1 Default figure link text

The DITA-OT allows to parametrize the default text to be generated when you create a cross reference `xref` to a figure or table. You need to edit the parameters of your DITAMAP PDF scenario

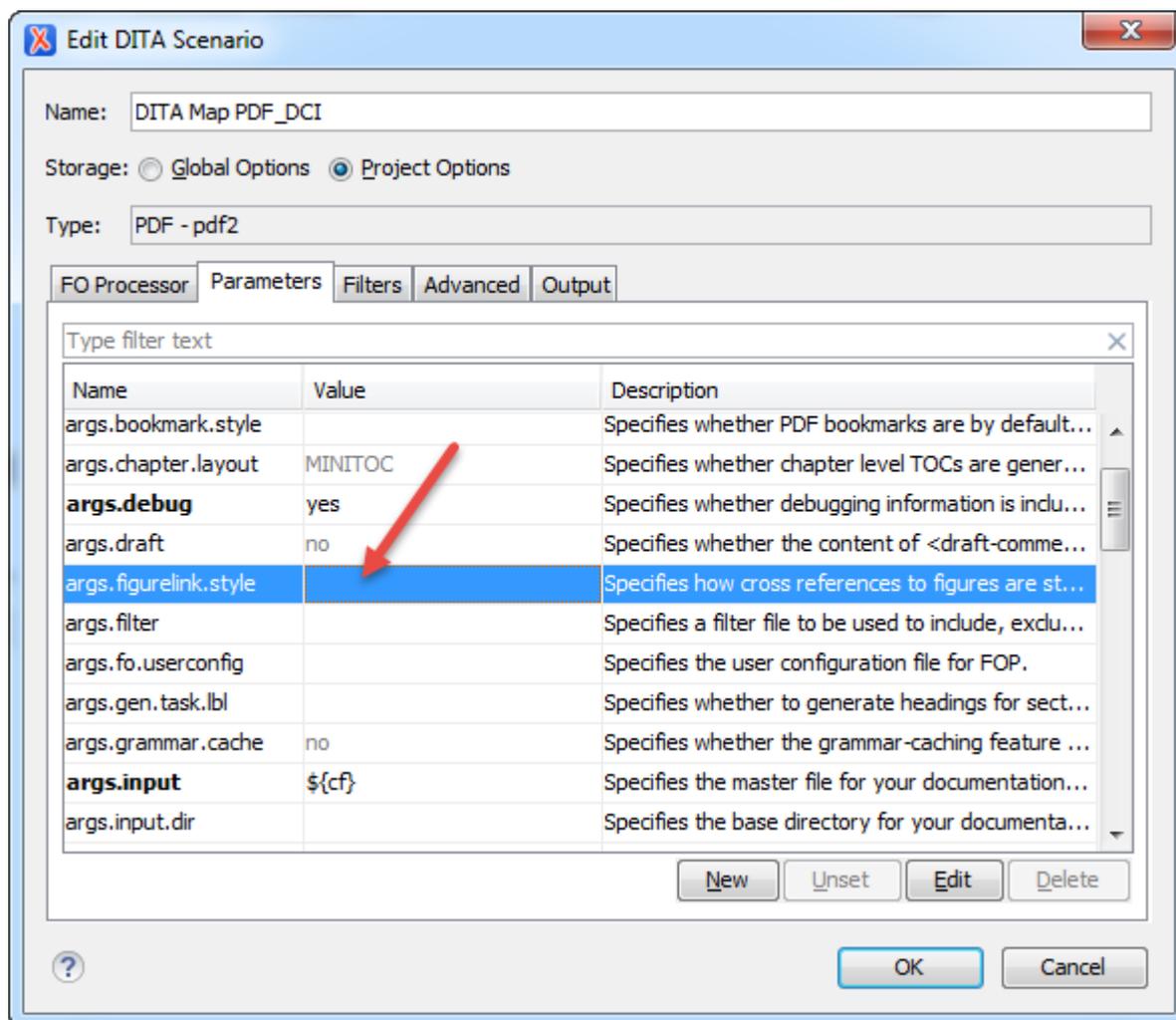


Figure 14: Edit the args.figurelink.style

The parameters allows you the following default settings for an empty `xref` to a figure

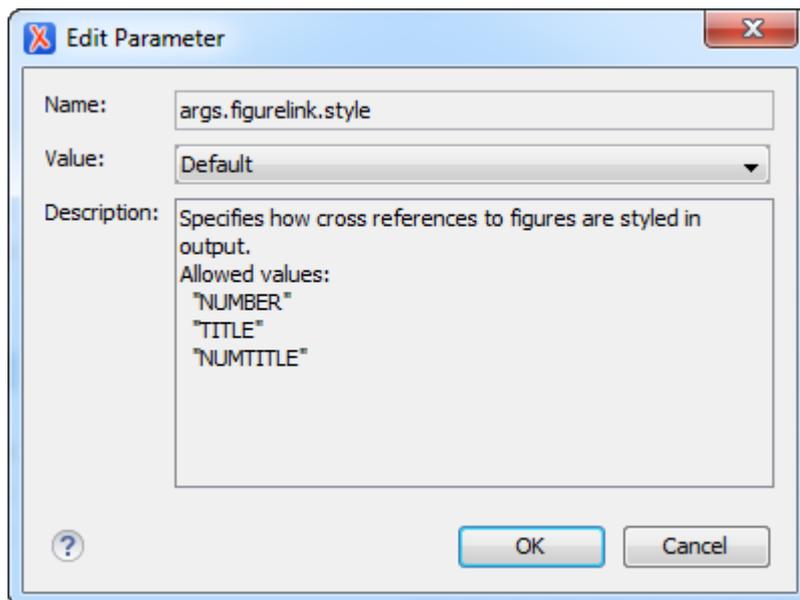


Figure 15: Empty xref options

NUMBER

will show like *Figure 5.1*

TITLE

will show the caption text like *"Creating scenarios"*

NUMTITLE

will show label and caption like *Figure 5.1 "Creating scenarios"*

Using the special `outputclass` option as described in [Chapter 4.6.1](#)

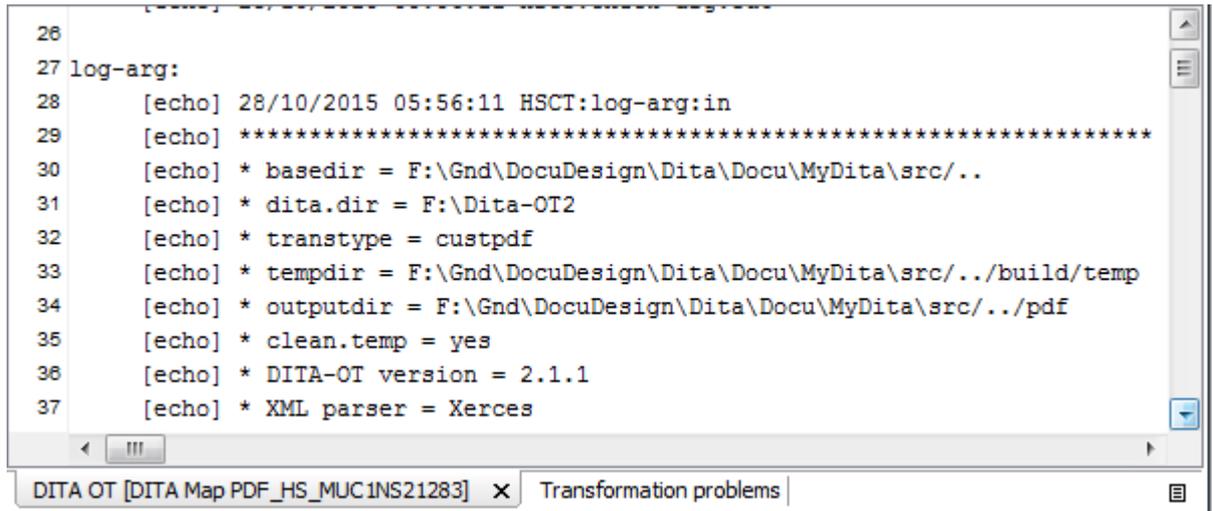
2.4 Running oxygen scenarios

To produce PDF and/or CHM you need to create an oxygen scenario according to [2.3 "Configuring oxygen for DITA-OT"](#) and associate it to your DITAMAP file.

To run such associated scenario

- select the DITAMAP file to be in the focus, this will let oxygen know to run the scenario associated to the "file in focus".
- Press the  button ... the scenario will start

- While the scenario runs you will see the log file progressing with messages



```

26
27 log-arg:
28 [echo] 28/10/2015 05:56:11 HSCT:log-arg:in
29 [echo] *****
30 [echo] * basedir = F:\Gnd\DocuDesign\Dita\Docu\MyDita\src\..
31 [echo] * dita.dir = F:\Dita-OT2
32 [echo] * transtype = custpdf
33 [echo] * tempdir = F:\Gnd\DocuDesign\Dita\Docu\MyDita\src\../build/temp
34 [echo] * outputdir = F:\Gnd\DocuDesign\Dita\Docu\MyDita\src\../pdf
35 [echo] * clean.temp = yes
36 [echo] * DITA-OT version = 2.1.1
37 [echo] * XML parser = Xerces
  
```

Figure 16: Progress log during oxygen-scenario

- At the end of the process the window switches to the result window indicating the possible warnings or errors

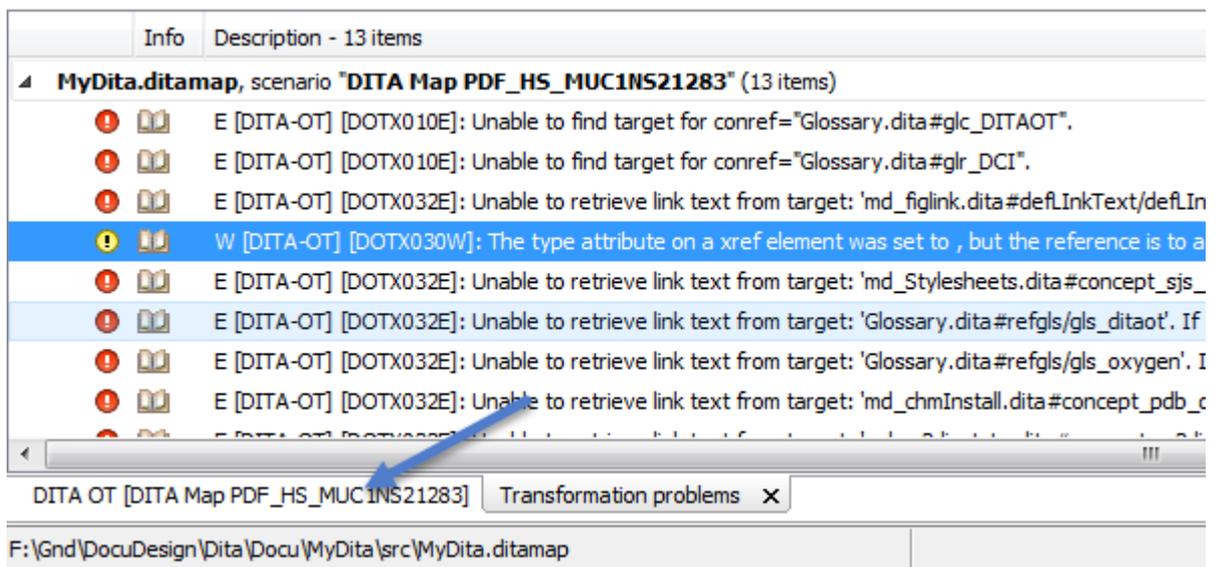


Figure 17: Result window

- Solve the errors, except for the [DOTX032E] which simply indicates that you have empty `xref` topics which is intentional if you want to feed the `xref` description from the target's content (e.g. chapter title).

For harder problems you might want to see the log files. Hence you need to select the tab with the log file message as indicated in [Figure 2](#)

You can also save the content of the log window, using the right mouse key to get to the associated context panel.

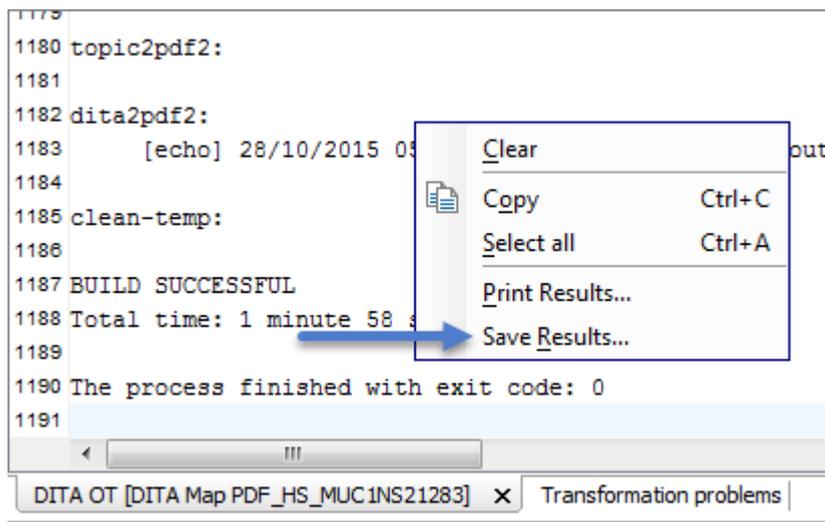


Figure 18: Log window context dialog after right mouse click

- Use **Save Results** in order to save the log to a file for further investigation.

- 
Important: I highly recommend to use the command line interface if you have problems to fix. See [2.2 Running DITA with command line](#) to learn how to launch from command line.
- 
Attention: You cannot use the command line if your potential problem is with the configuration of the oxygen parameters (see [1.4 oxygen configuration](#)). Using the command line does not use the oxygen parameters, hence you wouldn't be able to debug them.

2.5 How DITA is processed...

The XML DITA file will be processed using a stylesheet.

The stylesheet has to be created manually. The first draft is delivered with the DITA-OT (Dita Open Toolkit)., However, if you want more - it's work.

Processing ... The **DITA-OT** will use the stylesheet and process the DITAMAP (all files in it).

The result is the `topic.fo`.

Making PDF Then the **Antenna House formatter** will produce the PDF from the `topic.fo`.

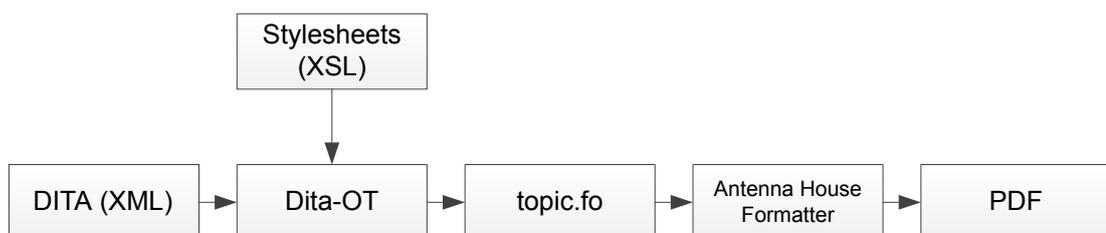


Figure 19: Dita Process

Here is a list other list

2.6 PDF post processing

Unfortunately there are several drawbacks in the PDF production that are currently unsolved by the tool providers

Antenna House Formatter

The Antenna House **Formatter** will generate URL links if you supply it a hard coded target address for a link.

```
<xref href="../../../pdf/MyDita.pdf" format="pdf">Formatter</xref>
```

will resolve correctly into a direct file link to a PDF file using a "Launch" directive. This is controlled by a variable in the [AHF#use-launch] variable in the AHF options file.

If, however you give it a hard coded address

```
<xref href="file:/Z:/Work/MyDita.pdf" format="pdf">Formatter</xref>
```

then the AHF formatter will create a URL-type link which will resolve the link by opening your **Default Internet Browser** which of course is a most annoying situation.



Note: Although oxygen warns you by an appropriate

message  W [REF] Absolute references are not portable and should be avoided: "file:/ you might not be able to avoid that situation because

- a.) it is quite annoying to find out the relative address of your target
- b.) if the target is on another (e.g. network-)drive, then you have no chance to create a relative address

Links from graphics to internal chapters

One of the great things we can do is to create links in a VISIO graphic, export as SVG and the links will be still contained in the final PDF. However, these links do not actually "know" their target because during processing the DITA-OT changes the names of the target with a prefix (e.g. "unique_7_connect_42_<name>") in order to avoid ambiguous targets.

The Acrobat function **ND-xtreme** → **Links** → **DITA post processing** will repair these links and therefore shall be applied on the final PDF.



Important: You need to have installed the ezRead Plugin on your Adobe Acrobat installation → [ezRead#1.1]

3 Important Topics

This chapter explains the most important topics and their attributes for the use with DITA

Topics	3.1 Using figures	35
	3.2 Basic figure aspects	35
	3.3 Working with the Glossary	37
	3.4 Working with equations	38

3.1 Using figures

How figures are used and the most important variants

Figures are important for anything. The most discussed aspect of figures are

- How to place/wrap text above/below/around figures
- How to overlay a figure with text and links in this text (Helmut's special)

The answers will be available in the following chapters

The following is an example of a figure



Figure 20: Test figure

3.2 Basic figure aspects

What is common in all figures

A basic figure is made of the following tag (in that order)

- `<fig>`
- `<title>`
- `<image>`

fig is the container for the `title` and the `image`

title is the figure caption (some say 'title') and whether it is printed on top or bottom of the figure is determined by the stylesheet - you don't need to care

image is the actual JPG/PNG/EPS/SVG image. It contains an `href` attribute that links to the target picture (PNG/JPG) and hence it does not allow text content.

The image topic has a lot of attribute, the most important attributes are

- `width`
- `height`
- `placement` (inline | break)
- `scale`
- `align`
- `scalefit`
- `expand`



Figure 21: Test figure

3.2.1 image-width

Explains the width attribute in an image

The width attribute determines the width of the image. Either the width or the height attribute shall be specified to maintain the aspect ratio of a figure.

...

...

The width attribute is described in [DitaSpec#3.1.1.2.16]

3.2.2 Wrap text around figures

How to wrap text around figures

Figures typically may appear in the text flow as single blocks. However, good text allows to insert figures with the text flow i.e. the text wraps around the image.



The figure on the left side figure is placed within a table. This text is written in columnt 2 of the table and hence it will have a look-and-feel like it was wrapping around the figure.

The potential layout power is not to what you can do with professional layout programs, however, for technical writing the table approach is likely to give you satisfyable results

This is solved by using <try, test, describe>



Tip: Having thought about it ... using a hidden table seems to be the best approach since there you have most text control. With many prior editorials, the "wrap around" can sometimes look quite ugly.

Using a table, at least gives you some better WYSIWYG feeling.



Notice: OK - I admit. that's not really a solution but a workaround.

3.3 Working with the Glossary

The glossary entries shall be addressed by a keyref.

An entry Helmut write can just be refered by e.g. gls/gls_DCI.



Important: You shall, however, assign an id to the glossentry, even if you acutally refer to the glossterm..

The ID's in the glossary shall be standardize

glossentry	gle_<name>
glossterm	gls_<name>
ph	glr_<name>

where `<ph>` is the element that should surround an expression that we want to catch with a `conkeyref` link.



Note: Do not use figures (`fig`) in the glossary. You may well use images using the `image` topic, however a *figure* will cause the DITA-OT to forget the right margin and the following text will exceed the right page margin. This can be fixed in the future, however, as of 19Oct15 there is not fix available.

3.3.1 Creating a local glossary from a master list

Using master list is a powerful idea since you can maintain one single large glossary which holds hundreds of entries.

If you create a document you will simply add references to any glossary entry. Then you apply the `glsSelectAll.xsl` stylesheet to any of your chapters and this stylesheet will create you a `REF_gls_Local.dita` file in your source (`src`) path that can be copied over your existing glossary or added to the DITAMAP as 'the' glossary.

The created glossary only consists of those entries that you have referenced in your (entire) document.



Attention: The stylesheet does not respect the `ditaval` technology i.e. if you excluded documents or chapters, the stylesheet will find those excluded file's entries nevertheless.



Note: Technically the `glsSelectAll.xsl` parses every file that exists in the same directory as the file to which the `glsSelectAll.xsl` is applied to. So it is good advice to move those files that you did not use in your ditamap.



Note: Future enhancement will parse the DITAMAP to identify exactly those files that are relevant for the glossary. An evaluation of the `audience` may be done and the actual glossary entry to be created can receive the same audience if it is called uniquely. If it has callers with different audience, then it might apply some default whether to take all audiences or none.

3.4 Working with equations

The easiest way to include high quality formulas is to export them after editing into SVG format.

$$\sqrt[3]{(a + b)}$$

Figure 22: Testing a formula (export as SVG)

The formula itself was created with MathMagic™ and then exported as SVG.

Advantages

Using [SVG](#) allows adding links in the formula just as described in the

4 DITA-OT extensions

The following chapters explain the use of extension to the DITA-OT.

