



Giesecke & Devrient



Dita book Library

DITA for High End Documentation

Reference Guide for the xPower Dita-OT plugin

Version 6.2.1

Printed on 10th April 2016



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

Table of Contents

Tables v

Figures vi

About xPower viii

1 Installing the DITA environment 10

1.1 Quick Installation 11

1.2 Installing the command line interface 11

1.3 Installing the DITA toolchain 13

1.4 oxygen installation 14

1.5 oxygen configuration 17

1.6 Installing a plugin 19

1.7 ezRead Installation 19

2 Processing the DITA-OT 20

2.1 Using the command line 20

2.2 Running DITA with command line 21

2.3 Configuring oxygen for DITA-OT 22

2.3.1 Additional parameters 32

2.3.1.1 Default figure link text 32

2.4 Running oxygen scenarios 33

2.5 How DITA is processed... 35

2.6 PDF post processing 36

3 Important Topics 38

3.1 Using figures 38

3.2 Basic figure aspects 39

3.2.1 image-width 40

3.2.2 Wrap text around figures 40

3.3 Working with the Glossary 40

3.3.1 Creating a local glossary from a master list 41

3.4 Working with equations 41

4 DITA-OT extensions 42

4.1 Extensions on the Paragraph. 42

4.2 Extensions on Section 43

4.3 oxygen Annotations 44

4.4 Extensions to Tables 45

4.4.1 Repeat table header 46

4.5 Extensions to fig/image 47

4.5.1 Creating Figures From Visio 47

4.5.2 Links within graphics 50

4.5.3 Tryout figures 53

4.6 Extension to Links 56

4.6.1 Linking figures and tables	57
4.6.2 Trying out links	58
4.6.3 ezRead auto-linking	60
4.6.4 ezRead Main-Book flag	62
4.6.5 Manual Named Destinations	62
4.7 Extensions to Notes	63
4.8 Extensions to Lists	64
4.9 Extensions to Mini-Toc	65
4.10 Page Control	66
4.10.1 Landscape page format for sub-chapters	67
4.10.2 Page Heading Control	69
4.10.3 Newpage Enforcement	69
4.10.4 Keep Lines Together	70
4.10.5 Extension on the Title Element	70
4.11 System Variables	71
4.12 Equations	72
4.12.1 Using embedded MathML	73
4.12.2 MathML within XSL-FO	74
4.13 CHM extensions	75
5 Managing the front page	76
5.1 First Page Layout	76
5.2 Company Logo	76
5.3 Security Class	78
5.4 Watermark	78
5.5 Front Picture	79
5.6 Second Page layout	80
6 Author's support	81
6.1 Creating a Bibliography	81
6.1.1 Creating a Master Bibliography	81
6.1.2 Referencing external documents (Bibliography)	82
6.1.3 Generating a Local Bibliography.	83
6.1.4 Ignore Lists	84
6.1.5 Creating the oxygen scenario	85
6.1.6 Repair Master Bibliography	86
6.2 Creating a Glossary	86
7 MS-Word Docx 2 Dita conversion	87
7.1 Installing Docx2Dita	87
7.1.1 Style mapping	87
7.1.2 Create docx2dita scenario(s)	87
7.2 Running the docx2dita conversion	91
7.2.1 Prepare the DOCX for conversion	91
7.2.2 Converting docx2dita from oxygen GUI	94
7.2.3 Converting from the command line	94
7.2.4 Post Processing	95
8 Docbook to Dita conversion	96
9 Other Tools	97

9.1 Plant UML 97

Appendix A - Bibliography 98

Terms and Abbreviations 100

Index 101

Tables

Table 1: ANT parameters 27

Table 2: Example using compressed rows 46

Table 3: Row colors 46

Table 4: Target 1 56

Table 5: Table in note 60

Table 6: Table in list 60

Table 7: Table in normal text flow 60

Table 8: Example for the linktext construction 61

Table 9: Example for the !~[...] construction 61

Table 10: Bibliography 81

Table 11: Bibliography 98

Figures

Figure 1: Finding the command prompt for installation	12
Figure 2: Copy command prompt to desktop	13
Figure 3: Copy the topic to the desktop	13
Figure 4: Fonts settings	17
Figure 5: Locate the DITA-OT	18
Figure 6: Disabling network requests	19
Figure 7: Setting the external DITA-OT (Example)	23
Figure 8: Duplicate the existing DITA Map PDF transformation scenario	24
Figure 9: Basic settings	25
Figure 10: Filter Settings	26
Figure 11: Parameters settings	27
Figure 12: Advanced settings	29
Figure 13: Library settings	29
Figure 14: Adding JAVA libraries	30
Figure 15: Output settings	31
Figure 16: Edit the args.figurelink.style	32
Figure 17: Empty xref options	33
Figure 18: Progress log during oxygen-scenario	34
Figure 19: Result window	34
Figure 20: Log window context dialog after right mouse click	35
Figure 21: Dita Process	36
Figure 22: Test figure	38
Figure 23: Test figure	39
Figure 24: Testing a formula (export as SVG)	41
Figure 25: Preview references in oxygen	45
Figure 26: Selecting Margins	48
Figure 27: Setting margins	48
Figure 28: Set Page width	49
Figure 29: Set "Fit to Drawing"	49
Figure 30: Set "Fit to Drawing"	49
Figure 31: Set "Fit to Drawing"	49
Figure 32: Plain figure - no width specified, the frame cannot be determined from the image size	53
Figure 33: Plain figure, image:width=50mm	54
Figure 34: fig:expanse=page, image:placement=break, width=50mm (ignored by expanse=page)	54
Figure 35: fig:expanse=column, image:align=right, width=50mm, placement=break	54
Figure 36: image:outputclass=page, placement=break	54
Figure 37: image:outputclass=flow, align=right, width=50mm, placement=break	55
Figure 38: image:width=50mm, align=left, placement=break	55
Figure 39: image:align=right, width=50mm, placement=break	55
Figure 40: image:align=right, width=50mm, but placement=inline (inline does not obey 'align')	56
Figure 41: Image with #hstarget1..4	56
Figure 42: Configuring mini-TOC in runtime parameters	66
Figure 43: Example for a correct sysvars.dita	71
Figure 44: Front page definition with logos	77
Figure 45: Security Class definition in Author mode	78
Figure 46: Watermark specification in the front page	79
Figure 47: Front picture definition in oxygen author mode	80
Figure 48: Creating a local bibliography	83
Figure 49: New scenario	85
Figure 50: Select scenario type	85
Figure 51: Create Scenario	86
Figure 52: All scenarios view	88
Figure 53: Open oxygen scenarios	88
Figure 54: Change the scenario options	89