Topics	4.1 Extensions on the paragraph.	39
	4.2 Extensions on section	40
	4.3 oxygen Annotations	41
	4.4 Extensions to tables	42
	4.5 Extensions to fig/image	43
	4.6 Extension to links	52
	4.7 Extensions to Notes	56
	4.8 Extensions to lists	57
	4.9 Extensions to Mini-Toc	58
	4.10 New page	59
	4.11 Keep Lines together	60
	4.12 Extension on the title element	60

4.1 Extensions on the paragraph.

Several extensions are on the paragraph

```
p:outputclass='mrg | :right | :dialog | :heading'
p:outputclass='keep'
p:outputclass='compact' for no-spacing
image:outputclass='valign=top' for marginalia
```

Marginalia will put out the "mrg" paragraph to the marginalia flow.

mrg:right will right align the marginalia. The space between the marginalia right boundary and the text flow's left boundary is specified in

```
<xsl:variable name="mmMarginaliaGap">4</xsl:variable>
```

in the `basic-settings.xml`.



mrg:dialog

will produce a dialog box and a heading in the text flow. This can be attractive if you need to explain software dialogs and you don't want to make every dialog a separate section.

In general, your text flow require as much vertical space as the left margin requires. Otherwise the next marginalia item might come later than your associated text flow as it cannot overlay the previous marginalia content. You can always achieve this with empty paragraphs.

mrg:heading

will produce header style. Here the marginalia can be used to emphasize or point out an expression. The feature invites to use it for definition lists, which is indeed possible, however, not recommended. The author should put definition lists in `dlentry` tags in order to maintain the correct markup for proper reuse of the DITA philosophy.

Nevertheless, this features is not a definition list and can be quite helpful for some typical authoring situations.



Of course an image can be placed in the marginalia section. Typically, adjacent text will be bottom-aligned.

However if we set the paragraphs's `outputclass` to `valign=top` then the image will be top-aligned with the adjacent marginalia text.

It depends on the actual context which of the choices is more appropriate.



Left to here is a text with a marginalia containing an image and the paragraph's `p:outputclass:valign = top`. Of course the image shall be rather small and you will use it together with text only, if your `side-col-width` is large enough to make *image+text* still look pretty enough for the reader.

To get text aligned, in all cases the image shall have the default `placement=inline` i.e. you don't need to do anything if this attribute is not explicitly set to `placement=break`

Paragraph with `outputclass=keep` will keep together any consecutive elements to be joined on a page.



Remember: This is subject to an exercise to challenge that mechanism.



Note: The remember type is described in [\[DitaSpec#note\]](#).

4.2 Extensions on section

The extension of section mostly consider the layout. Coming from the conversion of the TC-Toolbox there is some legacy for the `outputclass` definition.

Default section layout

Possible outputclasses are

- **mrg** `.[anything]`, containing `Block`. will print title in left margin as marginalia



Note: If the body part is too small, the section title text will spill into the next paragraph. As such a situation is bad style anyway, it doesn't need to be corrected technically.

- **flow**: prints a separator and title in the text flow
- **page**: prints separator and title spanning over the page

Section with outputclass = mrg

This is the first paragraph in the section, The text flow starts with the section because the section's title is entirely positioned in the marginalia flow.

It is obvious that `outputclass=mrg` is practical only if you have short titles.

Section with outputclass = flow

This is the first paragraph in the section. The title and separator start in the text flow area only.

The `flow` option is very practical if you need to express kind of a sub-section that shall not appear as totally separate new section.

Section with outputclass = page

This is the first paragraph in the section, same as default - title and separator line go over the entire page

4.3 oxygen Annotations

oxygen supports **markup** that allows intermediate formatting and change history

- **Insertions** **will be indicated by green color font on background grey** content
- **Deletions** **will be indicated by red strike-through** content
- For insertions and deletions associated **track bars** will be visible on the side.
- Our stylesheet allows to *see the revised content* in the printout.
- The **editor colors** can be defined in oxygen differently for every contributing author. That specification has no impact on the output formatting (which is done in the stylesheet). Use the **Preferences** → **Editor** → **Edit modes** → **Author-Review** settings.

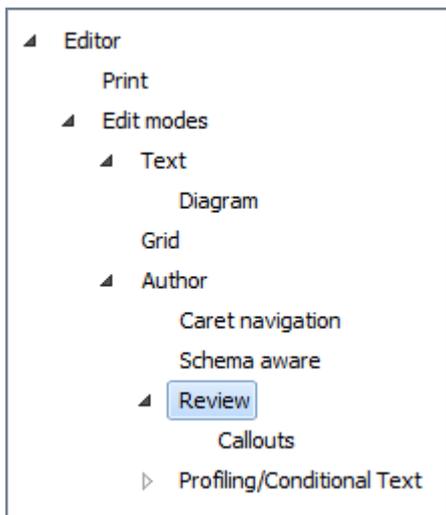


Figure 23: Preview references in oxygen

- The stylesheet, however, allows to switch off the interpretation of tracking information. Nevertheless those changes require re-processing of the document. You cannot remove the markup in the final PDF (except for the comments).
- In order to remove the tracking information,

There are other markups supported in oxygen. The most important is the **background color markup**, which allows **different colors** to be used to **highlight** text.



Another important feature are the **comments**. They compare to actual comments in the PDF and they may be actually managed in the final PDF.



A special trick is to place a special @-character where you want to insert a comment that does not surround text



Warning: One thing you cannot do you cannot span insertion or markup over the end of a paragraph or any other tag. There you need to apply the markup separately until the topic's end and restart from the next topic on.

4.4 Extensions to tables

Several extensions were made to tables

- **Auto span calculation:** If the width of a table exceeds the *text flow width*, then the table is automatically placed with `pgwide=1`
- **Header repetition** → set `TitlePosition=table_titleRepeat`
- **row:outputclass=compact** for (differentiates whether `rowsep = 0` or `1`)
- **p:outputclass=compact** for table paragraphs
- **table:outputclass:rowcolor=#1280FF | yellow | red ...** specifies rowcolor for the entire table body.
- **entry:outputclass=left:<distance>** e.g. `left:0pt` to enforce the distance from the left within the cell. Can be most helpful when using tables as a hidden layout background.
- Variable 'Table-backgroundRow' for **default table background** color (e.g. `antiquewhite`) or `#E0E0CC`
- **Table Title on top/bottom**
- **Title EnumerationMode #** or `<chapter>` - #allows table number prefix and renumber on every chapter

Table 2: Example using compressed rows

No.	Sender		Receiver
	Get Random	→	Receive Data Compute random number R = RNG(1024)
	Get Result	←	Return result
	row:outputclass with rowcolor=yellow		
	row:outputclass rowcolor=#1280FF		

4.4.1 Repeat table header

An important extension was the repetition of the table title on every page break. The actual implementation can be controlled through the plug-ins variable in `.../cfg/fo/xsl/basic-settings.xml`:

```
TitlePosition=table_titleRepeat
```

which also controls through the variable `table_titleBelow` that the table title should be printed below the table. The technical standards do most use the title *above* the table, which is the default in the plugin.

 **Notice:** Technically the solution is obvious, the table title must be coded to become 'hidden' header row. As the XSL-FO Formatter can only repeat `thead/rows`, the title is processed in such a row with the rule settings such that the reader will not recognize the title to be part of a table row.

Here is a table, it repeats the table title on every following page.

Table 3: Row colors

Sender		Receiver
SELECT FILE	→	
SELECTX FILE	→	
	←	Status OK
GET CHALLENGE		
yellow row		
Page overflow We create a page over flow by a table with as many rows that it cannot fit into one page		
Page overflow We create a page over flow by a table with as many rows that it cannot fit into one page		
Page overflow We create a page over flow by a table with as many rows that it cannot fit into one page		

4.5 Extensions to fig/image

4.5.1 Creating Figures From Visio

A great thing is, that you can create vector graphics with Microsoft-Visio and apply links to figures which will be maintained until the final PDF if you **export the image as SVG**.



Tip: If you don't use MS-Visio, I recommend the free and powerful **IncScape** SVG editor.

However, to link to a chapter in DITA you have to follow the following rules

- You link should use the `id` of your target **preceded by a hash #** e.g. `#visionotes` whereas the `id` of a corresponding `title` would be `id=visionotes`.
- You cannot refer to a link of a Head1 chapter (e.g. `concept:id="head1ch"` because the DITA-OT will replace all `id`'s being assigned to Head1 chapters. However, you can (and consequently 'should') refer to the Head1 chapter's **title**. So you shall give the `title` the `id` and refer to that.
- In order to activate the link, you need to perform post-processing on the final PDF [2.6 PDF post processing](#) page 33

The exported (Visio-)SVG file will not work for an absolute file path unless it is given as URL

- **Absolute links to files** shall be given in the URL notation e.g. `file:///C:/ProgramData/ezRead/Documentation/dev/ref/stb/ISO4/ISO4_Ch5.3.stb`. This is in particular important to refer to stubs → [\[ezRead#7.1.6\]](#)

The **margins of the final Visio figure** should be set to zero. This is done with the **Setup** button in the Print Setup (or press Shift-F5)

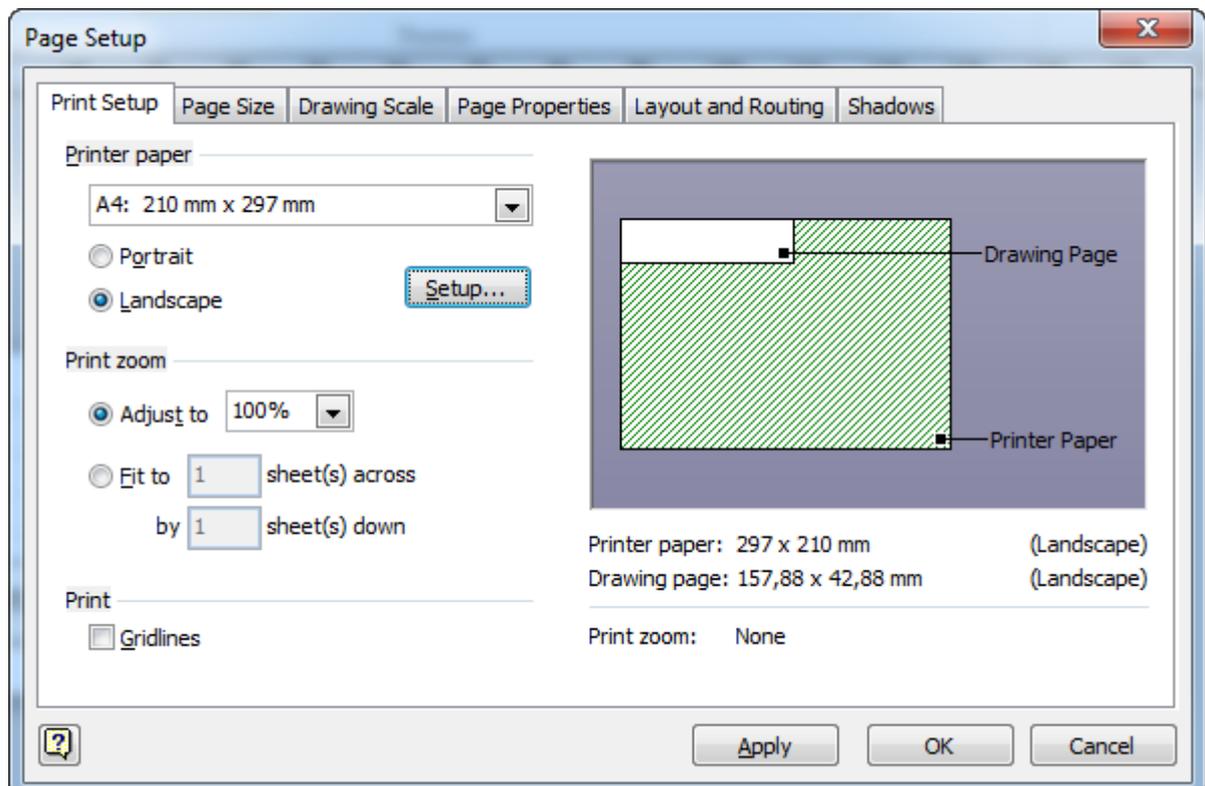


Figure 24: Selecting Margins

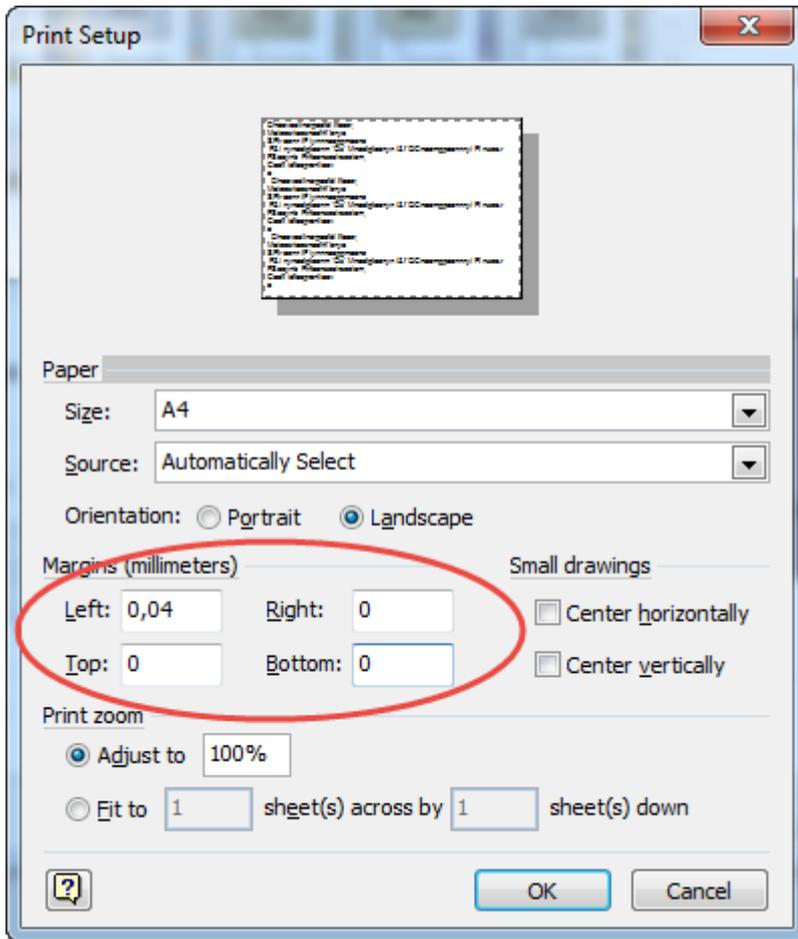


Figure 25: Setting margins

and

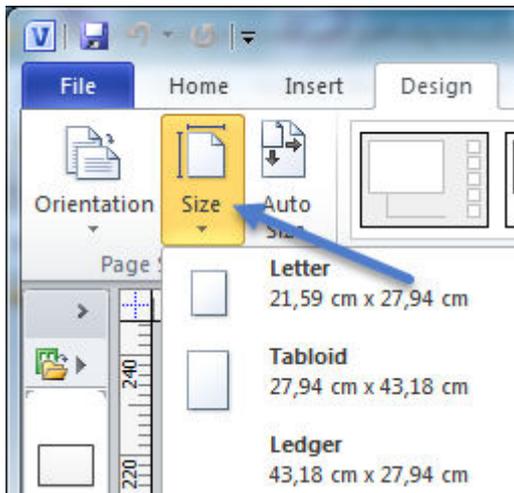


Figure 26: Set Page width

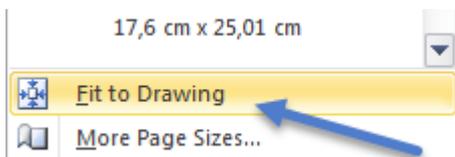


Figure 27: Set "Fit to Drawing"

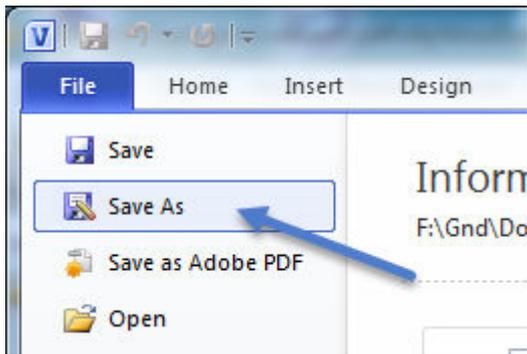


Figure 28: Set "Fit to Drawing"

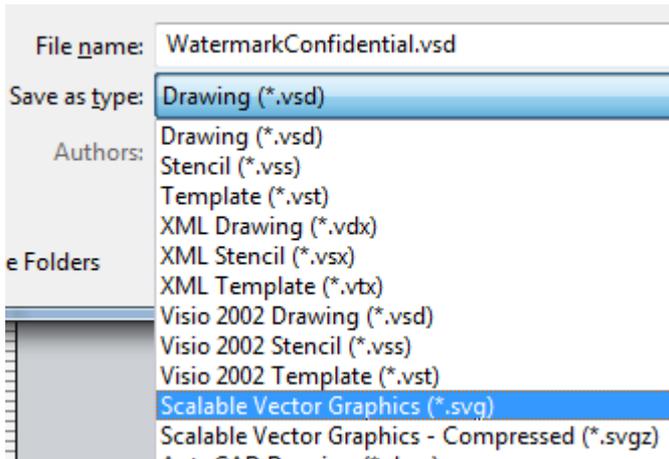


Figure 29: Set "Fit to Drawing"



Warning: Never **Save As** when you have objects selected. First use ESC to deselet any object. If objects are selected, only these objects will be saved and the dimensions of the target are the dimensions of the object, not that of a page.

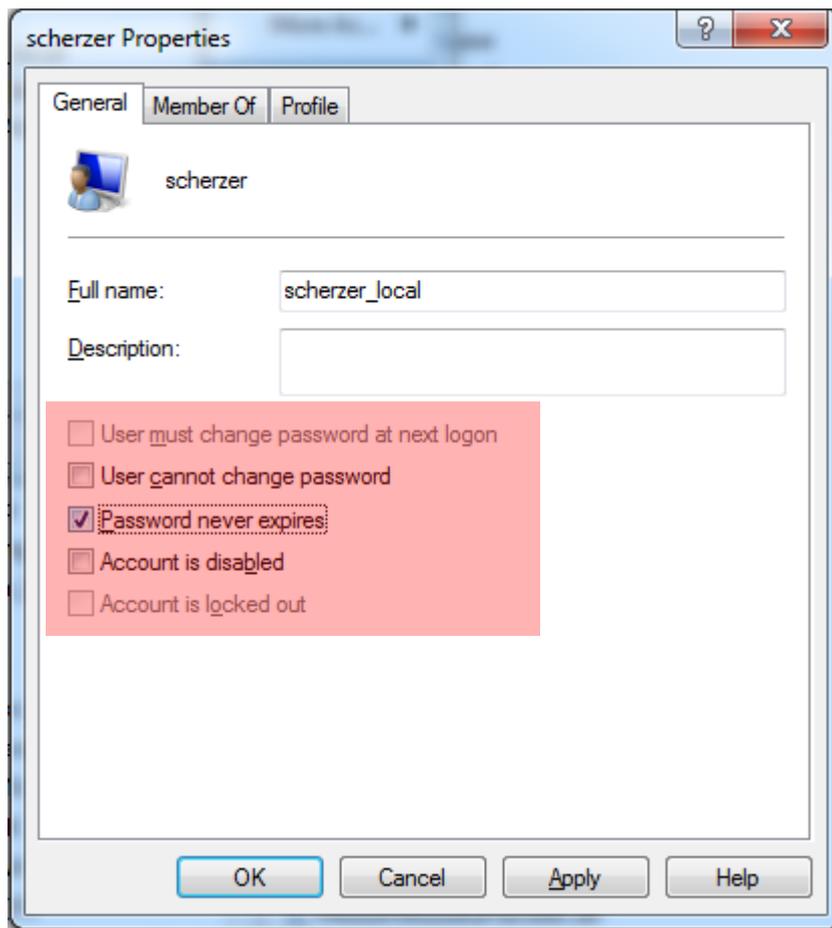
4.5.2 Links within graphics

The `imagemap` topic [\[DitaSpec#imagemap\]](#) is available to **place hyperlinks** on graphics. A simple `imagemap` consists of an `area` specifying

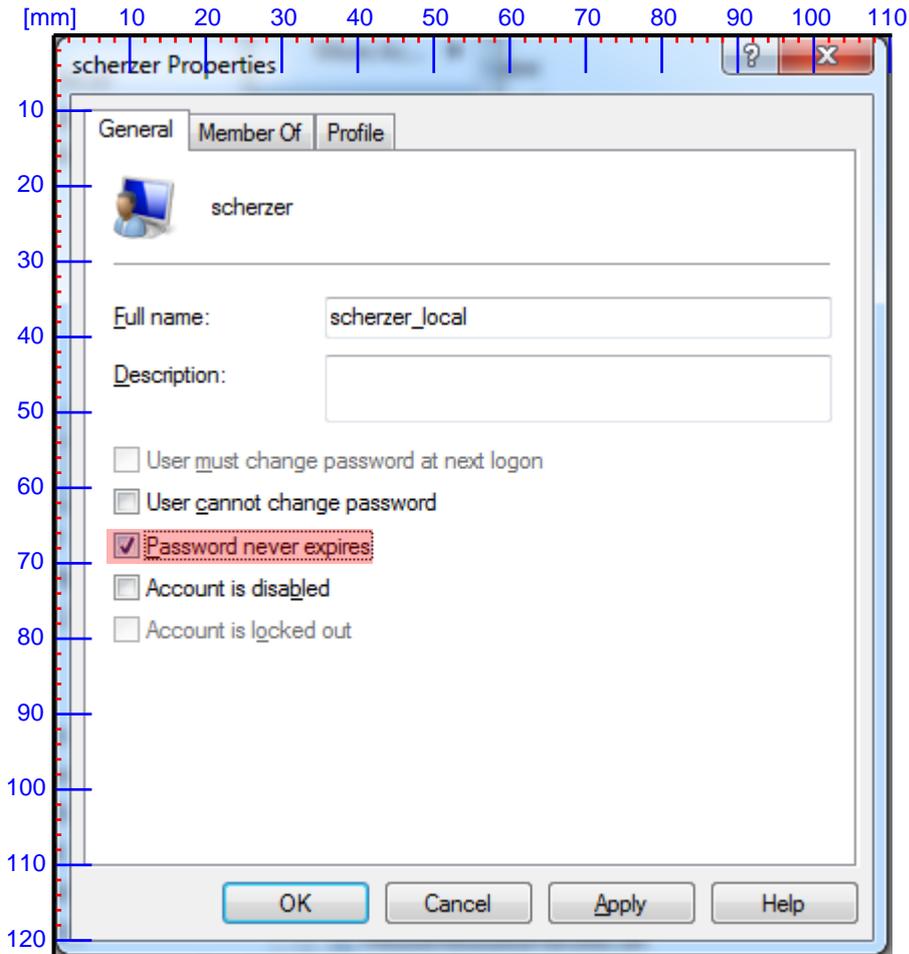
- **shape:** type of the shape, in most cases 'rect' is the most wanted shape type
- **coords:** The left-top-width-height coordinates in the regular units (mm, in, pt, px, cm)
- **xref:** the link URL to the associated target



Important: The `imagemap` cannot be child of a `fig`, however, there is a workaround if you place a paragraph `p` in a `fig` element and under the `p` you may apply the `imagemap`.