Figure 55: Edit Parameters 90

Figure 56: Edit output parameters 91

Figure 57: Select the 'styles' context menu 92

Figure 58: Apply styles 93

Figure 59: Style Dialog 93

About xPower

This is the user guide for the xPower Plugin package. The user guide is a growing document and it is not yet on the professional quality level as I am used to produce, e.g. the index is not yet correctly established because I have to add `indexterm` statements into the chapters yet.

- Maturity* From time to time there might be only some remark where I will accomplish the chapters later. Nevertheless this user guide is already a very rich documentation and at about 100 pages, certainly more than other's packages documentation provides you with.
- Most of my freetime energy went into the reliable and powerful layout of the plugin. So please give me some time to increase the quality of this user's guide ☺.
- What it does ...* This xPower package offers a very powerful extension to the official DITA-OT. It solves many open points in the official distribution. The package delivers an entire DITA-OT installation based always on the latest available release. "X"power wants to convey the spirit of "extreme", "extended", "XML" and term "power" simply tries to imply that this is quite a large and rich package solving about 90% of the professional demands of industrial high quality documentation.
- How it started* The idea of the package came from our own demand to high quality documentation in my company Giesecke & Devrient (a German money printing and SmartCard producing company → <http://www.gi-de.com>) but the package was entirely developed on my private time.
- The logos in the user manual are there to demonstrate that the package supports up to two companies in the front page - it does not mean to say that G&D or IBM was actually involved in the creation - although I used to work in IBM for 20 years and at Giesecke & Devrient until today.
- Impact ...* Some minor changes had to be done in the `org.dita.pdf2` directory to fix a few DITA-OT bugs, all other extensions are implemented in the `org.dita-community.xpower` plugin, not in the `org.dita.pdf2`.
- I also commented my extensions (HSX) to make them more understandable.
- That's why this package does not simply come as one "plugin" directory.
- ezRead Plugin* The package covers a test installation of the full *Acrobat ezRead plugin* which allows necessary PDF post processing of DITA files to get out proper PDF files that the formatters XEP / AHF cannot produce.
- Installation* Due to its many new features and extensions the work has about doubled the size of the DITA-OT but it does still follow strictly the idea of the DITA-OT and will in the future.
- To achieve xtreme editorial performance the xPower package is quite rich, therefore the installation has been made "1-button"-easy.

WHAT YOU NEED

The executable tools are only available for Windows. If someone wants to bring them to Linux, I would cooperate with the source code. The documentation explains oxygen editor and AHF Formatter, however the package does not require any of these for its tools. (except that the oxygen annotations like comments/track changes are processed into PDF ... a feature that is only available with oxygen processing-instructions).

TO UNDERSTAND:

Read the BOOKS\REF_xpwr_xPowerUserGuide.pdf

TO INSTALL:

Launch 01_ExtractE.bat

TO ASK:

Contact helmut.scherzer@gi-de.com

TO BUY?

There is nothing to buy, the package is offered for free. Currently the ezRead Plugin is available as test version which corresponds to a "professional" version that I plan to sell somehow from 2016.

- Free ezRead* When the "professional" version will be available for purchase, I will deliver a free "light" version with this package which will allow to do everything required to process the actions required to get high professional DITA output. For the promised performance output of this package you will not depend on the "professional" version's additional features.
- Copyright* The package is open source, but the copyright for the free documentation and the ezRead plugin stay with Helmut Scherzer.
- Know more ...* If you want to dive into the magic of Dita-OT extension, you might consider purchasing "Dita for Print" from Leigh W. White. → <http://shop.oreilly.com/product/9781937434274.do>
- Her book gave me a perfect start and even if I meanwhile have gone to another dimension of DITA-OT programming ... without that great book I don't think I'd ever been able to create this package. In many positions of my extension you will find references to her book that indicate the chapters from which you can learn about the associated magic.
- No change without change* The package was created to allow you creating very sophisticated PDF output. In a time of information saturation it becomes more and more important to provide smart and understandable documentation (... the idea of "ezread").
- At the same time the old statement is still valid ... "A fool with a tool is still a fool". No tool can take away the author's responsibility to use it right. If you don't create `indexterm` statements, your document won't have an index.
- My wishes ...* Have a successful and highly professional output with features that no documentation has ever seen before. (Promised !) - and "May the Force be with you ..."
- Helmut Scherzer
(Creator of the xPower package)

1 Installing the DITA environment

1.1 Quick Installation

1.2 Installing the command line interface

1.3 Installing the DITA toolchain

1.4 oxygen installation

1.5 oxygen configuration

1.6 Installing a plugin

1.7 ezRead Installation

11

11

13

14

17


19

19

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	<p>There are a couple of batch files to help processing in various situations.</p> <div><p>As an example ... to create a new DITA file, there is a batch file <code>newDita.bat</code> which copies a working set of a DITAMAP into the current directory. This template can be used and processed immediately without further editing.</p></div>
Install XML editor (oxygen)	<p>To write DITA files, an XML editor is required. The present toolchain supports the <code>oxygen</code> 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 <code>SernaFree</code>.</p> <p>Send a request to Helmut Scherzer to obtain this alternative.</p>
Install AHF formatter	<p>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.</p> <p>For single users, a stand-alone AHF (named license) is available for about \$1250</p>
Install the DITA-OT	<p>The DITA-OT (Dita Open Toolkit) is a open-source set ot 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.</p>

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 Quick Installation

The package comes with an "all-inclusive" installation process realized by

```
01_ExtractE.bat
```

This will install the entire xPower package including all tools and registry entries.