During the design-phase you can use some very helpful `@outputclass` attributes



Using the attribute `imagemap:outputclass=scale:mm` creates a coordinate system measured in *mm*. This is most helpful to quickly determine the fields of the area section. Allowed values for the unit are

- mm
- cm
- px
- pt

otherwise the system defaults to millimeters.

Using the attribute `area:outputclass=show` will show the *link area* to allow better placement

The above example uses two areas containing a link.

- **Area 1:** left-top-width-height = 4mm, 10mm, 14mm, 5mm - the "General" tab = **invisible** because `outputclass` is not set
- **Area 2:** left-top-width-height = 7mm, 65.5mm, 35mm, 4.5mm - **visible** because `outputclass=show`

```
<imagemap id="demo_imgmap" outputclass="scale:mm">
  <image href=" ../gfx/lusrmgr03.png" id="image_nft_sbr_hs"/>

  <area>
    <shape>rect</shape>
    <coords>4mm, 10mm, 14mm, 5mm</coords>
```

```

    <xref href="md_ezReadLink.dita#ezReadLink"/>
  </area>

  <area outputclass="show">
    <shape>rect</shape>
    <coords>5mm,52mm,65mm,31mm</coords>
    <xref keyref="lkfigs/linkfigs"/>
  </area>

</imagemap>

```



Note: Unfortunately on fully qualified (absolute) paths the AHF generated links are an URL link (`file:///...`) which would open your browser PDF plugin → [\[AHF#use-launch\]](#). This can be corrected in the final PDF using [\[ezRead#12.1.20.1\]](#)

4.5.3 Tryout figures

This chapter presents several figures with the attributes available in the extended DITA-OT toolkit.

The first picture is simply a plain figure with no further attributes i.e. the default position. The stylesheet tries to derive the frame width from the image width, however, if the image width is not specified, then that mechanism cannot work and the frame will be set to the page margins.

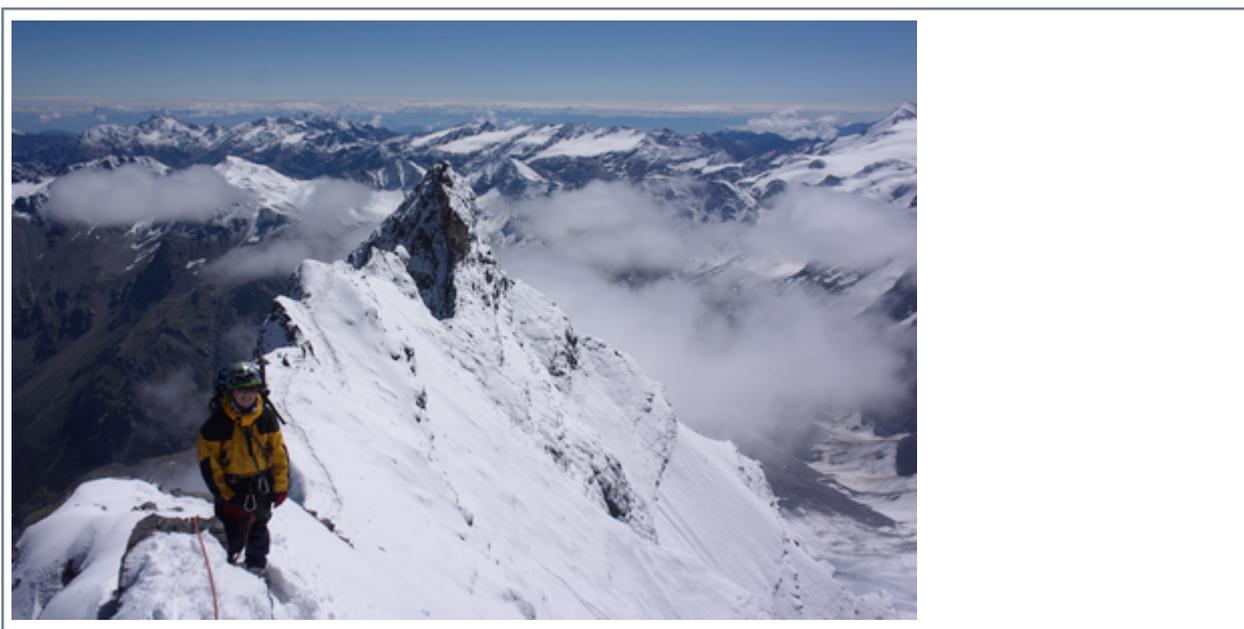


Figure 30: Plain figure - no width specified, the frame cannot be determined from the image size

The next figure specifies a width which allows the stylesheet to put the frame around the actual width

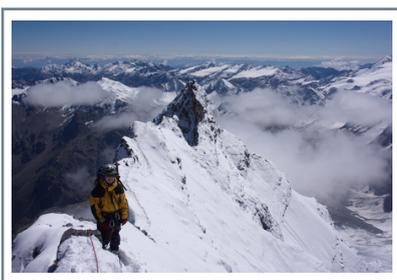


Figure 31: Plain figure, image:width=50mm

The figure below expands to the entire page



Figure 32: fig:expanse=page, image:placement=break, width=50mm (ignored by expanse=page)

The next figure aligns right, "expanse=column" fixes the frame to the page whereas the image will move according to the image:align attribute



Figure 33: fig:expanse=column, image:align=right, width=50mm, placement=break

The outputclass=page currently does not buy any more than if there was no output class

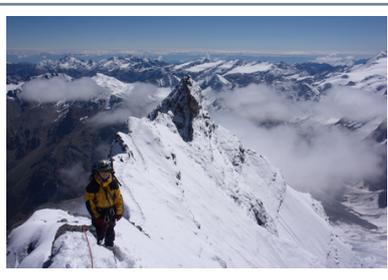


Figure 34: image:outputclass=page, placement=break

outputclass = flow is important to expand the image to the present text flow



Figure 35: image:outputclass=flow, align=right, width=50mm, placement=break

Left alignment will take place in the text flow



Figure 36: image:width=50mm, align=left, placement=break

Right alignment in the figure below works because we have placement=break.

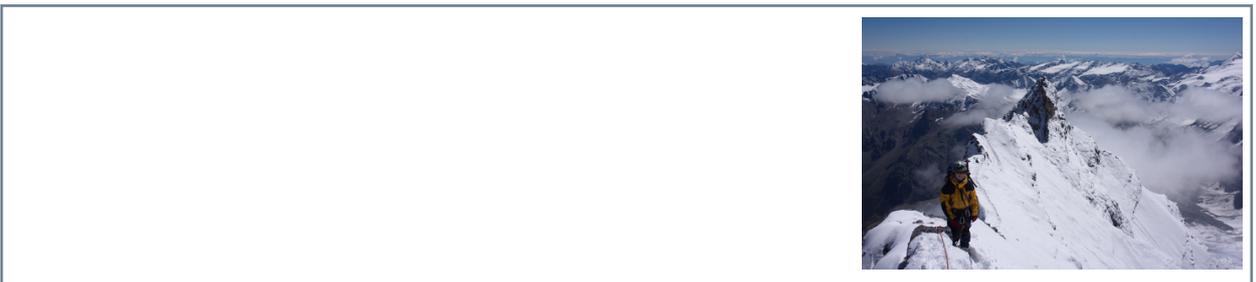


Figure 37: image:align=right, width=50mm, placement=break

Right alignment does not work if we have placement=inline because for an inline image such thing does not exist, although for a figure it would be valid, because a figure always literally implies a placement=break, but not technically.

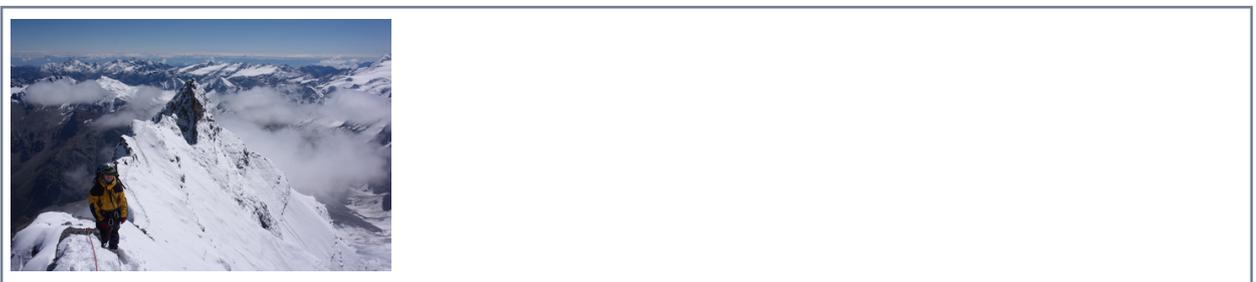


Figure 38: image:align=right, width=50mm, but placement=inline (inline does not obey 'align')

Links in figures

A specific formatter can be set.

Here is an [image](#) with links [\[ISO4#5.3\]](#) to different [targets](#)

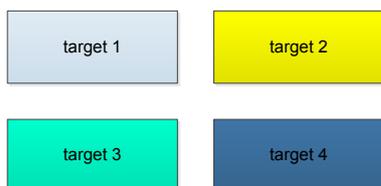


Figure 39: Image with #hstarget1..4

Target 1 is a table



Note: Only links to Head1 chapters do not work because DITA would totally replace the Head1 ID's by `unique_n` IDs. Shall be fixed later. Neither could you link to a Head1 title, but that I could fix meanwhile in the stylesheet. For any Head N>1 chapter, you can link to the header and to the title and both will work.

Table 4: Target 1

Head A	Head B
e1	e2
e3	e4

Section target

Any text

4.6 Extension to links

Several outputclasses have made the link content more powerful. The `outputclass` can contain

- `see`
- `num`
- `chp`
- `onpage`
- `pageonly`
- `pagenumonly`
- `label`
- `noheading`

A quick look can be taken in [4.6.2 Trying out links](#)

The explanation is

see

adds a "(see)" prior to the actual cross reference text and puts the statement in parenthesis

num

prints the chapter number (e.g. 4.1.7)

chp

prints chapter label and number (e.g. "Chapter 4.1.7")

label

prints only the label e.g. of a figure ("Figure 14") or table ("Table 42") in contrast to the entire caption

onpage

adds "on page #" after the cross reference text

page

adds "page #" after the cross reference text

pagenumonly

just prints the page number as cross reference

The actual use of these tags follows a syntax, powerful enough to allow the author rich combinations. The basic syntax is

```
outputclass = <text> [see]
<text> [num] <text> ...

[num|chp]
<tst> [onpage|pagenum|pagenameonly] <text>
```

You can however omit any of the <text> or [...] expression, whereas the order cannot be changed. As [chp] includes the chapter number, there is no reasonable use for both of them.

The idea to place text between any of the [...] macros allows personalized layout of reference information.



Note: These tags only work for empty XREF statements i.e. if you have text content in your XREF statement, then this text content will be the only text being shown as 'clickable'.

4.6.1 Linking figures and tables

A special `outputclass=[label]` tag is available for *figures* and *tables*

outputclass = [label]

Link to figure [Figure 24](#)

which only shows the figure/table label ("Figure"/"Table") and its number if the `xref` statement is empty.. This is often used.



Attention: Since DITA-OT version 2.1.2, this is also the default. so the `outputclass = [label]` is not required, however there is another use described below.

Special use

As an empty `xref` automatically generates a `<label> <number>` text the [label] macro can still be used, if more than the label shall be displayed. The combination `outputclass=[label] <text> [title]`

will yield the label (e.g. "Figure 4.2") followed by some optional user text and the caption e.g. "Modifying stylesheets". This is a flexible method to circumvent the DITA-OT default and the only way to get the figure caption out unless you are using configuration parameters.

DITA-OT default

The DITA-OT default parameters of course can get quite close to this approach and are described in [Chapter 2.3.1.1](#) The present method allow you to change that default within the document whereas the parameters will always be valid for the entire document (=default) unless you override by the `outputclass` method.

Referencing fig or title?

Technically the implementation allows you to refer to the `id` of a `fig` element or the figure's `title` element (same applies to tables). However see the difference here:

- Reference to [fig-topic](#)
- Reference to [Title](#)
- Reference to empty [Selecting Margins](#)

The reference to the title will align the title (at the bottom of the figure) to the top line of the PDF viewer (e.g. Acrobat). Linking to the figure will show the figure itself which is certainly what the reader wants.

Recommendation: Reference the `fig` topic in order to get proper placement of figures when referencing them in PDF

4.6.2 Trying out links

This chapter shows examples for links.

Using empty xrefs

Our experiments linking concept

outputclass = Find more in [num]	test Find more in 4.6
outputclass = Find more in [chp] "[title]"	test Find more in Chapter 4.6 "Extension to links
outputclass = Find more in [chp] "[title]" [onpage] ff.	test Find more in Chapter 4.6 "Extension to links" on page 52
outputclass = [see] in [num]	test Find more in (see 4.6)
outputclass = Find more in [see] [chp]	test Find more in (see Chapter 4.6)
outputclass = Find more in [see] [chp] "[title]" on the page [pagenumonly] of many	test Find more in (see Chapter 4.6 "Extension to links" on the page 52)
outputclass = [num]	test 4.6
outputclass = [chp] "[title]"	test Chapter 4.6 "Extension to links

Feel free to play with other combinations

Refer to [\[DitaSpec#3.1.1.2.24\]](#) which [\[DitaSpec#note\]](#) is the note ...

Empty `xref` with `outputclass=<empty>`. Linking table [Target 1 link](#)

Empty `xref` with `outputclass='label'`. Linking table [Table 3 link](#)

Empty xref with outputclass='label'. Linking figure [Figure 1 link](#)

Empty xref with outputclass='see'. Linking figure [Figure 1 link](#)

Text xref with outputclass='see'. Linking chapter [MyText link](#)

Text xref with outputclass='onpage'. Linking chapter [MyText link](#)

Text xref with outputclass='pagenumonly'. Linking chapter [MyText link](#)

Text xref with outputclass='see page'. Linking chapter [MyText link](#)

Text xref with outputclass='see onpage noheading'. Linking chapter [link](#)



Note: Empty xrefs will produce a warning *[DITA-OT] [DOTX032E]* indicating that your xref is empty. As this is intentional in order to feed the xref-text from the target title, this warning shall be ignored.

Table 5: Table in note

Header 1	Header 2
<p>here is test</p> <p>this is a paragraph next para with compact</p> <p>next para next para with compact</p>	

- Table in List

Table 6: Table in list

Header 1	Header 2
<p>here is test</p> <p>this is a paragraph next para with compact</p> <p>next para next para with compact</p>	

Table 7: Table in normal text flow

Header 1	Header 2
<p>here is test</p> <p>this is a paragraph next para with compact</p> <p>next para next para with compact</p>	

Indexterms

Indexterms on this page = security

4.6.3 ezRead auto-linking

Using ezRead links as described in [\[ezRead#9.3.3\]](#) and [6.1.2 Referencing external documents \(Bibliography\)](#) will create the appropriate links to chapters or even paragraphs of external PDF documents without entering `XREF` topics. The plugin detects the `[...#...]` construct and assigns an associated link to around the `[...#...]` entry that matches the chapter of the referenced document.

To make those links work you need to prepare the target document according to [\[ezRead#7.1.6\]](#) which describes the Stub-Technique. If stubs are available, the links will work.



Note: As a matter of fact, the links are created regardless from whether stubs are available or not. This means you can create the associated stubs even later.



Tip: Also read [\[ezRead#7.2.1\]](#) which explains how you export a linked files tree in order to remove stub references when you want to deliver a tree of linked documents

The auto-linking feature is quite a powerful feature since you do not need to care for any link going outside your document.

4.7 Extensions to Notes

The `note` topic already brings a powerful set of types, each related to a specific context. The available types are described in [\[DitaSpec#3.1.1.2.24\]](#)

```
note | tip | fastpath | restriction | important |  
remember | attention | caution | notice | danger | warning | other
```

Several icons have been chosen to express the note context



Note: This is note of type `note`



Tip: This is note of type `tip`



Fastpath: This is note of type `fastpath`



Restriction: This is note of type `restriction`



Important: This is note of type `important`



Remember: This is note of type `remember`



Attention: This is note of type `attention`



Caution: This is note of type `caution`



Notice: This is note of type `notice`



Danger: This is note of type `danger`



Warning: This is note of type `warning`



othertype: This is note of type `other`

4.8 Extensions to lists

The most important extension is done on the list items `li` | `sli` | `dd` introducing `outputclass=compact`. Furthermore the tag icons of the unsorted list can be changed

Compact list items

The following unsorted list has set `ul:outputclass=compact:all`

- First list item - none of the list items has `outputclass` defined
- Second list item
- Third list item

so all list items - even the first - are adjacent and compact. To separate the text after the list (just where you read) you may simply start a new paragraph after the closing `ul`.

Very often, however, the list items shall have a space to the previous text, this is part of the official DITA-OT and you only need to set `ul:compact=yes`.

- First list item
- Second list item
- Third list item

li:compact

If only individual list items shall be compact, then the list itself shall not contain `compact=yes`, but the individual list item may do `li:outputclass=compact`.

- First list item - not `compact` to create space to previous text
- Second list item - `compact` to closely follow the first item. However as this list item produces lengthy text which spans over more than one line, very often you would like to keep some space between **this** list item and the **next** one. Then you can have no `outputclass` with the next list item and it will create space by default.
- Third list item - no `outputclass` defined
- Forth list item - again we set `outputclass=compact`

Unordered list icons

The unordered list can be configured with different replacements for the tag icon (typically a bullet).

Using `ul:outputclass=folder` allows folders to precede the unsorted list entry.

 Folder 1

 Folder 2

An unsorted list can also have `ul:outputclass=checklist` which results in

- First item with `ul:outputclass = checklist`
- Second item with `ul:outputclass = checklist`



Note: There are more types possible. The definition is done in the customized `plugin:lists.xsl` like

```
<xsl:when test="../@outputclass='checklist'">
  <fo:inline font-size="18pt" baseline-shift="20%">
    <xsl:call-template name="insertVariable">
      <xsl:with-param name="theVariableID"
select="'Checklist bullet'"/>
    </xsl:call-template>
  </fo:inline>
</xsl:when>
```

The `baseline-shift`-value moves the symbol up to align correctly with the text. This is a requirement you will often find if you use special characters

4.9 Extensions to Mini-Toc

The `mini-TOC` summarizes the `Head2` chapters on every `Head1` chapter. As a consequence the `Head1` chapter always consists of a content part and the `mini-TOC`.

```
concept:outputclass=tocfirst
```

will print the `mini-TOC` right **after** the `Head1` title.

Otherwise the `Head1` text will be printed **first**, followed by the `mini-TOC`.

When to use "tocfirst"

For better readability use `tocfirst` whenever the `Head1` text is large. It feels irritating for a reader if you have already started with a longer explanation in the `Head1` chapter and then a `mini-TOC` follows.