The package does (of course) not install

- any XML editor (e.g. oxygen)
- any XSLFO formatter (e.g. Antenna House Formatter or XEP renderer)

In order to understand the magic behind the quick installation `01_ExtractE.bat` the following chapters explain more details about the setup of the system

1.2 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 Pograms** → **Accessories** → **Command Prompt**

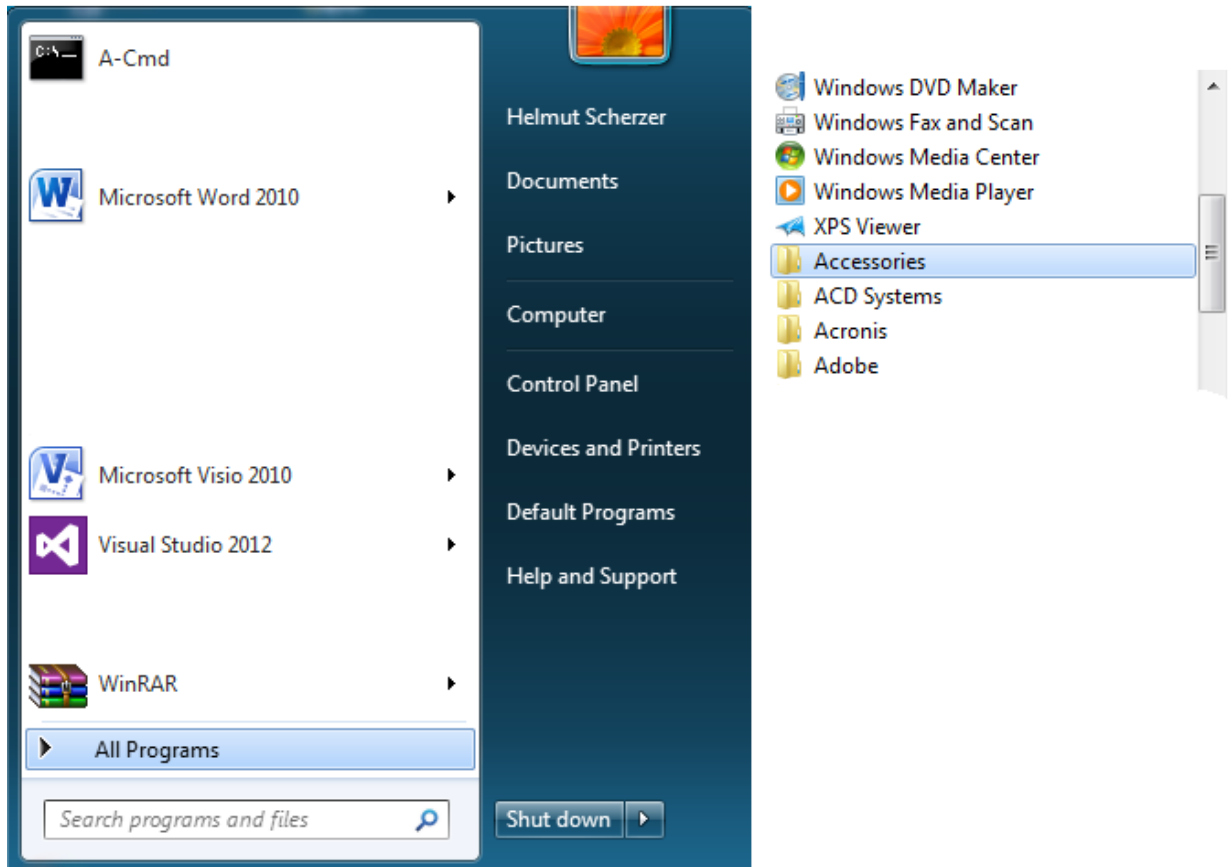


Figure 1: Finding the command prompt for installation

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

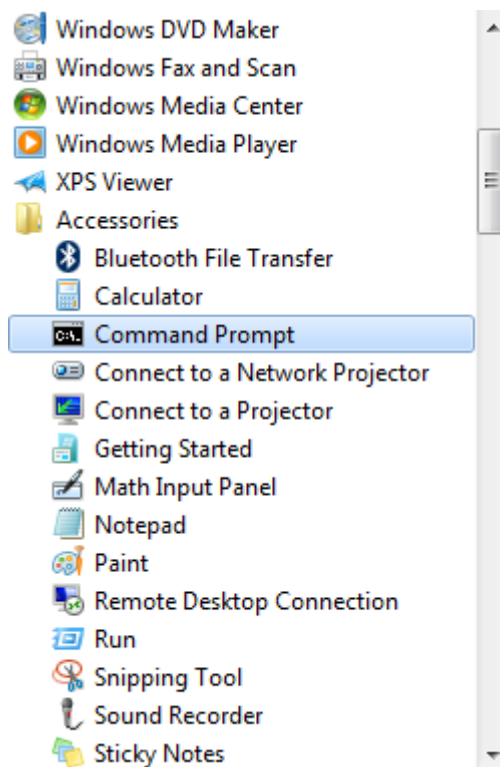


Figure 2: Copy command prompt to desktop

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

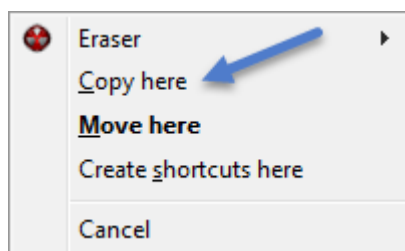


Figure 3: Copy the topic to the desktop

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

1.3 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.

1.4 oxygen installation

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

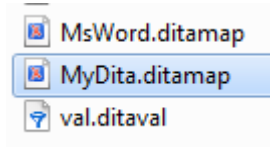


Attention: Do not forget to associate `.ditamap` with oxygen, this is not done automatically during [Oxygen](#) installation.