If the `Head1` text is short then it often feels more readable if the `Head1` text gives a short explanation of "what's coming" followed by the `mini-TOC`

 **Notice:** You can switch the creation of a mini-TOC by the runtime build parameter

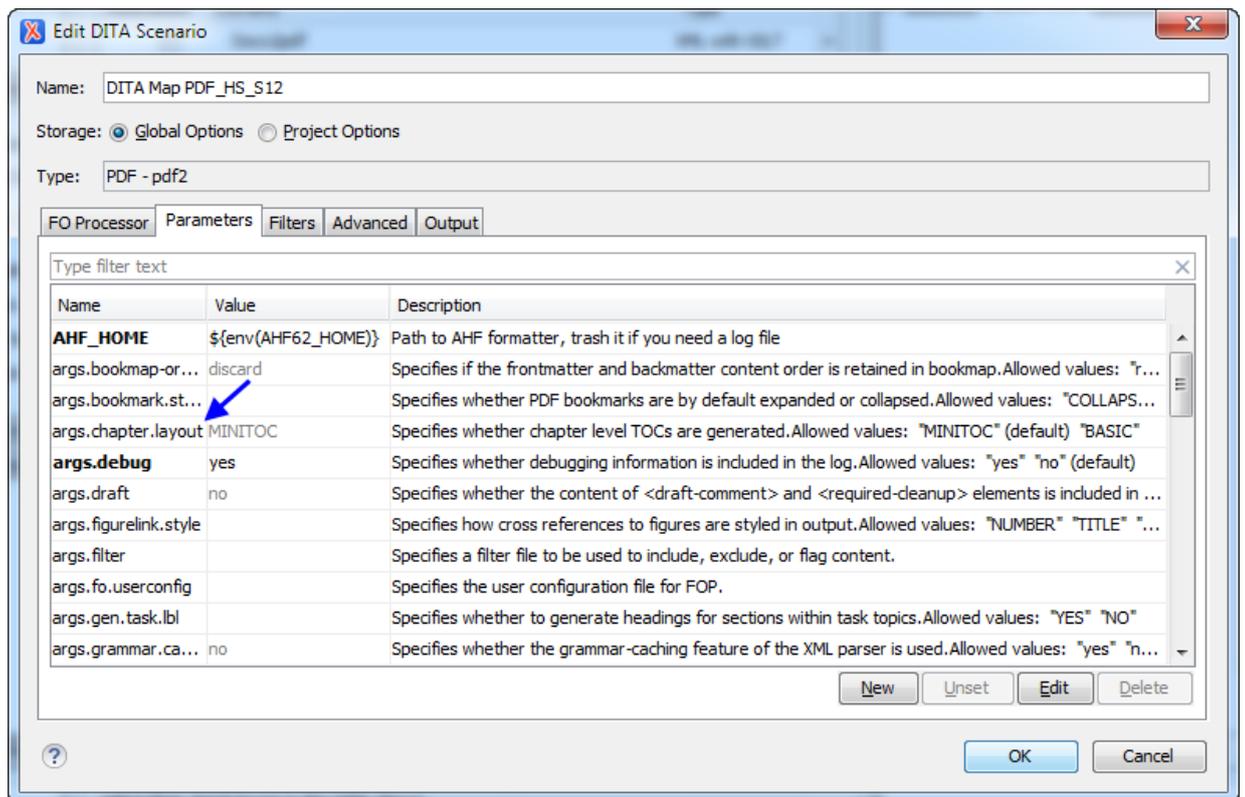


Figure 40: Configuring mini-TOC in runtime parameters

4.10 New page

This chapter starts on a new page after setting its **title** to

```
title:outputclass = newpage
```

4.11 Keep Lines together

A smart computation was created to keep - if possible - paragraphs together if they are in a certain distance from the previous title. This avoids new chapters to break right after the heading.

The System variable

```
<xsl:variable name="maxKeepLines" select="4"/>
```

in the `basic-settings.xml` determines the strength of this rule.

Technically the `maxKeepLines` variable checks the number of non-empty text blocks prior to the current text block. If the current text block has more than `maxKeepLines` non-empty text blocks, then it will no more get the `keep-with-previous` condition → [\[xslfo#keep-with-previous\]](#).

4.12 Extension on the title element

The title element has two extensions

- `outputclass = newpage`
- empty titles

`newpage`

The next chapter will start on a new page although there is still plenty of space on this page.

Empty titles

A title is mandatory for a new topic. However, the present extension supports an empty title i.e. nothing will be printed that would indicate that the following text is part of a new chapter, neither such empty chapter would increase any heading count..

Empty titles can sometimes be helpful to separate a (typically large) chapter into invisible sub-chapters. That may also be attractive for the purpose of reusing text blocks on file level.



Note: Typically the reuse of text blocks is recommended by using the `conref` element. [\[DitaSpec#3.4.2.4\]](#).

5 Managing the front page

The front page is edited in the `<mytitle>.ditamap`. It contains several fields which determine the text of the first and second page.

Topics	5.1 First page layout	61
	5.2 Company Logo	61
	5.3 Security Class	62
	5.4 Watermark	63
	5.5 Front picture	64
	5.6 Second page layout	65

5.1 First page layout

To be completed

5.2 Company Logo

The company logo shall be available in the `gfx`-folder. We highly recommend `.SVG` files in order to present the company in best available quality.

The company logo is coded in the `.DITAMAP` in `bookowner` section

```
<bookowner>
  <organization>Giesecke & Devrient
    <data name="logo" value="GdLogo.svg"/>
  </organization>
  <organization>IBM
    <data name="logo" value="IbmLogo.svg"/>
  </organization>
</bookowner>
```

The oxygen Author layout shows the section of the DITAMAP as follows

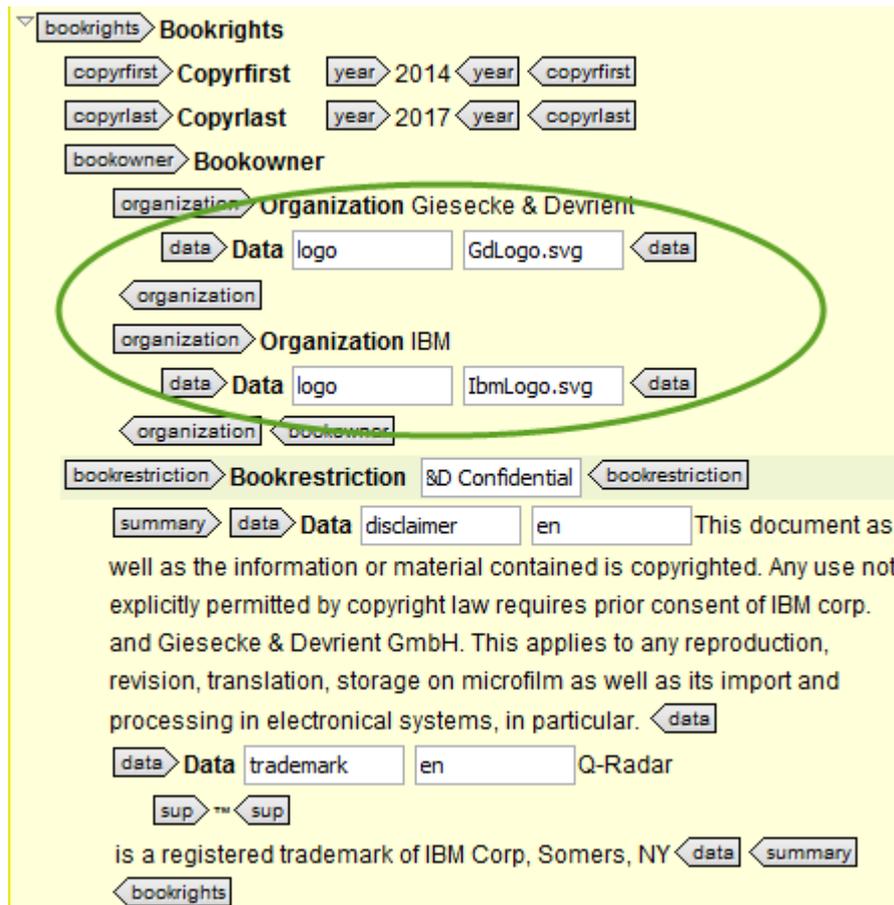


Figure 41: Front page definition with logos

A maximum of two companies is supported, as the above example shows. For one company only, delete the second `<organization>` section from the example.

Logo Position

The first company's logo will be displayed on the left upper corner and the second logo will appear on the upper right corner of the front page.

t.bd: The stylesheet specifies the dimensions in `front-matter.xsl` - could this be made user configurable (however, it is a lot of tweaking which the user doesn't really like to do)

5.3 Security Class

The security class is displayed in the footer of every odd page. Its definition is done in the DITAMAP in the `bookrestriction` section.

```
<bookrestriction value="IBM / G&D Confidential"/>
```

which appears in the oxygen-Author mode as

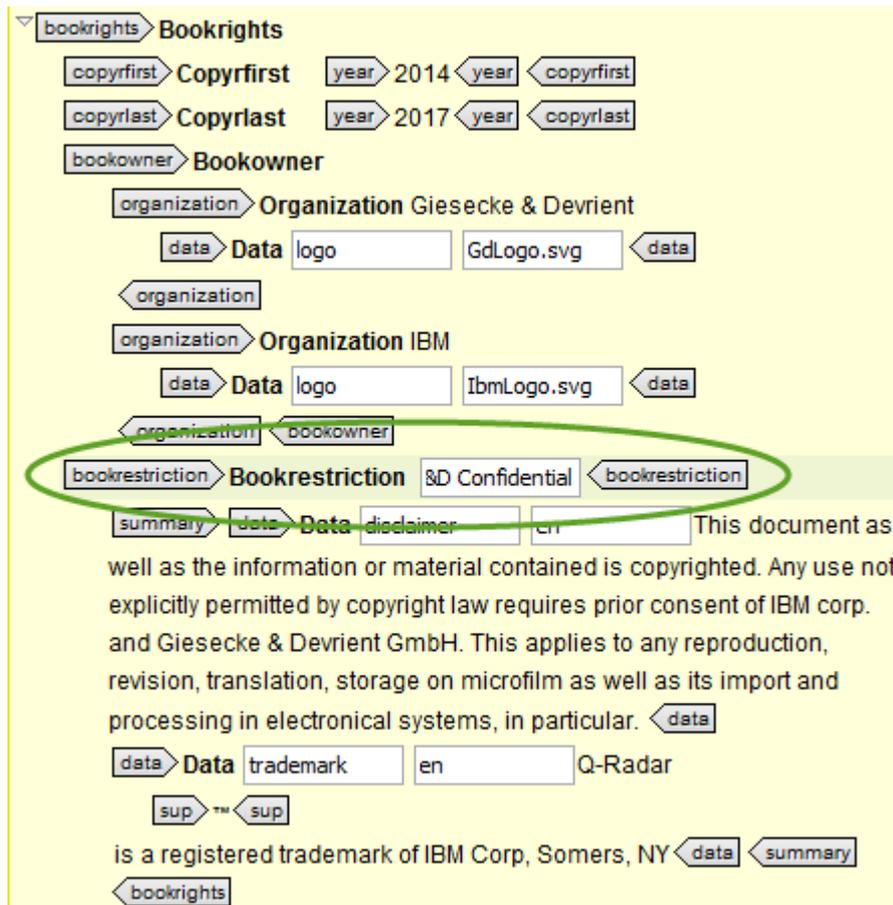


Figure 42: Security Class definition in Author mode

5.4 Watermark

A watermark can be added on every page of the document. From the editorial point of view, the watermark is annoying for the reader, therefore no final document shall make use of watermarks on every page.

The watermark is supported in the DITAMAP in the `modification` field

```
<prodinfo>
  <prodname>Version
    <data name="image" value="front.png"/>
    <data name="top-left-width" value="100mm 80mm 80mm"/>
  </prodname>
  <vrmlist>
    <vrm version="0.2" release="20150928" modification="wmConf.svg"/>
  </vrmlist>
</prodinfo>
```

```
</vrmlist>
</prodinfo>
```

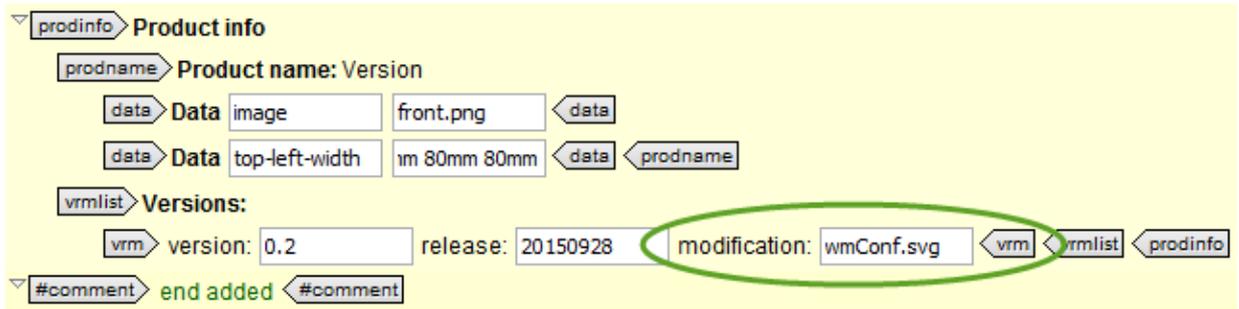


Figure 43: Watermark specification in the front page

Watermark on front page only

It might be required to have a watermark on the front page only.

The watermark drawing

The watermark drawing shall be transparent. This can be achieved with VISIO settings and the export to SVG.

5.5 Front picture

The front picture appears on the first page only. It is specified in the DITAMAP as follows

```
<prodinfo>
  <prodname>Version
    <data name="image" value="front.png"/>
    <data name="top-left-width" value="100mm 80mm 80mm"/>
  </prodname>
  <vrmlist>
    <vrmlist version="0.2" release="20150928" modification="wmConf.svg"/>
  </vrmlist>
</prodinfo>
```

The oxygen author mode shows the front picture definition as follows

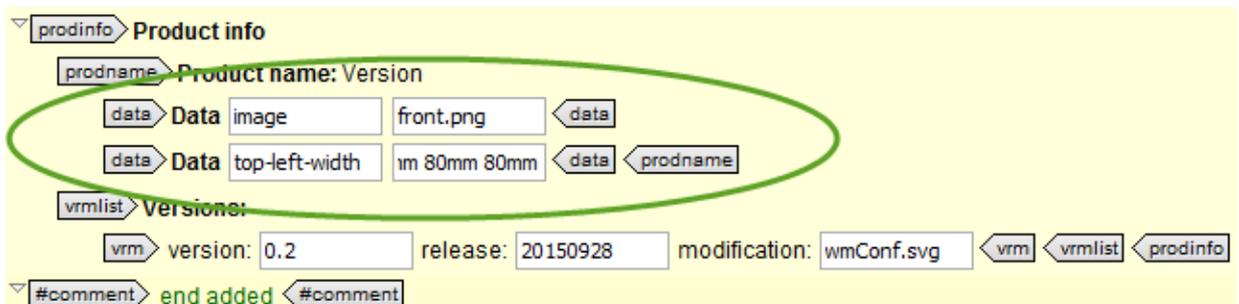


Figure 44: Front picture definition in oxygen author mode

The front picture itself must be in the `gfx`-directory.

Whenever possible - use a `.SVG` drawing, but very often you will have a photography which implies bitmap based content. High resolution photos will maintain the quality of the document.

5.6 Second page layout

If you omit the `<summary>` token from the basic template, the entire second page will disappear.

Otherwise it will contain a set of fields, that display copyright information and the SAP number of the book.

To be continued(just wanted to record the function of the `<summary>` tag.

6 Author's support

Several additional scenarios are provided in order to allow author's an enhanced comfort in creating high quality documentation

Topics	6.1 Creating a Bibliography	66
	6.2 Creating a Glossary	71

6.1 Creating a Bibliography

How to create a local bibliography using a master list

In many books a bibliography is a mandatory requirements. The manual creation and maintenance of a bibliography is annoying work. This can be facilitate by using master lists.

6.1.1 Creating a Master Bibliography

Master lists Using master lists is the first step to facilitate the tedious task. Enter all your bibliography entries in one larger file. Our present approach uses a table to store the documents most important parameters.

Master List Example A master bibliography is a concept that contains one or more tables of the following structure.

Table 8: Bibliography

BibEnt	Description	Publisher
[AHF]	Antenna House Formatter V6 User Manual	Antenna House
[Ant]	Apache Ant 1.8.1 Manual	Apache
[DitaSpec]	Darwin Information Typing Architecture (DITA) Version 1.2	OASIS
[DtPrt]	DITA for Print: A DITA Open Toolkit Workbook DITA Open Toolkit 1.8, Leigh W. White http://xmlpress.net	XML Press
[ezRead]	ezRead Documentation System Documentation Guide, Version 2.65	Helmut Scherzer
...

The three columns are specified as follows
BibEnt

The bibliography shortcut is a unique code specifying a document. This is exactly the code you will use when you reference a book. [\[ezRead#9.2\]](#) is such a reference and it points to the proposal of the naming convention for files.



Note: Of course you may use a reference without the chapter notation [\[ezRead\]](#), however, I highly recommend to use chapter suffix if you refer to a particular topic in the book. Find more to the philosophy in [\[ezRead#3.1.1\]](#)

In the (master-)bibliography, you will never use a chapter suffix, this doesn't make any sense.

There are some rules about the BibEnt

- It shall be unique in your entire document tree
- It shall not contain spaces
- It shall not contain special chars like [] ? . () – _ etc., use letters and numbers only

The BibEnt entry shall have an id of

```
spb_<term>
```

where <term> is the shortcut-text (here "ezread").

Description

The description shall reflect the short title and the other reference information. You may structure the content with paragraphs - finally the entire entry will be copied to the local bibliography.

The **Description** entry shall have an id of

```
spd_<term>
```

where <term> is again the shortcut-text (here "ezread").

Publisher

The Publisher entry shall contain the publisher's name and information. You may structure the content with paragraphs - finally the entire entry will be copied to the local bibliography.

The **Publisher** entry shall have an id of

```
spp_<term>
```



Notice: To facility the writing of the id's, you may only create the BibEnt-id (`spb-<term>`) and use the [RepairBibliography-Scenario](#) on order to create the other to ids

6.1.2 Referencing external documents (Bibliography)

For long times it is good practice to refer to external documentation with bracket [...] notation. Scientific articles are used to [7] numbered references whereas technical documentation often uses a more liberate (and efficient) notation like [\[ezRead\]](#).

We recommend an even more sophisticated and very powerful scheme. Using `[BibEnt#Chapter]` notation like [\[ezRead#9.2\]](#) is a unique and powerful method to address a particular point in a document, being a chapter number or any possible item e.g. [\[DitaSpec#fig\]](#).

The method is based on ezRead Technology which is explained in [\[ezRead#9.3.3\]](#).

What to use

Whenever you know the precise point of your reference i.e. the chapter or even a paragraph that you need to refer to, use the ezRead Notation [\[ezRead#9.3.3\]](#). If you only need to reference the title of a document but your statement does not related to any specific content, you may use the `[Abbrev]`

notation where you omit the chapter or target name. Only if you are enforced to use the numbered version [3] you may use that one.



Note: A future extension of our plugin will allow you to write the full notation, but the PDF will **print** the numbered notation instead. This would allow to maintain a link into the chapter while you still keep the rules of scientific discipline.

Auto-Links

The use of the ezRead Links buys you another advantage. The plugin will automatically create links in the PDF file without having you to enter tidy XREF statements. Find more in [4.6.3 ezRead auto-linking](#)

6.1.3 Generating a Local Bibliography.

The master bibliography might contain hundreds of (well maintained) references. Writing a new book, however, you would not want all of them in your book's (=local) bibliography.



Note: You might need to specify the location of your master file.

The setting is made in the included transformation `spcSelectRefMap.xsl`.

```
<!-- MasterPathName: Specify the path to your master file -->
<xsl:variable name="MasterPathName" select="'/F:/scherzer/
RefDita/src/McSpecification.dita'"/>
```

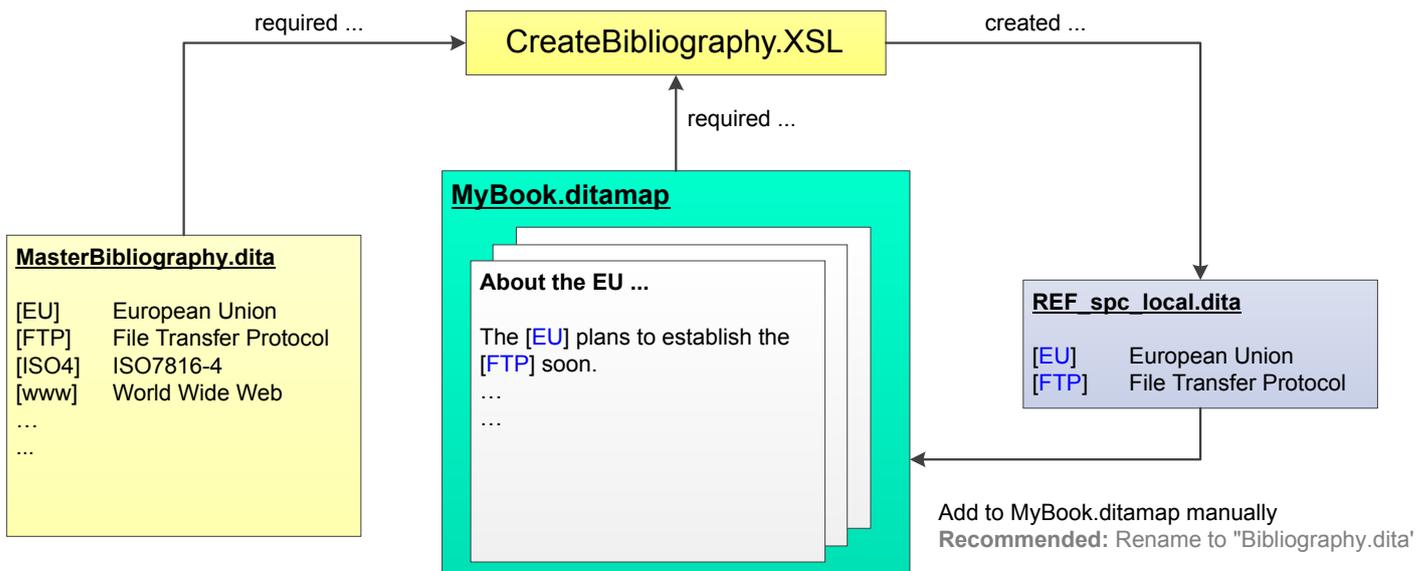


Figure 45: Creating a local bibliography

The `CreateBibliography.xsl` scenario creates a local glossary in the same path as your DITAMAP. The file name is `REF_spc_Local.dita`.