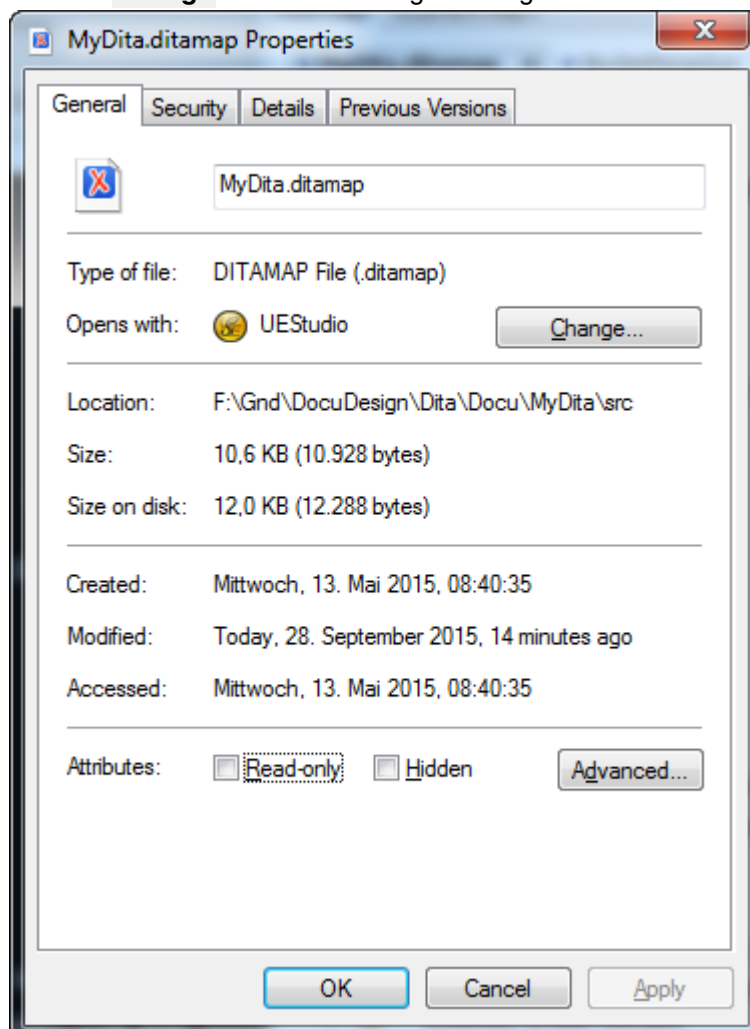
Associate file types (e.g. `.ditamap`) to oxygen

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](#).

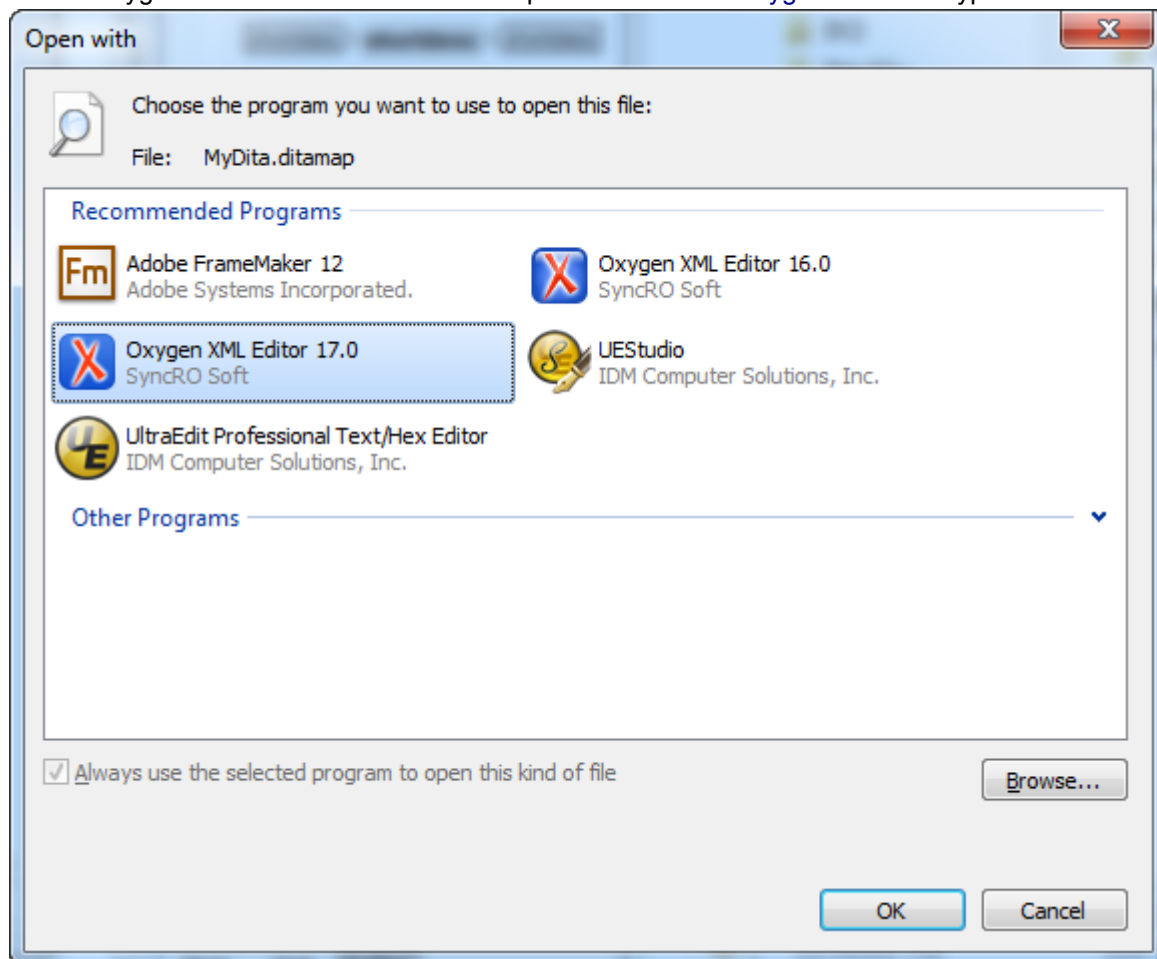
1. Open the Windows™ file explorer and navigate to any such file (e.g. `.ditamap` that should be associated.



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.