The result file `REF_spc_Local.dita` contains a bibliography with only those entries that you have referenced in any of your books chapters. Therefore it is important to apply the scenario `CreateBibliography.xsl` to your current DITAMAP in contrast to any of its chapters. Otherwise the scenario cannot find the files that belong to your current book.

The `REF_spc_Local.dita` is created in the DITAMAP source folder. This is made intentionally, you shall copy this generated file manually into your `concept` directory. At the same time, I recommend to give it another name (e.g. `MyBiblio.dita`) which will distinguish it from auto-generated content.



Attention: This manual copy is good advice. Using auto-generated chapters without author's review can lead to unwanted results. Therefore this little step is suggested and implemented.



Note: Although I wouldn't recommend it, the location of the target can be changed to create right into the books target directory. You need to edit the transformation sheet `CreateBibliography.xsl` for that purpose.

```
<!-- Create the local bibliography
{concat($folderURI,-->
  <xsl:result-document href="{concat($folderURI,
$bibName)}" format="xml">
    <xsl:apply-templates select="$mergedFiles"
mode="spc"/>
  </xsl:result-document>
```

Using

```
<xsl:result-document
href="{concat($folderURI, 'concept/', $bibName)}"
format="xml">
```

will bring place the result file in the `concept` (or whatever shall be your desired directory).

6.1.4 Ignore Lists

The auto-creation of the local bibliography implies an a-priori problem ... if you use the [...] notation for anything else (e.g. the indices of an array (A[3,6]) then the plugin cannot recognize whether you are just addressing a document to be listed in the bibliography or this is another use of brackets. To avoid warning entries on the created bibliography, you may use the `ignore.xml` list which will tell the scenario which [...] terms shall be ingored.

The `CreateBibliography.xsl` scenario generates a `Ignore_local.xml` which lists all [...] terms that were found eligible as candidates for a bibliography reference. In order to manage ignores, do the following;

- Rename `Ignore_local.xml` to `Ignore.xml`. `Ignore.xml` is the name of the file that lists terms to be ignored.
- Delete the valid bibliography entries from the `Ignore.xml`, you don't want valid entries to be ignored.
- Run `CreateBibliography.xsl` again and you get a proper Bibliography which ignores all the [...] constructs you have left in `Ignore.xml`.

6.1.5 Creating the oxygen scenario

Create a new scenario with **DITA Maps** → **Configure Transformation Scenarios**

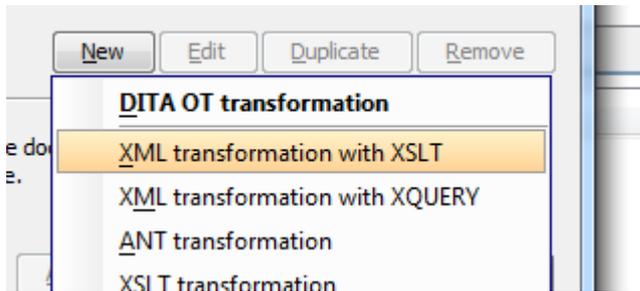


Figure 46: New scenario

Be sure to use the right Transformer type

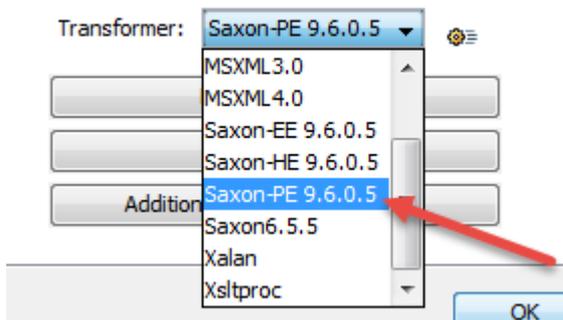


Figure 47: Select scenario type

Create the scenario according to Figure 3

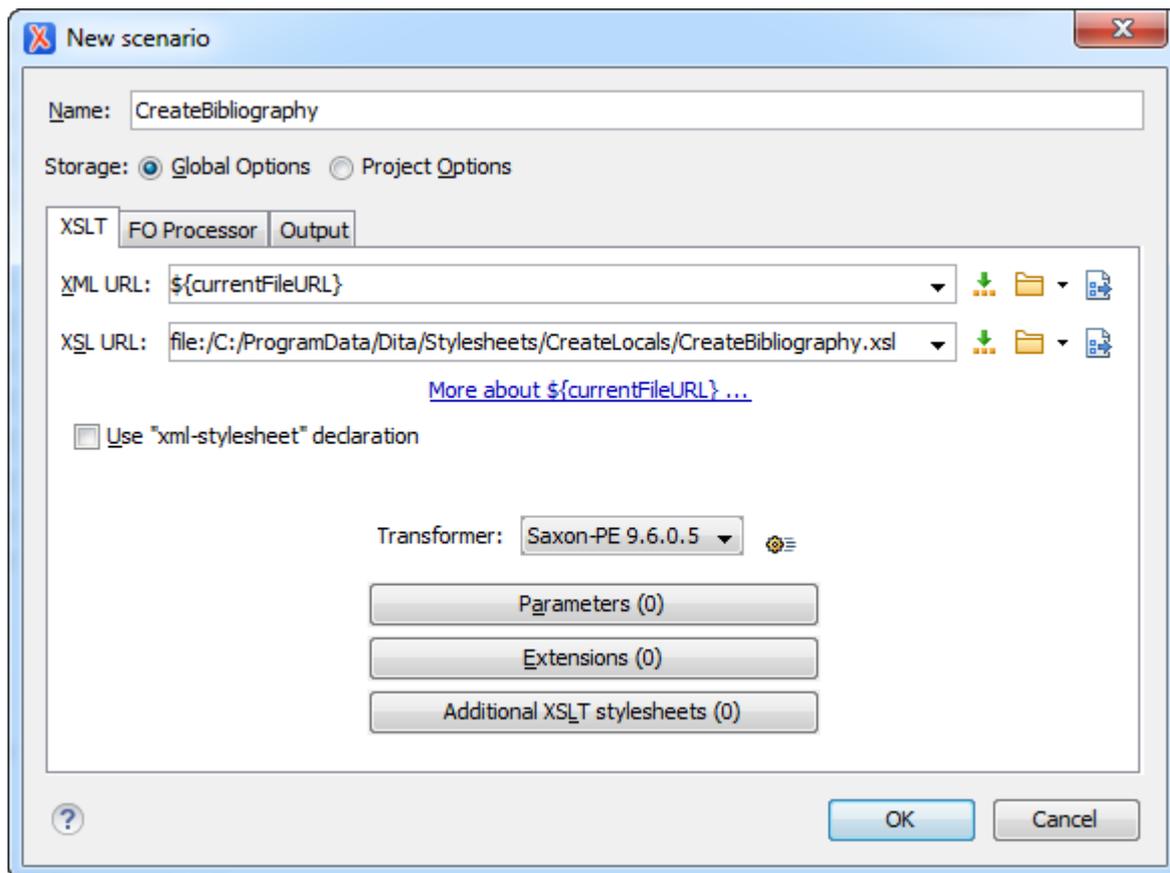


Figure 48: Create Scenario



Tip: Use the storage type **Global Options** which allows you to use the scenario for any other .ditamap.

6.1.6 Repair Master Bibliography

If you only created an id for the **BibEnt** entry, you may use the `RepairBibliography.xls` scenario. If applied to a *master bibliography*, it creates the corresponding ids for the second and third row (`spd_<entry>` and `spp_<entry>`)

6.2 Creating a Glossary

7 MS-Word Docx 2 Dita conversion

MS-Word files (DOCX) can be converted to DITA. A separate plugin is required for this - the good news ... it is part of the the [installation](#).

Topics	7.1 Installing Docx2Dita	72
	7.2 Running the docx2dita conversion	76

7.1 Installing Docx2Dita

How to install the docx2dita environment

Most of the installation comes already with the installation process, in particular the additional plugins and their integration into the [DITA-OT](#). However, some additional steps have to be taken until a docx2dita conversion can be performed.

style mapping A control file `style2tagmap.xml` is required to instruct the docx2dita process how to translate the MS-Word styles. As authors may invent any kind of style, the docx2dita process cannot know them and therefore they need to be specified in the `style2tagmap.xml`.

The `style2tagmap.xml` file is found in `C:\ProgramData\Dita\setttings\style2tagmap.xml`. A default file is already contained in the [installation process](#).

7.1.1 Style mapping

t.b.d. How to map styles

7.1.2 Create docx2dita scenario(s)

How to create the oxygen scenarios

To run the conversion from the oxygen environment, first a scenario has to be created.

1. Open oxygen and open any file of your choice.



Note: It is not important which file you open, but oxygen often expects an open file until it lets to edit the scenarios.

The transformation scenario panel has an  icon on the right to corner. Use this to select the "all scenarios" view

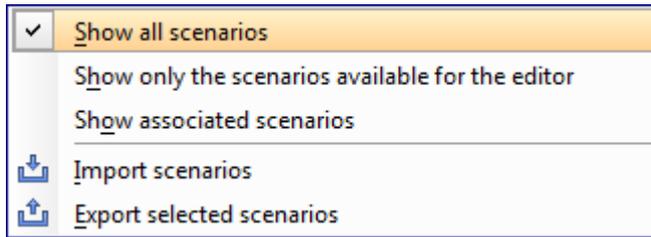


Figure 49: All scenarios view

- 2. Duplicate the DOCX DITA scenario on the OOXML section

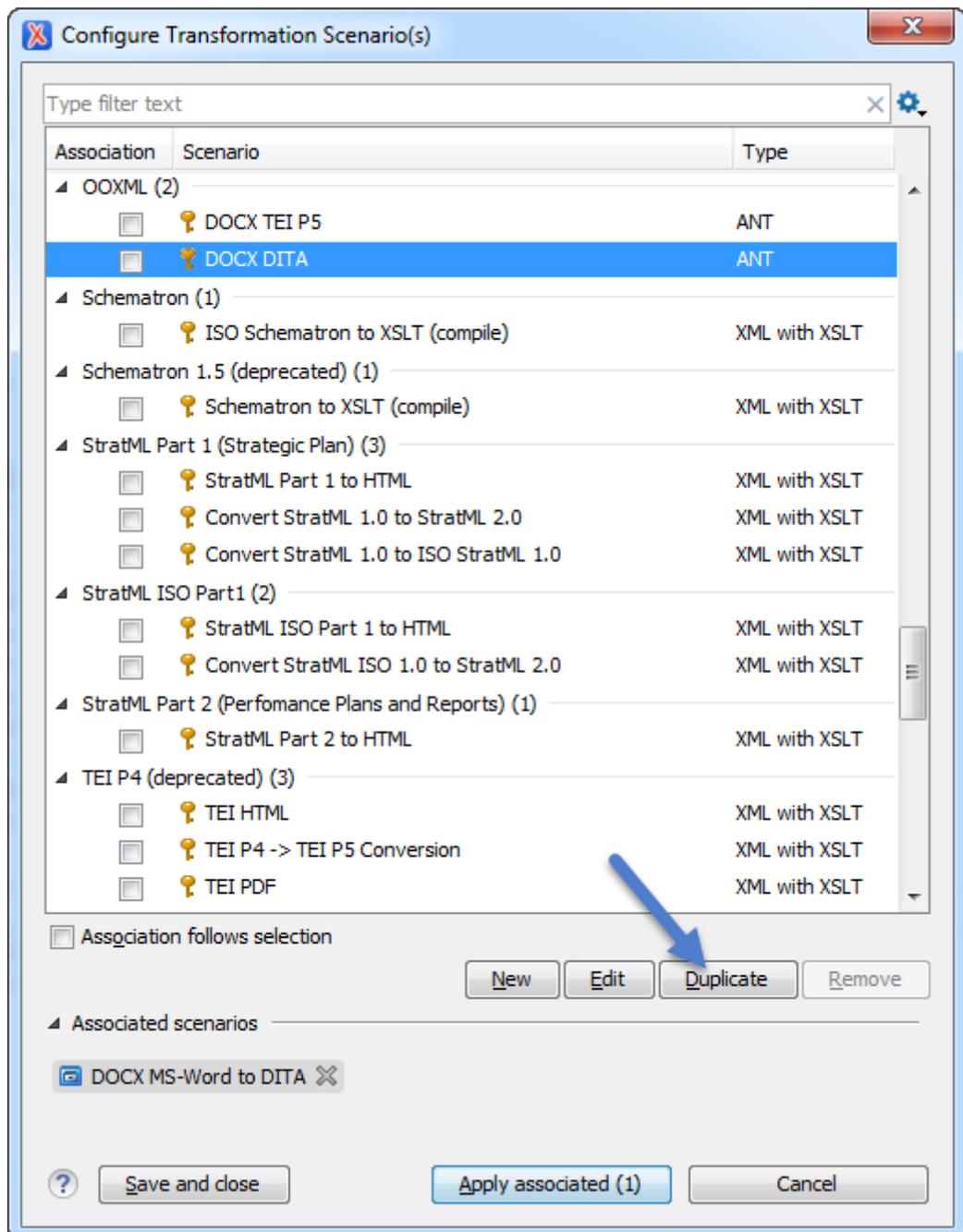


Figure 50: Open oxygen scenarios

and give it the title `DOCX MS-Word to DITA - DCI`.

3. Select the new scenario (before you may change the view back to local view) and change the **Options**

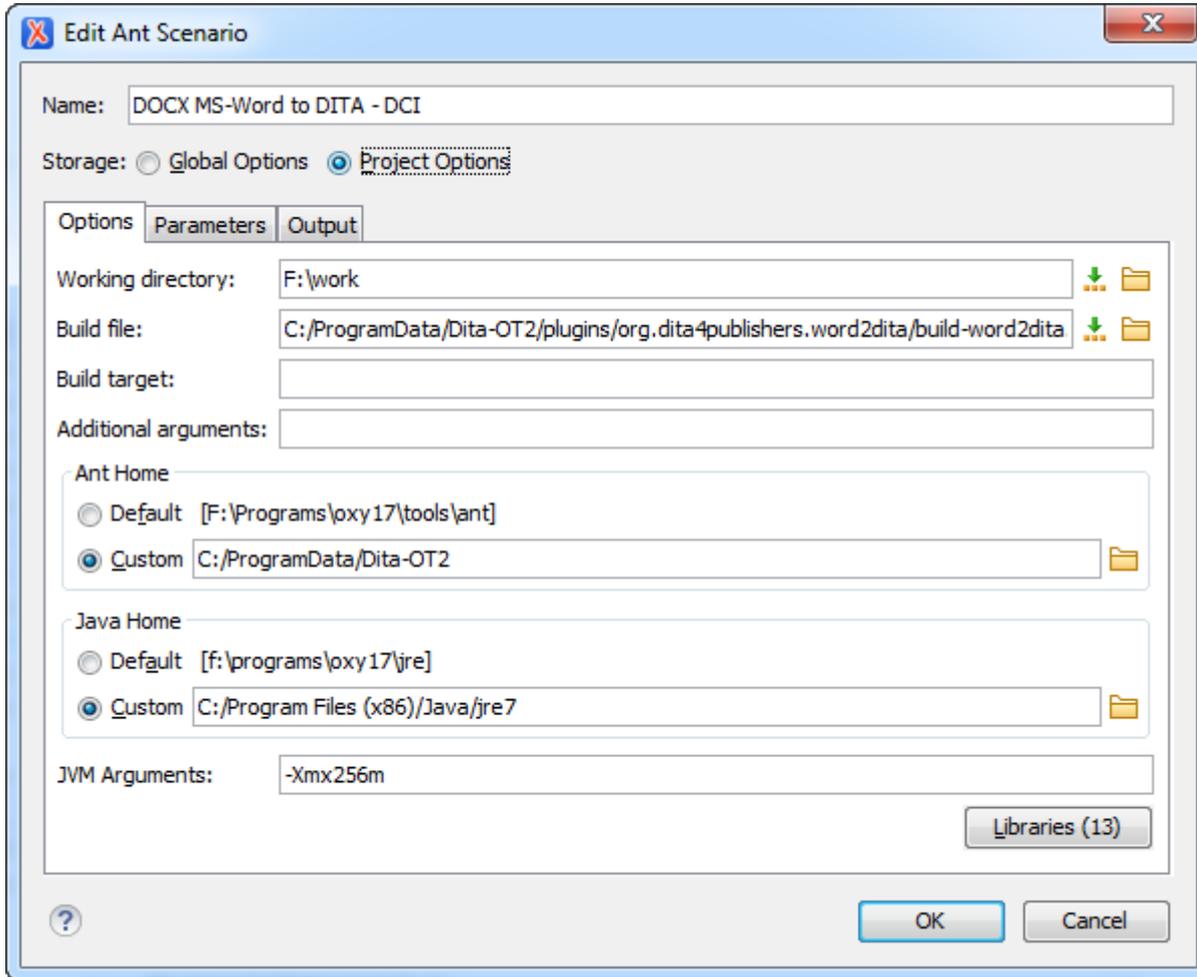


Figure 51: Change the scenario options

4. Change the **Parameters** as follows

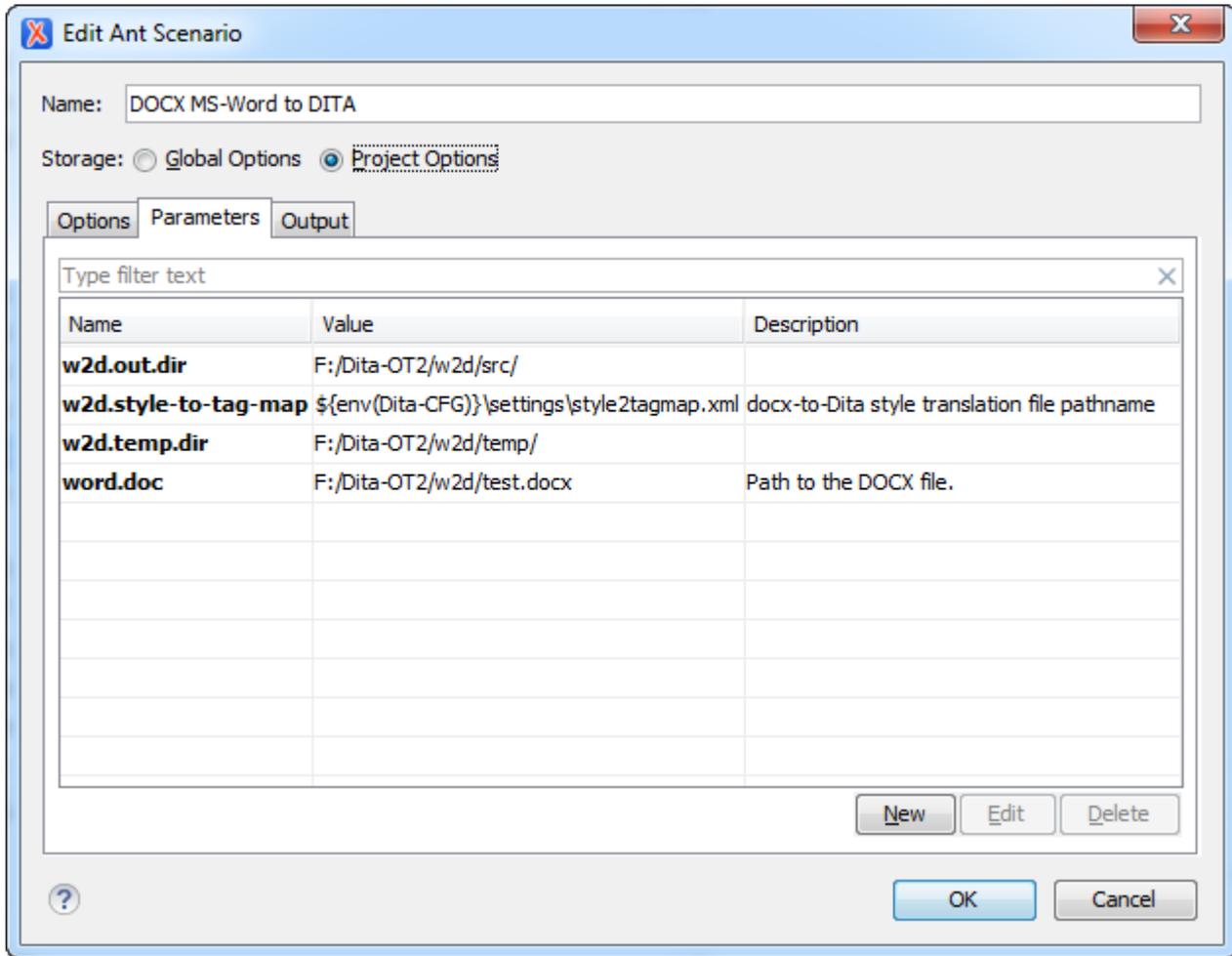


Figure 52: Edit Parameters



Important: The `word.doc` variable shall contain your specific `docx` input file.



Warning: You shall enter the **full path** to your `.docx` file. Otherwise the conversion will fail with an error. Do not use relative paths or even the oxygen variables.

5. Change the Output accordingly

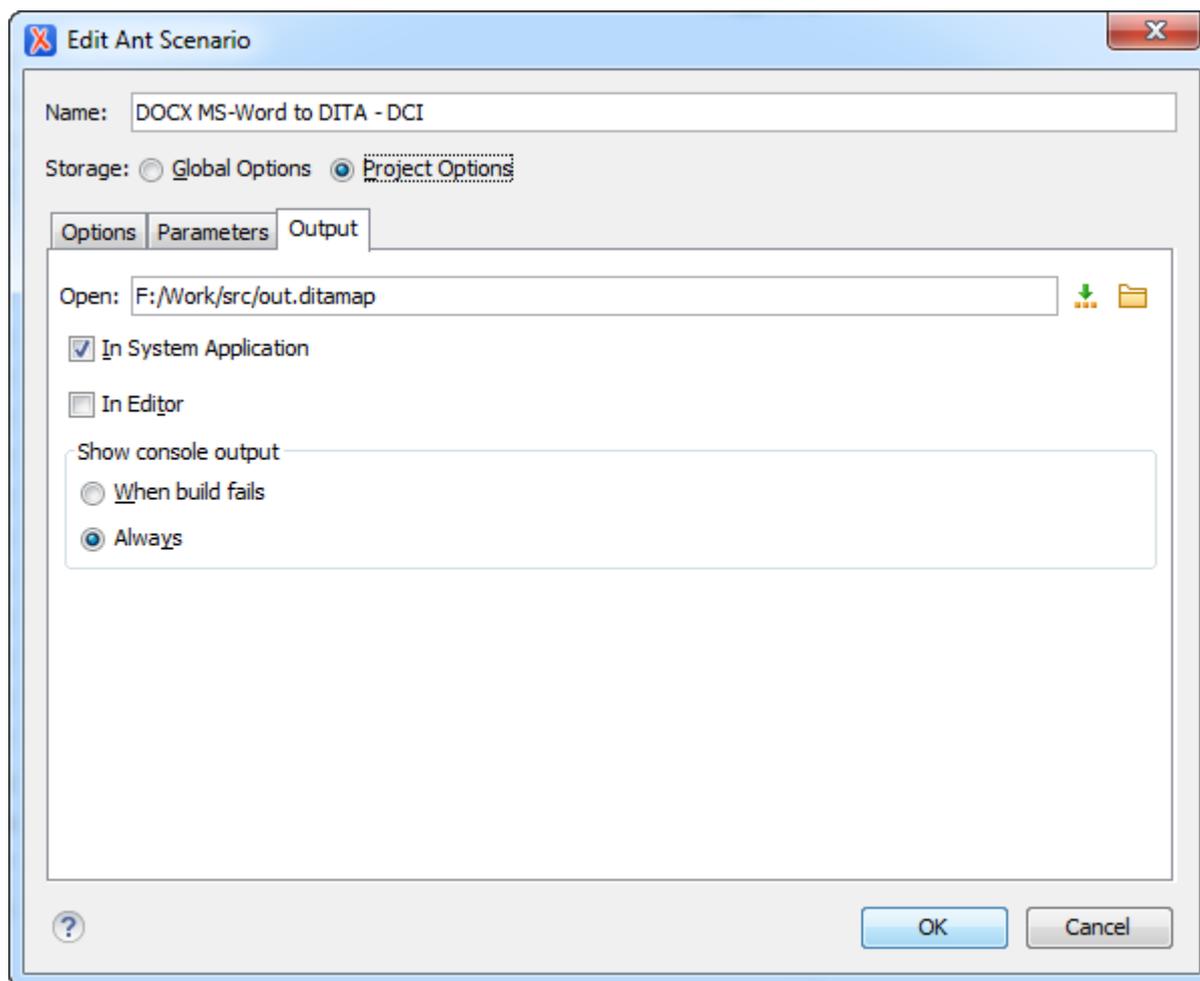


Figure 53: Edit output parameters

Of course you may choose your own output directory.

7.2 Running the docx2dita conversion

7.2.1 Prepare the DOCX for conversion

If your MS-Word document is a .DOC document first you need to

1. Open the document with MS-Word
2. Save-As the document as .DOCX

The DOCX2DITA conversion uses the first paragraph with `style=Title` in order to start the conversion. Hence you need to

1. open the .DOCX document
2. go to the top of the document and write some text e.g. "Dummy Title"

- 3. Assign the paragraph style "Title" to the document.



You might not find the style "Title" easily. Then you need to select the small button in the **Change Styles** Home ribbon. The style box will open and you should go to its lower right corner where you select the **options**



When you open the Style Pane Options, you should select **All styles**. This will show the Title style in the Style list.

- 4. Assign the Title style to your text.
- 5. An alternative to step3 and 4 is the use of the right mouse-key to open the context menu

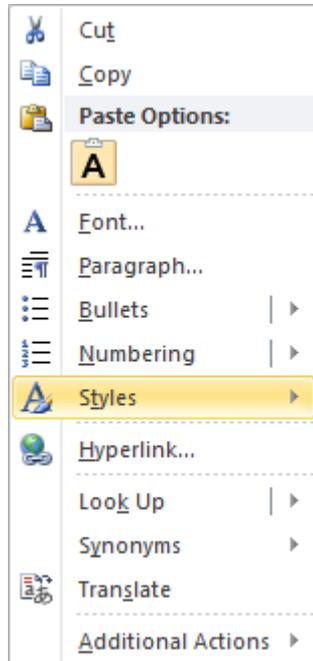


Figure 54: Select the 'styles' context menu

Select **Styles** and

then

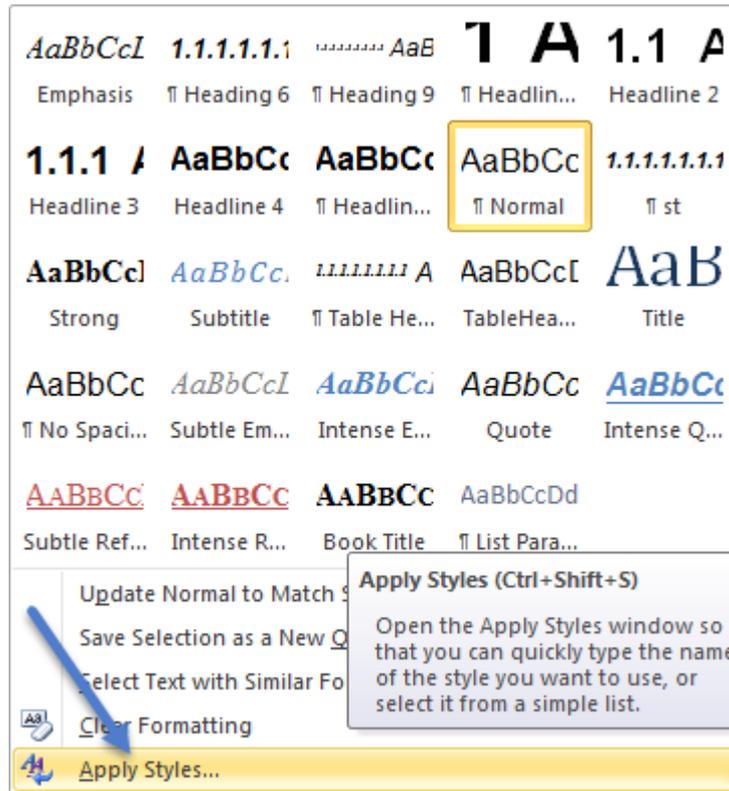


Figure 55: Apply styles

The following dialog let's you enter the style `Title`

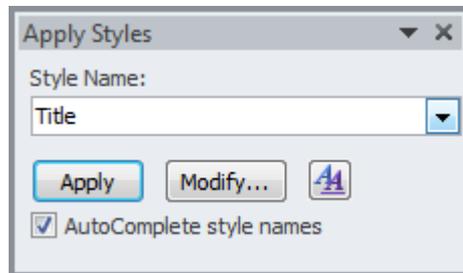


Figure 56: Style Dialog

6. **Verify** that there is **no other text on the page** with the `Title` style - if this is the case, just assign another style (e.g. `Normal`) to such text, otherwise the conversion might get some trouble.



Danger: Removing the rest of the title page's text is very important - do not ignore this. I had several documents which converted only after I removed all text from the front page (except the created "title" styled text of course).

The reason for this step is, that front pages very often have the most odd ideas of styles and layout which is hard to control by the conversion. Removing all that information does not pay hard because the conversion would not bring this to the DITAMAP anyway. A bit of manual work is always required.

After these preparations, you should be able to convert the document

7.2.2 Converting docx2dita from oxygen GUI

At a first glance it seems natural to convert from the oxygen GUI. The comfort, however, has to be paid with changing the input file for every DOCX file to be converted. This is because you cannot just open such DOCX file because it is a binary (zipped) file which oxygen cannot understand.

1. **Open any file** and change the `word.doc` parameter as described in [Figure 4](#)
2. Assign the `DOCX MS-Word to DITA - DCI` scenario to your opened file. The file isn't really used because the actual input file is that which you named in the `word.doc` parameter.
3. Run the scenario as usual.
4. The system will convert the input file and the log is shown in oxygen's console output



Tip: You might also need to change the `style2tagmap.xml` to accomplish full conversion
[7.1.1 Style mapping](#)

7.2.3 Converting from the command line

How to start the conversion from the command line

Converting a `<filename>.DOCX` from the command line is more powerful because the scenario parameters for the oxygen environment do not need to be changed. Instead the variable parameters (e.g. the input file name "`test.docx`") can be given on the command line.

The command line invocation is

```
F:\work>word2dita <filename>.DOCX
```

which already assigns all parameters correctly.

*Use
style2tagmap*

To optimized the output result it is quite likely that you need to edit the `style2tagmap.xml`, which controls the translation of word-styles into DITA topics. If your change is more of a global nature (i.e. useable for later translations) you might edit the default `style2tagmap.xml` in `C:\ProgramData\Dita\settings\style2tagmap.xml`

Special styles

If your change is rather special (because you want to translate a DOCX which has very odd and special styles) it might be good advice not to change the default file `C:\ProgramData\Dita\settings\style2tagmap.xml` but to copy that file to a local directory. Then you need to set a system variable to address the local file.

1. copy `C:\ProgramData\Dita\settings\style2tagmap.xml` to your local directory. In the following we will assume `F:\work` as an example where we have our `complexstyles.docx` input file to be converted.
2. In the command line window ... issue

```
F:\work>set Dita-Settings=%CD%/style2tagmap.xml
```

This statement sets an environment variable `Dita-Settings` to the current working directory `F:\work` where you just copied also the `style2tagmap.xml`

3. Launch

```
F:\work\word2dita complexstyles.docx
```

and the conversion will be done with your local `style2tagmap.xml`. The log file (automatically opened after the process) will show you warnings that indicate styles not being covered by the `style2tagmap.xml`.

4. Edit `style2tagmap.xml` to cover all styles.
5. Process again until you are satisfied with the result.

This should give you fast results. The `word2dita.bat` batch file is found in `C:\ProgramData\batch`.

7.2.4 Post Processing

Steps to be done after conversion.

You might need to do some steps after you successfully processed the document.

Dita violations If the results of your conversion violate DITA rules (e.g. a `Heading 1` style within in a table entry) there is a special `postProcess.xsl` available in

```
C:\ProgramData\Dita-OT2\plugins\org.dita4publishers.word2dita\xsl
\postProcess.xsl
```

If you are experienced and have worked along [10 Programming Stylesheets](#) , you can repair such situations using template matches on the violating situations.



Notice: The default template removes `title` tags within table entries, this occurs if an author has used `Heading n` styles within table entries.



Notice: In general it is impossible to avoid violations because MS-Word is not a structured and rule based authoring system.

8 Docbook to Dita conversion

to be done

Scenarios to run

1. TcDoc2DitaPrc
2. TcDoc2DitaMap

Things you need to consider:

- Delete the `xmlns="http://docbook.org/ns/docbook"` statement from the header of the input document.
- Find the glossary (`sect | appendix`) and add an attribute `type="glossary"` to the `sect | appendix`.



Note: The system cannot find a glossary if the author didn't use some corresponding docbook topic type. Therefore we need to mark the glossary with the `type="glossary"` attribute.

9 CHM output

CHM is known as a Microsoft format also called "HTML Help" format.

Topics	9.1 Installing CHM features	82
	9.2 Producing CHM files	87
	9.3 Changing CSM styles	88
	9.4 Changes to the DITA-OT	89
	9.5 Understanding the CHM process	91

9.1 Installing CHM features

To produce CHM output an oxygen scenario needs to be created. Until we extend the features, the following scenario creation will do.