1.5 oxygen configuration

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

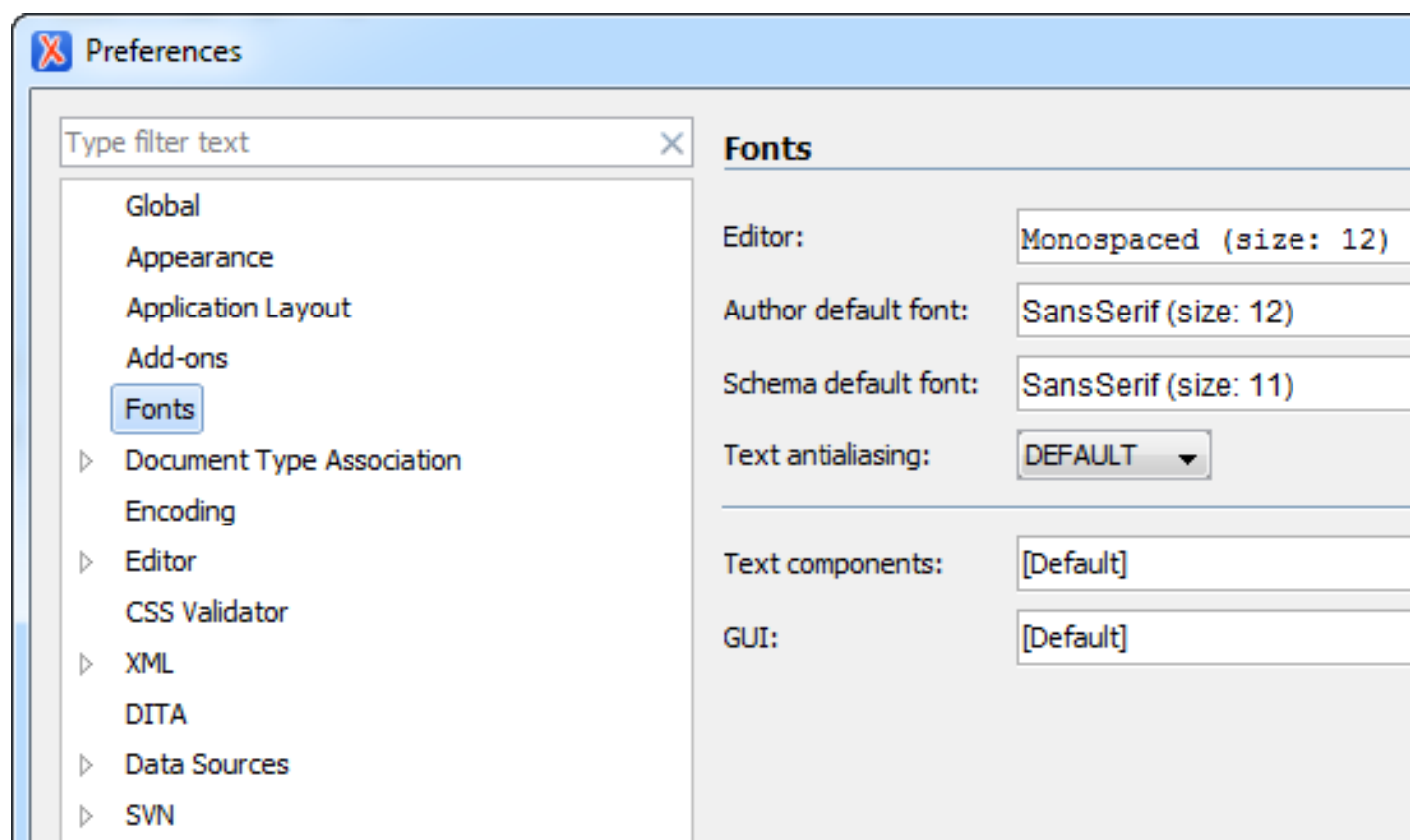


Figure 4: Fonts settings

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

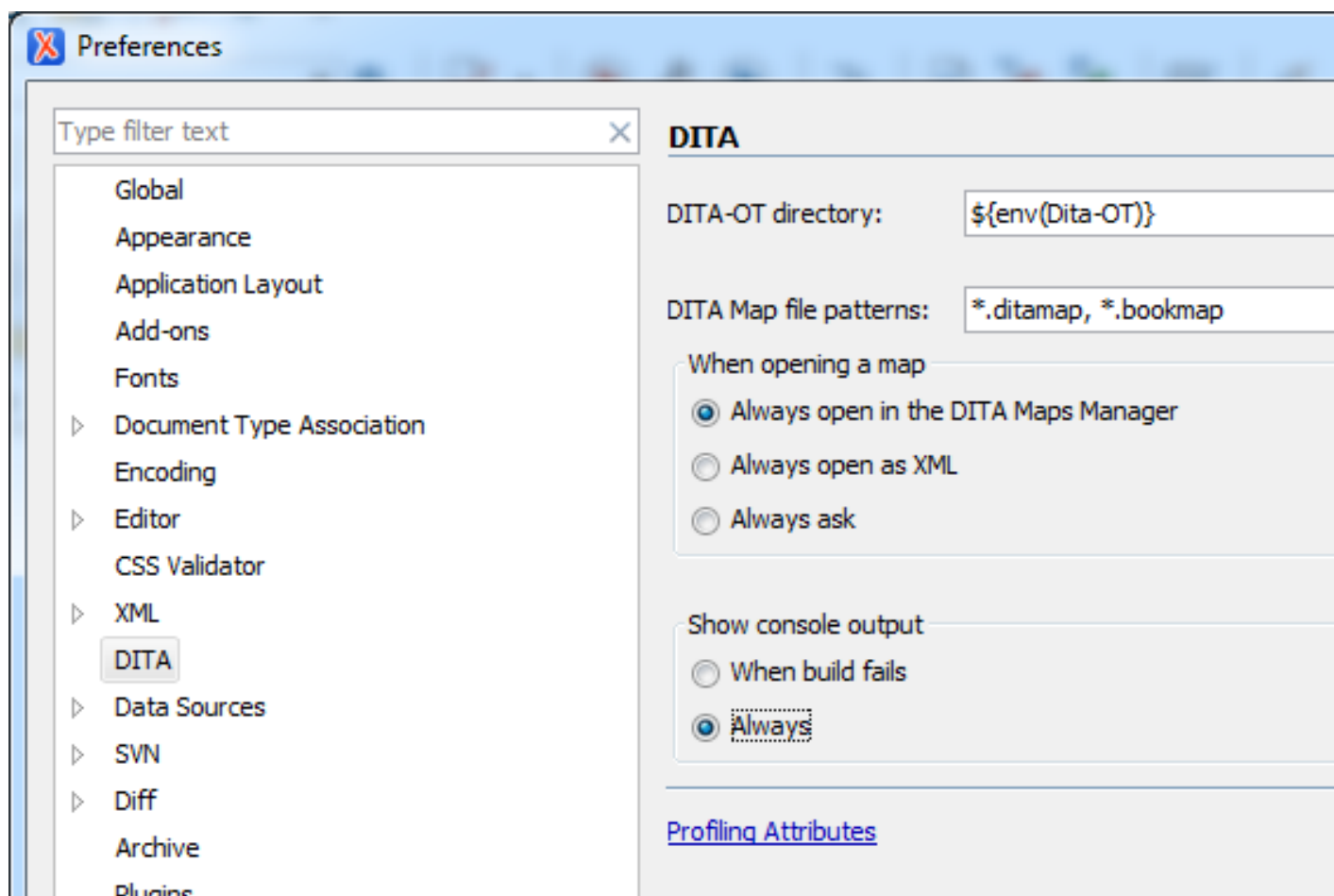


Figure 5: Locate the DITA-OT



Note: According to [Figure 5](#) 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.

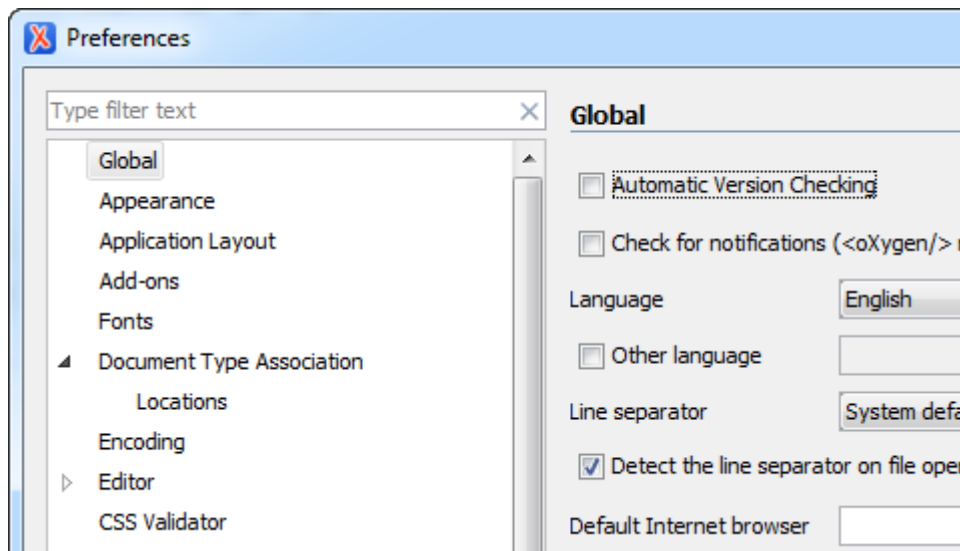


Figure 6: Disabling network requests

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

1.6 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*.

1.7 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	20
	2.2 Running DITA with command line	21
	2.3 Configuring oxygen for DITA-OT	22
	2.4 Running oxygen scenarios	33
	2.5 How DITA is processed... ..	35
	2.6 PDF post processing	36

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. <ul style="list-style-type: none">• 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 is used in order to make relative pathes 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 unless their path is given as a relative path → [AHF#use-launch]. On click - this triggers the Internet Browser instead of opening the PDF directly. That behavior can be fixed with the *ezRead Dita post processing* conversion function [ezRead#12.1.20.1].



Note: Find some more explanation on post processing in [2.6 PDF post processing](#)

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

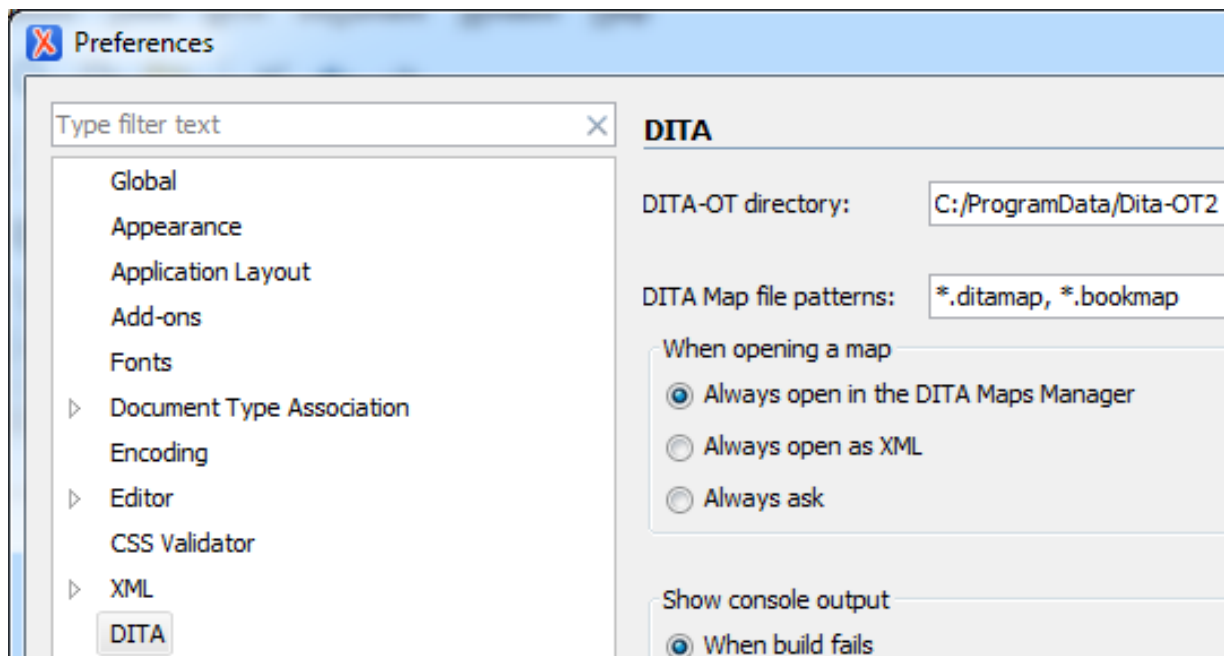


Figure 7: Setting the external DITA-OT (Example)