1. Create a new transformation scenario

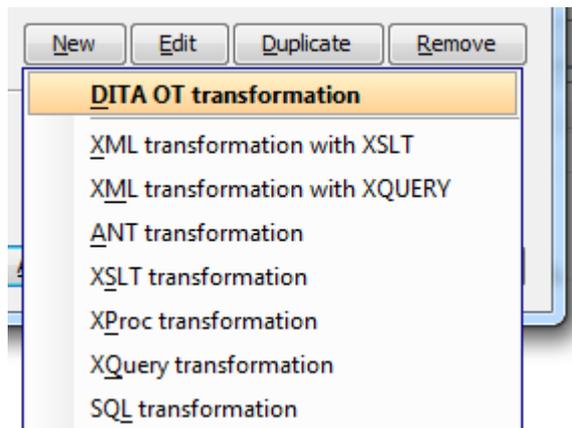


Figure 57: New transformation

2. Select the HTMLHelp type

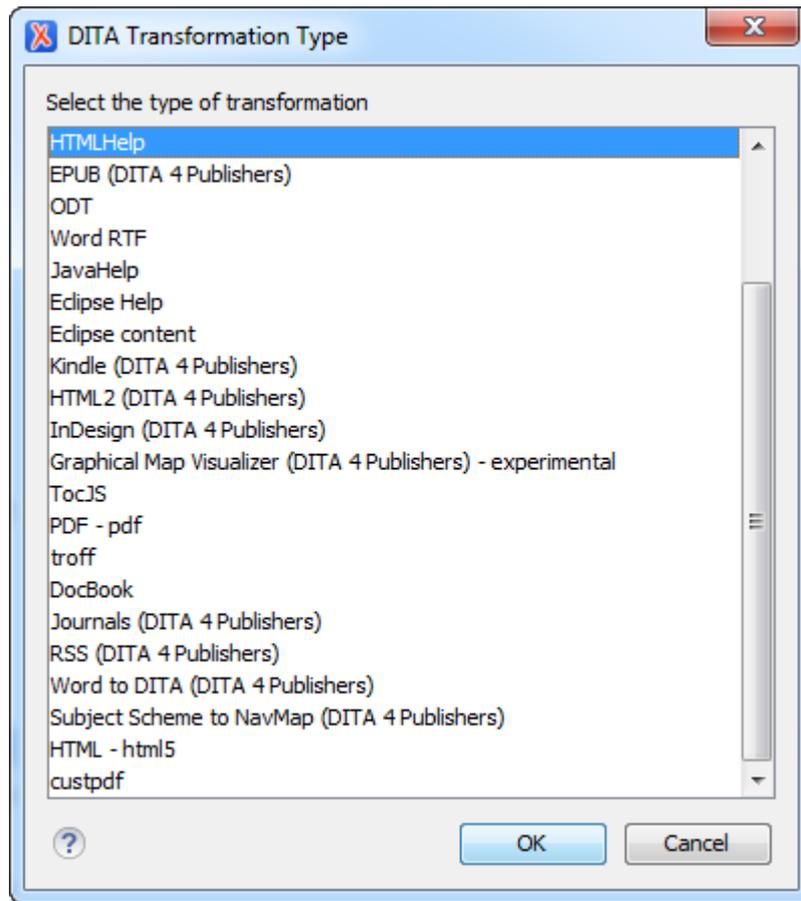


Figure 58: Type selection

3. For now ... no changes necessary in the **Parameters** tab

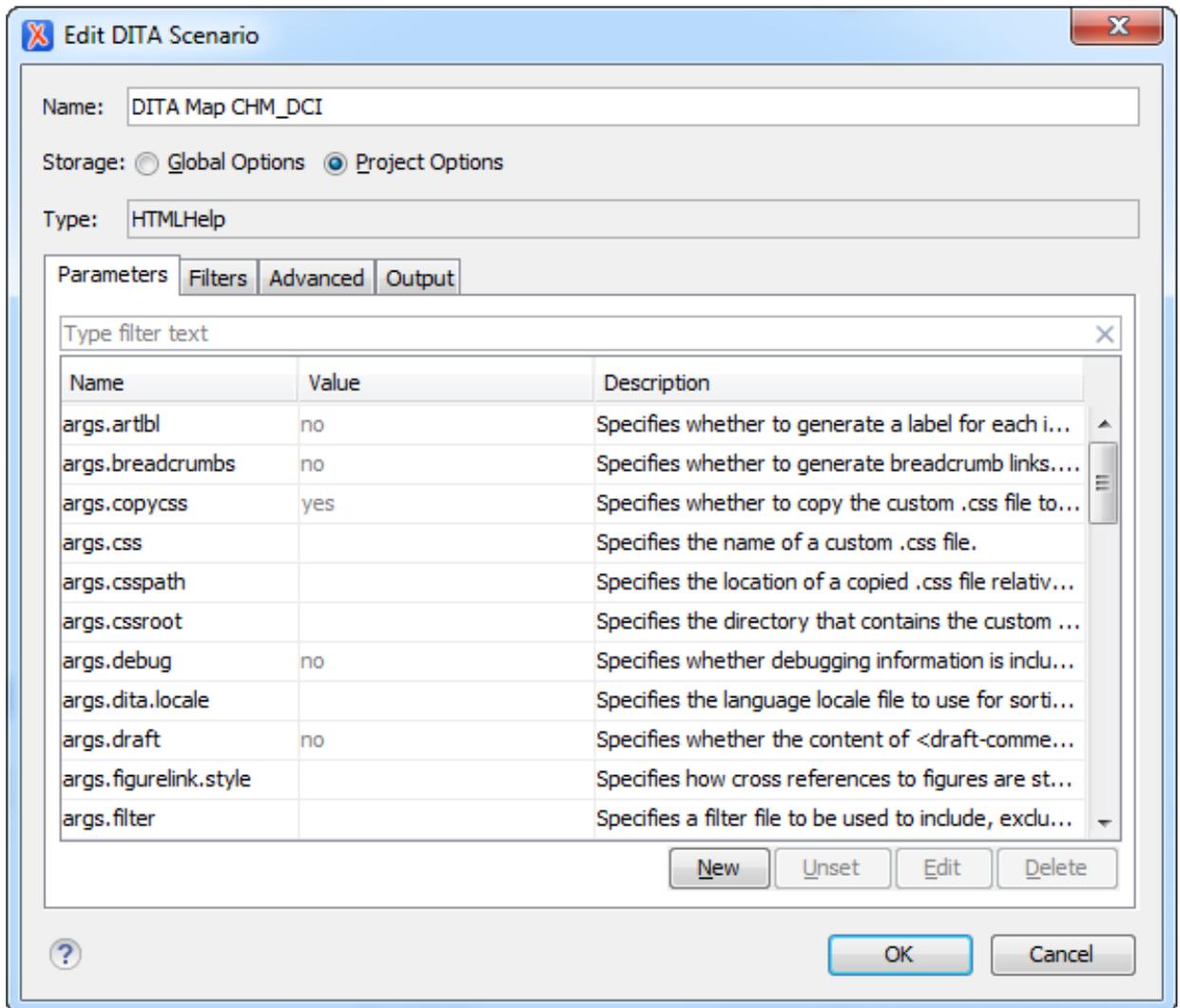


Figure 59: Parameters tab

4. DITaval file can be specified here

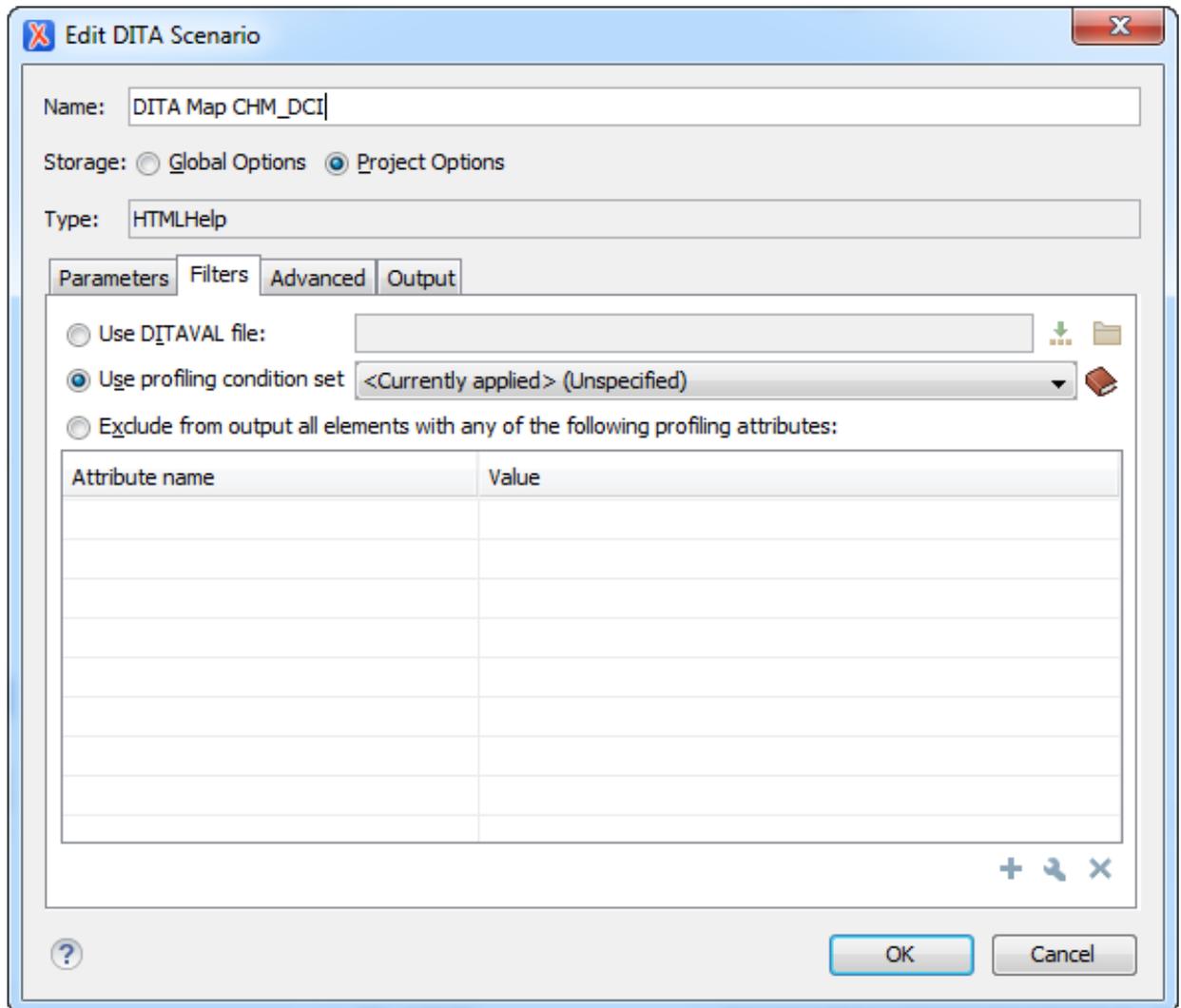


Figure 60: Filters settings

5. For now, no changes in the **Advanced** tab

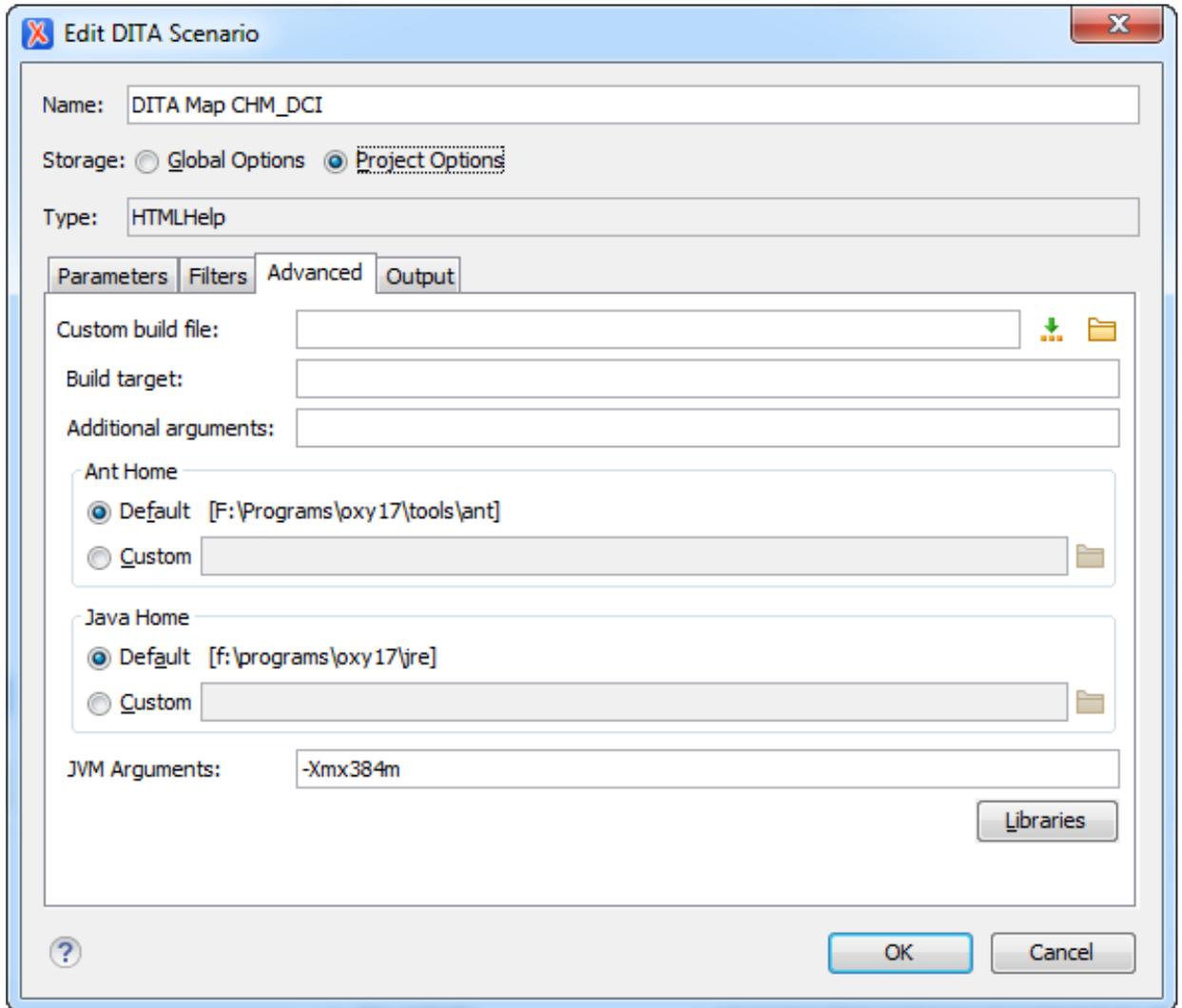


Figure 61: Advanced tab

6. Output settings shall be specified according to our common standards

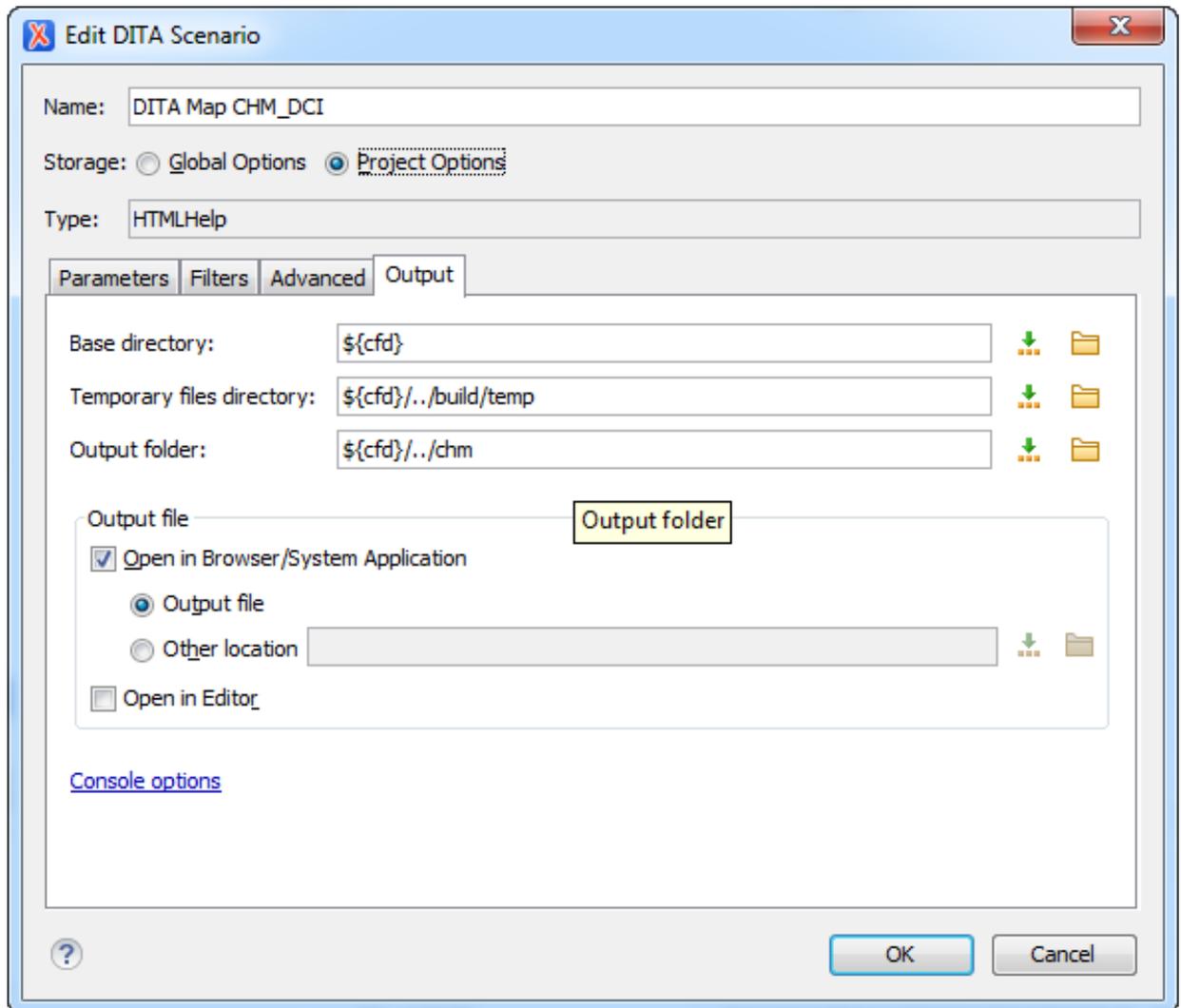


Figure 62: Output specification

9.2 Producing CHM files

CHM production is an extension of HTML production. Therefore there will be always HTML files in the output directory. The generation of CHM is just a post-processing step from the HTML.

The advantage of this is ... you can check the HTML files first for proper layout and consistency. A CHM file is a binary (compressed) file and therefore cannot be debugged easily.

To produce CHM ...

1. Run the scenario that you created in [9.1 Installing CHM features](#)
2. Test the HTML files in the output directory.

3. Be careful to change the DITA content if you are not happy with the result - it will affect the PDF output (which you might not like), there will be other ways to fix it.



Warning: Although we highly recommend to use [SVG](#) kind of figures, they do not (yet) seem to be supported by the HTML/CHM files. Therefore, for HTML, you need to use bitmap based formats (PNG/BMP)



Important: Do not think you should become practical and "always use bitmap based graphics". That will be a decrease in quality that we should not allow. We will later have an automatic conversion of a link to an SVG into a link to a PNG so you can and should still use SVG as the prime choice.

9.3 Changing CSM styles

The directory ... \src contains a `commonltr.css` which is all the magic about the right styles. (ltr = left to right reading).

The ... \src\chapters contains many HTML files which actually represent those files that will go to the CHM. Same formatting !!!

The important action now is to edit the `commonltr.css` such that the output is as pretty as technical writers would like it to publish. That's why this is a good task for the team. The idea of the CSS is easy to understand.

- Open an HTML file in the ... \chapters directory (e.g. CardFileSystem-91.dita) - you should open the file in the Internetbrowser (to see the formatting) AND in NotePad++ text editor (to see the source).
- Search the class of a desired topic to be changed (e.g. the `topicTitle1` or `sectionTitle`) in the HTML file. Search the definition of that class (e.g. `.sectionTitle { }`) in the `commonltr.css`.
- Add the desired changes to that definition.
- On the Internetbrowser (where you have opened the HTML file) press "F5" to "refresh". You should see your changes immediately (what makes the fun of it).
- After you have done many changes - SAVE `commonltr.css` to any directory !!!!



Warning: This is very important. If you process the document (using `r.bat`) your changes will be totally overwritten by the `commonltr.css` which is copied from C: \ProgramData\DITA-OT2\plugins\org.dita.xhtml\resource. So you need to care, otherwise your work is lost.

- If you are happy with the changes you shall
 - send them to me and
 - copy your changed `commonltr.css` to the C: \ProgramData\DITA-OT2\plugins \org.dita.xhtml\resource directory to make it permanent.

9.3.1 CHM Font definition

The `transtype = htmlhelp` process first uses the `org.dita.xhtml` transformation to catch the dita conversion to any web type. The `commonrtl.css` of this transformation type is used and copied to other post-transformations like `transtype = htmlhelp`.

The XHTML transform was the first transform created for the DITA Open Toolkit; it converts DITA topics into XHTML documents. In addition to the XHTML output, this transform also returns a simple table of contents file named `index.html`, which is based on the structure of the input map file.

XHTML output is always associated with the default DITA-OT CSS stylesheet "`commonltr.css`" (or "`commonrtl.css`" for right-to-left languages). Parameters are available to override the default CSS styling.

To run the default XHTML transform, set the transform type parameter to "`xhtml`".



Notice: Many of the other transform types run the same conversion to XHTML, followed by additional routines to create new navigation files.

Within the `commonrtl.css` the standard font can be set with the following statement

```
/* Set the default font and styles */
* {
    font-family: Arial;
}
```

The notation catches all classes and allows to set anything to the desired parameters as default.

Related Information

<http://dita-ot.sourceforge.net/1.6/readme/dita2xhtml.html>

9.4 Changes to the DITA-OT

To process CHM, some changes had to be done to the DITA-OT.

Copying .PNG files

During the build process the graphic files are copied from the source directory to the target directory. However, as we prefer `.SVG` files, there is a problem that browsers (and CHM) cannot display SVG files.

Therefore we have to provide `PNG` graphics (also exported from VISIO). The aliasing problem (low or odd resolution) does not exist in browser/CHM presentation mode, however, it does exist in PDF.



Note: Therefore - never include pixel based image files (e.g. `PNG/BMP`) if you can provide a vector graphics file (e.g. `SVG`)

Of course - here's the problem ... If we have included `.SVG` files, how can we then attach the corresponding `.PNG` to the DITA source? The answer is: "we don't have to". I changed the `DITA-OT` such that there is nothing to do but to provide a `.PNG` parallel to the `SVG` version.

The changes are as follows:

**F:\Dita-OT2\plugins
org.dita.base
build_preprocess.xml**

Changed the copy process `copy-image` to

```
<target name="copy-image"
unless="preprocess.copy-image.skip" depends="copy-
```

```

image-check" description="Copy image files">
  <condition property="copy-image.todir" value="$
{output.dir}/${uplevels}" else="{output.dir}">
    <equals arg1="{generate.copy.outer}"
arg2="1"/>
  </condition>
  <echo level="info">*HSC:Copying image files $
{dita.temp.dir}/${imagefile} from $
{user.input.dir} to ${copy-image.todir}</echo>

  <!--HSC use [Ant#filter] -->
  <copy file="{dita.temp.dir}/${imagefile}"
tofile="{copy-image.todir}imagepng.list">
    <filterchain>
      <linecontainsregexp>
        <regexp pattern=".svg"/>
      </linecontainsregexp>
      <tokenfilter>
        <replacestring from=".svg"
to=".png"/>
      </tokenfilter>
    </filterchain>
  </copy>

  <copy todir="{copy-image.todir}">
    <fileset dir="{user.input.dir}"
includesfile="{copy-image.todir}imagepng.list"/>
  </copy>

  <!--HSC original code needs to be maintained --
>
  <copy todir="{copy-image.todir}">
    <fileset dir="{user.input.dir}"
includesfile="{dita.temp.dir}/${imagefile}"/>
  </copy>
</target>

```

The change copies the file `image.list` from the *temporary directory* (created during the process) to the *image target directory* as `imagepng.list`.



Notice: You may change this to remain in the temporary directory, but for debugging purposes it was very practical to see what's happening.

While copying, a *filter* is applied which

1. only copies file names of **SVG** files
2. changes the extension **SVG** into **.PNG**
3. copies the **.PNG** files according to the new file list (`imagepng.list`)

Now the **.PNG** files are present in the target directory. An alternative would have been to change the original `image.list` since copying the **SVG** files isn't necessary anyway. I took the conservative way, but this may be easily changed.

Change href in the DITA source

Copying [.PNG](#) does not yet solve the problem that we refer to [SVG](#) wherever it is possible. Another change is taken in

F:\Dita-OT2\plugins
 \org.dita.xhtml\xsl
 \xslhtml
 \dita2htmlimpl.xml

This transformation processes most of the DITA source and also the href statement of images. The actual change is done where the href statement is interpreted. A regex-expression changes the extension [.SVG](#) into [.PNG](#)

```
<!--HSX replace svg extension by png extension in
order to automatically use
pixel based graphics for HTML and CHM -->

  <xsl:template match="*[contains(@class, ' topic/
image ')]/@href">
    <xsl:attribute name="src" select="replace(.,
'svg', 'png')"/>
  </xsl:template>
```

Processing transtype=htmlhelp will therefore automatically change href references from [SVG](#) to [.PNG](#).

9.5 Understanding the CHM process

While doing research to fix the problem that in CHM files the links to glossary entries were not resolved correctly I learned several things about the process.

Regular ID generation

Regular id's are generated in

```
<xsl:template name="setidattr">
  <xsl:param name="idvalue"/>
  <!--HSC for debugging purpose only -->
  <xsl:message>
    <xsl:text>Shit7-setidattr:[</xsl:text>
    <xsl:value-of select="name()"/>
    <xsl:text>] = </xsl:text>
    <xsl:value-of select="$idvalue"/>
  </xsl:message>
  <xsl:attribute name="id">
    <xsl:text>Shit41:</xsl:text>
    <xsl:value-of select="dita-ot:get-prefixed-id($idvalue/
parent::*, $idvalue)"/>
  </xsl:attribute>
</xsl:template>
```

The actual id comes in *\$idvalue* and is then translated to a unique value using `dita-ot:get-
prefixed-id`.

For example the paragraph `topic/p` template uses this

```
<xsl:template match="*[contains(@class, ' topic/p ')]" name="topic.p">
```

```

...
...
<xsl:call-template name="commonattributes"/>
<xsl:call-template name="setid"/>
<xsl:apply-templates/>
...

```

Consequently to get the setaname working for a glossterm, a template shall be created to catch the situation.



Tip: The glossterm is actually seen as kind of a title, it's class is `topic/title concept/ title glossentry/glossterm` which shows that glossary is derived from a concept title. Hence the handling of glossterm can be made through the title handler.

Heading ID generation

Any element that contains a class with `topic/title` will not pass the regular id-process, but be processed by

```

<xsl:template match="*[contains(@class, ' topic/topic ')]
/*[contains(@class, ' topic/title ')]">
  <xsl:param name="headinglevel" as="xs:integer">
    <xsl:choose>
      <xsl:when test="count(ancestor::*[contains(@class, '
topic/topic ')]) > 6">6</xsl:when>
      <xsl:otherwise>
        <xsl:sequence
select="count(ancestor::*[contains(@class, ' topic/topic ')]"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:param>
    <xsl:element name="h{$headinglevel}">
      <xsl:attribute name="class">topic:title<xsl:value-of
select="$headinglevel"/></xsl:attribute>
      <xsl:call-template name="commonattributes">
        <xsl:with-param name="default-output-
class">topic:title<xsl:value-of select="$headinglevel"
/></xsl:with-param>
      </xsl:call-template>
      <!--HSC create id of a heading in an HTML file -->
      <xsl:attribute name="id">
        <xsl:text>Shit22:</xsl:text>
        <xsl:apply-templates select="." mode="return-aria-label-
id"/>
      </xsl:attribute>
      <xsl:apply-templates/>
    </xsl:element>
    <xsl:value-of select="$newline"/>
  </xsl:template>

```

This is in particular valid for **glossary entries**. The template catches the `glossgroup` if the glossary has a title element, which is the case as `glossterm` is derived from title (see above). The `apply-templates` section will then visit the glossary entries. The called template

```

<xsl:template match="*[contains(@class, ' topic/topic ')]
/*[contains(@class, ' topic/title ')]"

```

```

mode="return-aria-label-id">
<xsl:choose>
  <xsl:when test="@id">
    <xsl:sequence select="dita-ot:generate-id(parent::*/@id,
@id)"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:text>ariaid-title</xsl:text>

    <xsl:number count="*[contains(@class, ' topic/title ')]
[parent::*[contains(@class, ' topic/topic ')]]"
level="any"/>
  </xsl:otherwise>
</xsl:choose>
</xsl:template>

```

is using `dita-ot:generate-id` to create the id.

href-Creation

Most important is also the creation of href links. The original DITA-OT (2.1.1) made several changes, this is where I stepped in but I found that there are several bugs

- Glossary terms (`glossterm`) are not referenced correctly
- title topics cannot be referenced at all because the DITA `@id` attribute is not propagated.