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

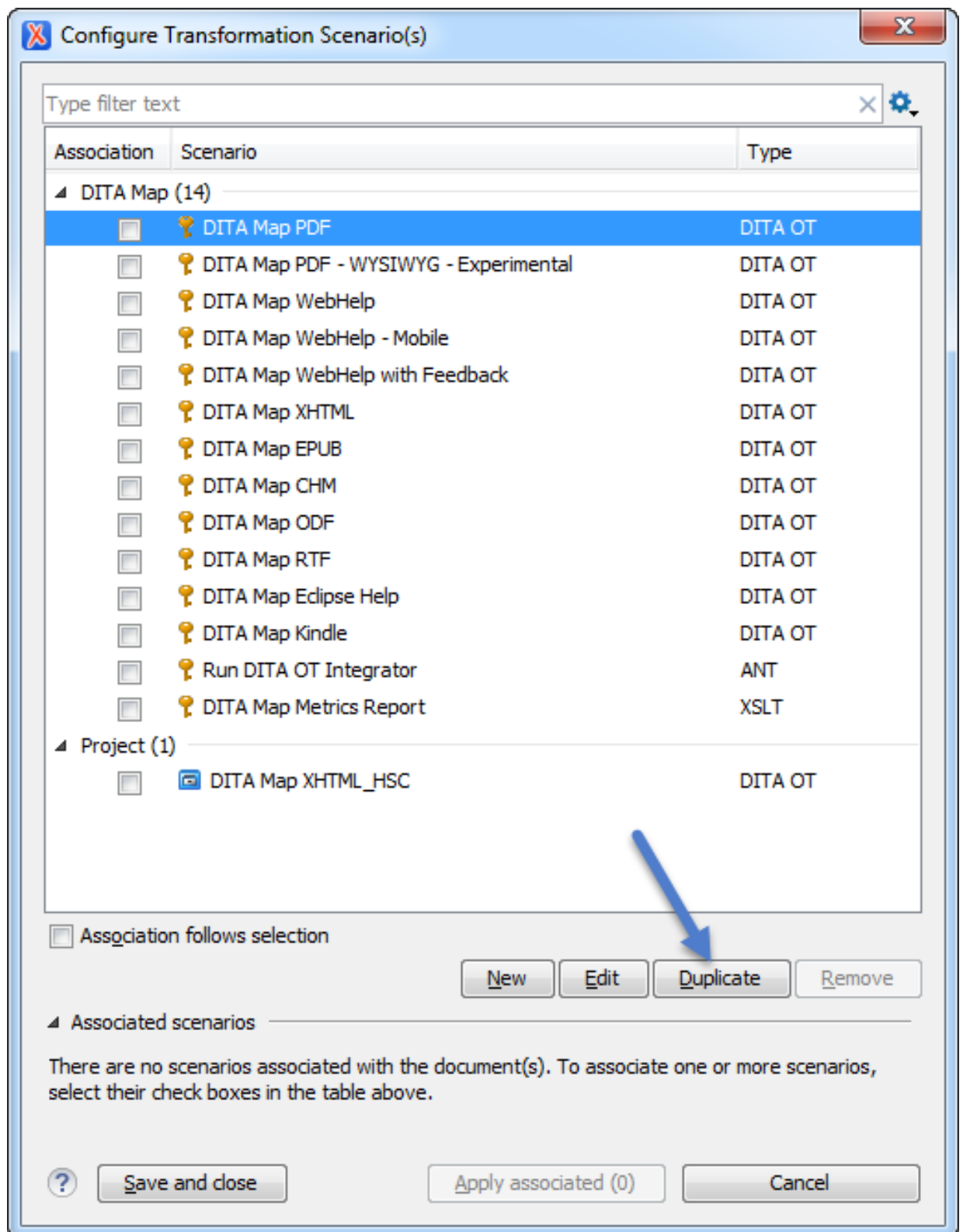


Figure 8: Duplicate the existing DITA Map PDF transformation scenario

After the duplication you will see the new transformation scenario