Finally I found that the href is produced in `rel-links.xml`

```

<!--template for xref-->

  <xsl:template match="*[contains(@class, ' topic/xref ')]"
name="topic.xref">
  <xsl:choose>
    <xsl:when test="@href and normalize-space(@href)">
      <xsl:apply-templates select="." mode="add-xref-highlight-at-
start"/>
      <a>
        <xsl:call-template name="commonattributes"/>
        <xsl:apply-templates select="." mode="add-linking-attributes"/>
        <xsl:apply-templates select="." mode="add-desc-as-hoverhelp"/>
        <!-- if there is text or sub element other than desc, apply
templates to them
otherwise, use the href as the value of link text. -->
        <xsl:choose>

```

where the actual creation is done in

```

<!-- When converting to mode template, move commonattributes out;
this template is dedicated to linking based attributes, and
allows the common linking set to be used when commonattributes
already exists for an ancestor. -->
<xsl:template match="*" mode="add-linking-attributes">
  <xsl:apply-templates select="." mode="add-href-attribute"/>
  <xsl:apply-templates select="." mode="add-link-target-attribute"/>
  <xsl:apply-templates select="." mode="add-custom-link-attributes"/>
</xsl:template>

```

coded as

```
<xsl:template match="*" mode="add-href-attribute">
  <xsl:if test="@href and normalize-space(@href)">
    <xsl:attribute name="href">
      <xsl:apply-templates select="." mode="determine-final-href"/>
    </xsl:attribute>
  </xsl:if>
</xsl:template>
```

whose implementation is

```
<xsl:template match="*" mode="determine-final-href">
  <xsl:choose>
    <xsl:when test="not(normalize-space(@href)) or empty(@href)"/>
      <!-- For non-DITA formats - use the href as is -->
      <xsl:when test="(empty(@format) and @scope = 'external') or
        (@format and not(@format = 'dita'))">
        <xsl:value-of select="@href"/>
      </xsl:when>
      <!-- For DITA - process the internal href -->
      <xsl:when test="starts-with(@href, '#')">
        <xsl:text>#</xsl:text>
        <xsl:value-of select="dita-ot:generate-id(dita-ot:get-topic-
          id(@href), dita-ot:get-element-id(@href))"/>
      </xsl:when>
      <!-- It's to a DITA file - process the file name (adding the html
        extension)
        and process the rest of the href -->
      <xsl:when test="(empty(@scope) or @scope = ('local', 'peer')) and
        (empty(@format) or @format = 'dita')">
        <xsl:call-template name="replace-extension">
          <xsl:with-param name="filename" select="@href"/>
          <xsl:with-param name="extension" select="$OUTEXT"/>
          <xsl:with-param name="ignore-fragment" select="true()"/>
        </xsl:call-template>
        <xsl:if test="contains(@href, '#')">
          <xsl:text>#</xsl:text>
          <xsl:value-of select="dita-ot:generate-id(dita-ot:get-topic-
            id(@href), dita-ot:get-element-id(@href))"/>
        </xsl:if>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates select="." mode="ditamsg:unknown-extension"/>
        <xsl:value-of select="@href"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

The reference-creating statement is actually the use of `dita-ot:generate-id` with the final target

href content

The `xref-content` is evaluated from the same template below

```
<!--template for xref-->

<xsl:template match="*[contains(@class, ' topic/xref ')]"
  name="topic.xref">
  <xsl:choose>
    <xsl:when test="@href and normalize-space(@href)">
```

```

    <xsl:apply-templates select="." mode="add-xref-highlight-at-
start"/>
    <a>
      ...
      ...
      <xsl:otherwise>
        <xsl:choose>
          <!--HSC xref content is empty or not, hence we show the
link info -->
          <xsl:when test="*[not(contains(@class, ' topic/desc '))]
| text() ">
            <xsl:value-of select="concat('Prefix:', name(),
'=')'"/>
            <!--HSX we fall into xref catching templates as we are
xref -->
            <xsl:apply-templates select="*[not(contains(@class, '
topic/desc '))] | text()"/>
            <!--use xref content-->
          </xsl:when>
          <!--HSC xref content available - use it -->
          <xsl:otherwise>
            <xsl:call-template name="href"/><!--use href text-->
          </xsl:otherwise>
        </xsl:choose>
      </xsl:otherwise>

```

which hits the available xref statements

[Continue here](#)

glossterm id

The glossterm is created directly from

```

    <xsl:template match="*[contains(@class, ' topic/topic ')]
/*[contains(@class, ' topic/title ')]">
      ...
      ...
      <xsl:element name="h{$headinglevel}">
        <xsl:attribute name="class">topic:title<xsl:value-of
select="$headinglevel"/></xsl:attribute>
        <xsl:call-template name="commonattributes">
          <xsl:with-param name="default-output-
class">topic:title<xsl:value-of select="$headinglevel"
/></xsl:with-param>
        </xsl:call-template>
        <!--HSC create id of a heading in an HTML file -->
        <xsl:attribute name="id">
          <xsl:apply-templates select="." mode="return-aria-label-
id"/>
        </xsl:attribute>
        <xsl:apply-templates/>
      </xsl:element>

```

and the section can be replaced by

```

    <xsl:template match="*[contains(@class, ' topic/topic ')]
/*[contains(@class, ' topic/title ')]">

```

```
...
...
<xsl:element name="h{$headinglevel}">
  <xsl:attribute name="class">topicitle<xsl:value-of
select="$headinglevel"/></xsl:attribute>
  <xsl:call-template name="commonattributes">
    <xsl:with-param name="default-output-
class">topicitle<xsl:value-of select="$headinglevel"
/></xsl:with-param>
  </xsl:call-template>
  <!--HSC create id of a heading in an HTML file -->
  <xsl:attribute name="id">
    <xsl:call-template name="setidaname"/>
  </xsl:attribute>
  <xsl:apply-templates/>
</xsl:element>
```

and a name tag is generated for titles.

10 Programming Stylesheets

Short description on stylesheets

Programming stylesheets is certainly not easy but we suggest a short start-up description which will allow to enhance complexity by studying good literature available.

- [\[XslTut#1\]](#) is a tutorial style - very good for the absolute beginner.
- [\[Xslt#1\]](#) is a very good nearly complete description of all xslt-functions and "the" reference for future stylesheet programming
- [\[Xslfo#1\]](#) is the full specification for the formatting-objects (fo) elements - indispensable for all who want to program stylesheets for PDF production.
- [\[svgspec#1\]](#) is the SVG specification, only required if intensive work is planned on the integration of figures.

Topics	10.1 Stylesheet Example	97
	10.2 Steps to the first stylesheet	98

10.1 Stylesheet Example

This example provides an unsorted and a numbered list and serves as an example for the first stylesheet. The task is, to output the content of those list entries, that have the attribute `outputclass=compact`.

Unsorted list This is an introduction text

- UL list entry 1
- UL list entry 2
- UL list entry 3 compact
- UL list entry 4 compact

Numbered list

1. OL list entry 1
2. OL list entry 2 compact
3. OL list entry 3
4. OL list entry 4 compact

```
<xsl:template match="li[contains(@outputclass, 'compact')] ">
  compact
</xsl:template>
```

```
<xsl:template match="li[not (contains(@outputclass, 'compact'))] ">
  not compact
</xsl:template>
```

```
<xsl:template match="concept">
```

```

        concept
        <xsl:apply-templates/>
</xsl:template>

<xsl:template match="title">
    title
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*">
    <xsl:text>unmated:</xsl:text>
    <xsl:value-of select="name()" />
</xsl:template>

```

10.2 Steps to the first stylesheet

The final layout was

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xsl xs" version="2.0">

    <xsl:template match="/">
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="concept">
        <xsl:text>we found a concept</xsl:text>
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="title">
        <!--
        <xsl:text>The title is:[</xsl:text>
        <xsl:apply-templates/>
        <xsl:text>]</xsl:text>
        -->
    </xsl:template>

    <xsl:template match="sup">
        <xsl:element name="template">
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="shortdesc"> </xsl:template>

    <xsl:template match="conbody">
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="ul">
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="ol">
        <xsl:apply-templates/>
    </xsl:template>

```

```

<xsl:template match="li[contains(@outputclass, 'compact')] ">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="li"> </xsl:template>

<xsl:template match="codeblock"> </xsl:template>

<xsl:template match="p[contains(@outputclass, 'mrg')]" priority="5">
  <xsl:apply-templates select="ul | ol"/>
</xsl:template>

<xsl:template match="p" priority="1">
  <xsl:apply-templates select="ul | ol"/>
</xsl:template>

<xsl:template match="*">
  <xsl:text>Unknown Match:</xsl:text>
  <xsl:value-of select="name()" />
</xsl:template>

</xsl:stylesheet>

```

After we have done all the work, we have to recognize, that XSL is a very powerful language. The entire job could also have been done with a few lines

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="xsl xs" version="2.0">

  <xsl:template match="/">

    <xsl:apply-templates select="//li[contains(@outputclass,
'compact')]"/>
  </xsl:template>

  <xsl:template match="li[contains(@outputclass, 'compact')] ">
    <xsl:text>&#xA;</xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>

```

but of of course, that approach is a highly efficient solution, but hardly does it suit to understand the mechanisms of XSLT.

10.2.1 DGI conversion

You have to use the transformation sheet "TcChangeALL.XSL". (TcDgi is included automatically)

- **Copy** all files to be transformed to a directory
- **Open** ONE of the files to be changed (XSLT always needs one input file although the script touches all in the directory)
- **Create** a scenario with the opened file as 'input', the TcChangeALL.XSL as transformation sheet. and any dummy output

- **Run** the scenario The system creates an "out"-folder which contains the converted files (with the same filename).

I will have a better description in the MyBook.DITA when I get it back ... the script changes any occurrence of 'xx xx' where x=0-9,a-f,A-F,x,X Darshana will be able to change TcDgi.XLS if you want an other algorithm,

11 Other Tools

Topics	11.1 Plant UML	101
--------	----------------------	-----

11.1 Plant UML

<http://plantuml.com/>

PlantUML is a component that allows to quickly write :

- Sequence diagram
- Usecase diagram
- Class diagram
- Activity diagram, (here is the new syntax)
- Component diagram
- State diagram
- Deployment diagram
- Object diagram
- wireframe graphical interface Diagrams are defined using a simple and intuitive language

see also [\[PlantUML#Title\]](#)

12 Contribute to DITA-OT

Topics	12.1 Overview	102
	12.2 GitHub	102

12.1 Overview

Check <http://www.dita-ot.org/get-involved> to get an overview what to do.

12.2 GitHub

The GitHub allows to compile the JAVA code of the DITA-OT. It is not meant to provide patches to the XSL files

13 Starting with DITA

This chapter explains how to use DITA in G&D. It will be extended whenever we have new knowledge or ideas.

Topics	13.1 Todo's in this document	103
	13.2 Why using a concept?	104
	13.3 Why using a topic?	104
	13.4 Why using a task?	104
	13.5 Why using a reference?	104

13.1 Todo's in this document

About this task

This book will be helpful for future generations of programmers and DITA users. It compresses the most important DITA tags and tactics for G&D employees and it will be a living document to catch all good ideas and helpful hints.

The main reference will always be the [\[DitaSpec#1\]](#) but that document has more the 1000 pages and should rather be used a reference than as a *good friend* to get DITA running.

This books explains DITA topics and at the same time it uses them and demonstrates their usage. So it should be a reference for coming documents.

The following is an (incomplete) list for things to cover in this book

 **Attention:** Do not write more than necessary but point out those traps that a newcomer is likely to step in. As you may consider yourself as a newcomer to DITA ... this is an excellent way to prevent others in the future (after you've been to the hard part).

 **Important:** The style may well be like "*I found that you cannot place a paragraph after steps in a task*". It is your choice of style and whatever you do is good.

Procedure

- 1) Replace the author by your name in the `MyDita.ditamap`.
- 2) Open the [\[DitaSpec#1\]](#) specification and create stubs (name= "DitaSpec") to make your links work.
- 3) Describe why/when using `topic/concept/task/reference`. Do this in a document of that type and play a bit with the types to find out the restrictions (e.g. you cannot do a paragraph within a step of a task).
- 4) Describe how to use links (`xref`), in particular describe the `keys` attribute in the `ditamap` to allow a key reference.

- 5) Describe how to create tables, play with `namest` and `nameend` and `morerows`.
- 6) Describe how to create figures
- 7) Describe how to create lists (`ul`, `sl`, `ol`, `dl`) and describe the `outputclass=compact` for a list item in those lists.
- 8) Describe how to create a `note` and show the different flavours of a note which can be applied to the `type` attribute.
- 9) will be coming



We should have a document that contains the most important aspects for starting with DITA>

13.2 Why using a concept?

explains when a concept type is to be selected

A concept shall be selected in 90% of cases - in particular if you don't know what else to choose.



13.3 Why using a topic?

A topic is an even more generic document type, it is very unlikely that you cannot do with a `concept`, but that may happen if you start to specialize a topic and you need a quick start on the base.

13.4 Why using a task?

About this task

Procedure

13.5 Why using a reference?

14 Exercises

The following exercises challenge the parameters available for the modified [DITA-OT](#)

Topics	14.1 Paragraphs	106
	14.2 Lists	106
	14.3 Figures	107
	14.4 Links	107
	14.5 Notes	107
	14.6 Index	108
	14.7 Index Test	108
	14.8 Creating the front page	109
	14.9 Changing system variables	109
	14.10 Extensions to tables	110

14.1 Paragraphs

Tryout writing paragraphs using

- `outputclass = mrg`
- `outputclass = mrg:right`
- `outputclass = compact` (you have to write two paragraphs whereas the second paragraph has "`outputclass=compact`")
- `outputclass = mrg:dialog`
- `outputclass = mrg.heading`
- `outputclass = mrg` with an `image:outputclass = align=top`. The image should have a width of 1 cm

Then manage a page to spill text over a page break. Use the `outputclass=keep` on all topics you might want to keep together. The first topic with "keep" should fall to the next page with all following topics with the `keep` attribute.

Also tryout what happens if there are other elements within a paragraph (e.g. figures, lists etc.) - Do you need to put the `outputclass=keep` attribute also on those?

14.2 Lists

The `compact` attribute was added to all kinds of list items.. Create the following lists

- **ul** with list items, some of them with `outputclass:compact`
- **sl** with list items (sli), some of them with `outputclass:compact`
- **ol** with list items, some of them with `outputclass:compact`
- **dl** with `dt:outputclass:compact`

Then try another ul with

- `ul:outputclass = checklist`
- `ul:outputclass = folder`

14.3 Figures

Explains the extensions on figures

- Verify the possible options from [\[DitaSpec#3.4.1.1\]](#)
 - `frame`
 - `scale`
 - `expanse`
- Use the image attribute `placement = break`
 - `outputclass = flow`
 - `outputclass = page`
 - `align = left | center | right`
- Check what happens on the above variants when `placement = inline`

14.4 Links

Links to exercise

- Create a link using a file reference (`href`)
- Create a link using a key reference (`keyref`)
- Give some arguments what is better using key reference or file references
- Create a link with empty content
- Create a link using the `outputclasses`
 - `see`
 - `onpage`
 - `page`
 - `pagenumonly`
- Create a link to a glossary entry
- Create a link to a glossary entry using a `conkeyref` in order to show the glossary definition.
- Create a link to an ezRead document using the ezRead notation [\[ezRead#8.3.3\]](#)

conref/conrefend

- Refer to the content of a glossary entry by a `conkeyref` (use the 'ph' trick)
- Do the same as above, but do not use `ph`, just leave the `xref` to the `glossentry` empty
- Tryout what `conrefend` is doing (write an example)

14.5 Notes

- Test notes with the type

```
note | tip | fastpath | restriction | important |
remember | attention | caution | notice | danger | warning | other
```

and verify whether you are satisfied with the chosen icons

- Test notes with the `outputclass`
 - `noLabel`
 - `noImage`
 - both of above

14.6 Index

- Play with the Indexing group elements from [\[DitaSpec#3.1.3.2\]](#)
- Test the variants and the nested variants and verify the generated index
- Try the start/end attribute in an `indexterm`

This is again security

Here comes a note



Danger: This is a note with no other `outputclass`

Danger: This is a note with `outputclass noImage`



This is a note with `outputclass noLabel`

This is a note with `outputclass noLabel` and `noImage`

Here' a paragraph

- test 1
- test 2

Table 9: test table

<ul style="list-style-type: none"> • item 2 • item 2 <ul style="list-style-type: none"> — 2.1 — 2.2 — 2.3 • atest 	
1.1adfsjakd;sf;klasdk;l k;ljasdfklasdf;lasdf;kas df;lkj	1.2
	2.2
	3.2

14.7 Index Test

security comes

14.8 Creating the front page

Exercises to change the front page

Change basic title elements

1. Change the name information in the `test.ditamap`. You will find two locations for names, only the `organizationinfo` will be visible on the front page.
2. Create the PDF to see that the second front page disappeared.
3. Redo the change by bringing back the summary
4. Change the `helmut@hscherzer.de` address (in organization info) into `helmut(at)hscherzer.de`.
5. Create the PDF and verify that the full address text on the front page disappeared.

Avoid the backside of the front page

1. delete the entire summary tag from the Ditamap
2. Create the PDF and see that page 2 disappeared

Adding trademarks

1. Add another `data-trademark` section to add a next trademark.
2. Create the PDF and verify

Use extended layout

1. Find the variable `docdesign` in `C:\ProgramData\Dita-OT2\plugins\com.ref1.pdf\cfg\fo\attrs\basic-settings.xsl` and set its value to `dsgn_smart`.
2. Create the PDF and see that the library notation is placed above the main title of the front page. There might come a time when you like to have such library categorization.
3. Redo the changes to `dsgn_gnd`.

14.9 Changing system variables

Annotations

- Find the system variables in `C:\ProgramData\DITA-OT2\plugins\com.ref1.pdf\cfg\fo\xsl\commons.xsl`
- Create a file with annotations (track changes on ... *delete/insert* text, create a comment, highlight some text)
- Create the PDF and see how the annotations are available.



Note: The change bars will only be visible if you have inserted/deleted text with track-changes "on".

- remove "change bars" from the `revmode` variable
- Create the PDF again and check whether the change bars disappeared.
- Do the same with the `show-annot` keyword and check that there will be no more PDF comments in the output

Debugging functions

1. Change the value in the `dbghs` variable from `dbg_NoShowLabels` → `dbg_ShowLabels`
2. Create the PDF and check noise in the layout but also some new fields in the headers and footers of running pages. These names are the
3. Change the value in the `dbghs` variable from `dbg_NoMarkFields` → `dbg_MarkFields`. On the PDF you will experience gray fields in the headers who show you the editable size of the headers/footers until they break into two lines.
4. Bring the value back to the old "no" key words to avoid the debugging features.

Figure/Table numbering

- Remove (or modify to a non-defined value) the values of the `EnumerationMode`
 - `tbl_prefix` → `tbl_Noprefix`
 - `xbl_numrestart` → `tbl_numrestart`
 - `xig_prefix` → `fig_prefix`
 - `fig_Nonumrestart` → `fig_restart`
- and check how the table captions change. The prefix will add the current chapter number (e.g. Table 4-2) and the restart value will let the numbering restart with every chapter (which only makes sense if you are using the prefix)

14.10 Extensions to tables

- Assign `outputclass=rowcolor` to a fixed verb (*yellow* or *red*)
- Assign `outputclass=rowcolor` to a value `#1280FF`
- Test tables with the `morerows` attribute
- Test tables with the `namest` attribute
- Test `colsep/rowsep` on `table/row-col/entry` level to see which of the separators goes when you assign '0'.

Appendix A - Bibliography

Table 10: Bibliography

BibEnt	Description	Publisher
[AHF]	Antenna House Formatter V6 User Manual	Antenna House
[Ant]	Apache Ant 1.8.1 Manual	Apache
[DitaSpec]	Darwin Information Typing Architecture (DITA) Version 1.2	OASIS
[DtPrt]	DITA for Print: A DITA Open Toolkit Workbook DITA Open Toolkit 1.8, Leigh W. White http://xmlpress.net	XML Press
[ezRead]	ezRead Documentation System  umentation Guide, Version 2.65	Helmut Scherzer
[ISO4]		
[oxy17]		
[PlantUML]		
[svgspec]		
[Xslfo]		
[XslTut]		
[Xslt]		

Glossary

DITA-OT	DITA Open Toolkit A set of files required to process DITA files into any format. The DITA-OT is an open source project, as of 20150804 it is distributed as version 2.0.1.
Forty-Two	Forty-Two is an executable program installed on G&D PCs. It allows temporary administrator rights. The full developer rights allow instant use of admin rights whereas the restricted version requires a challenge-response pair (given by the local IT department) in order to achieve the administrator rights.
Oxygen	oxygen XML editor
PNG	bla bla
SVG	Scalable Vector Graphic

Index

S

security

nesting level 2 **108**

sub 1

sub2 **55**

test nestin 3 **108**