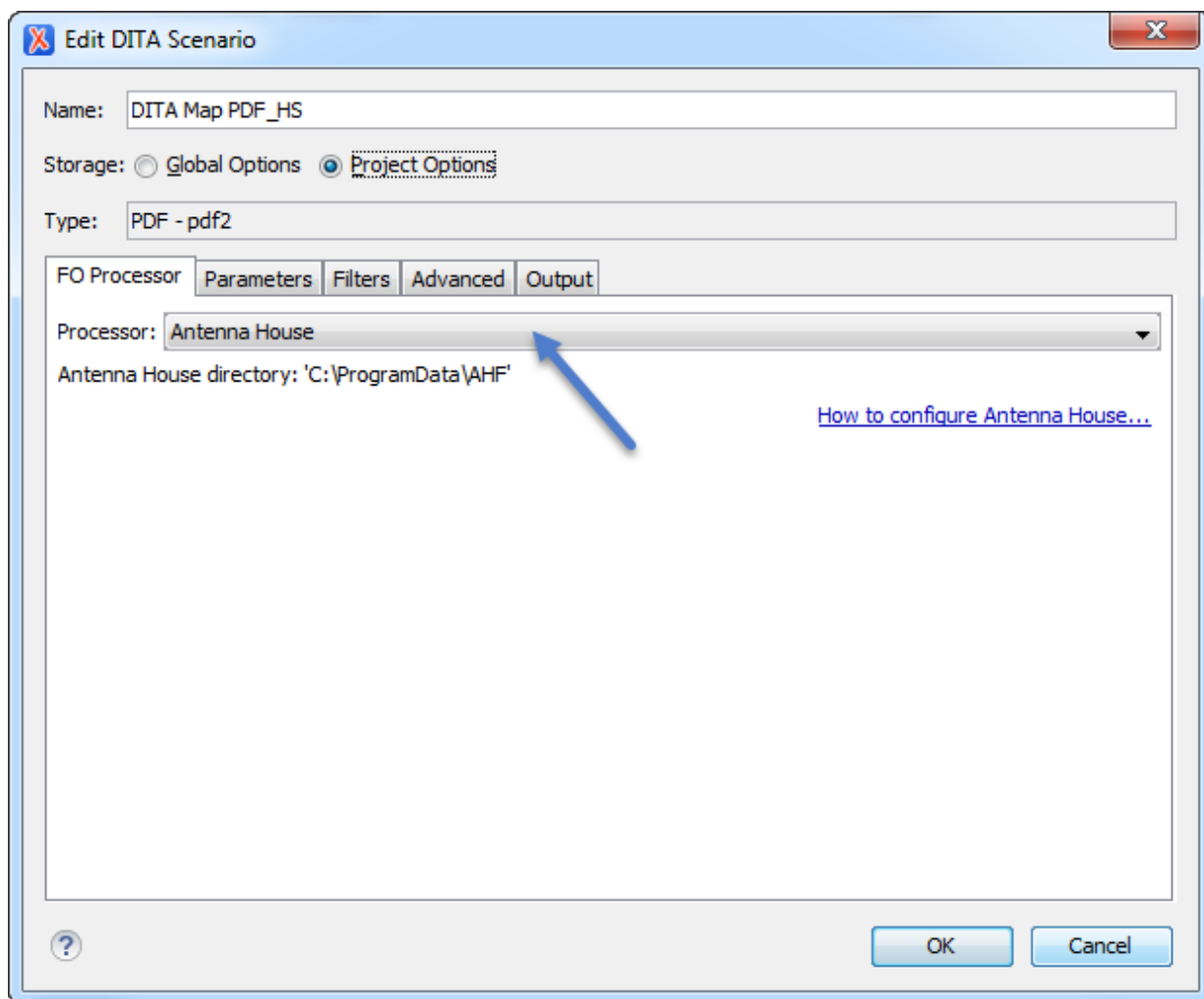


Figure 9: Basic settings

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

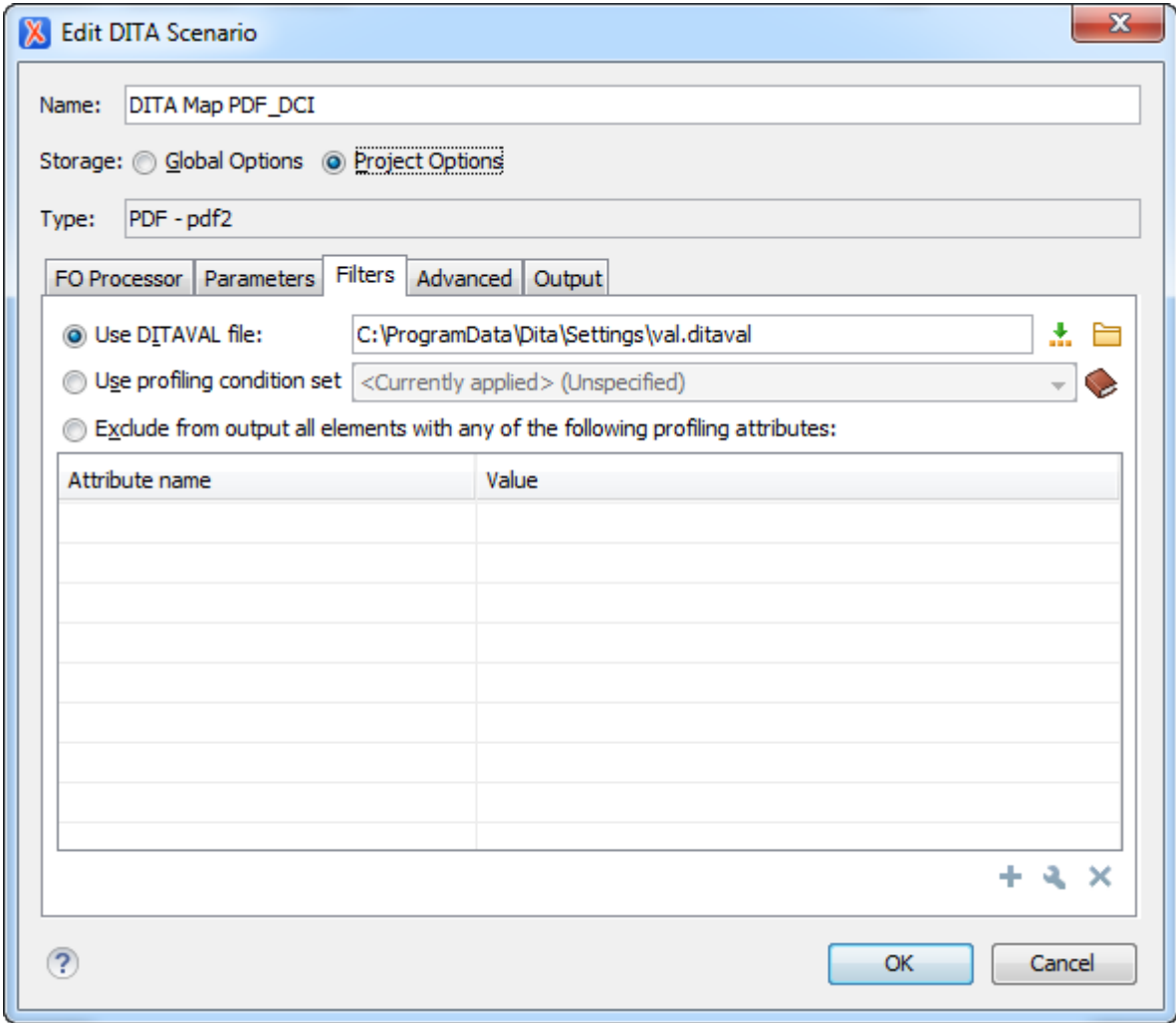


Figure 10: Filter Settings

7. Goto the Parameters tab

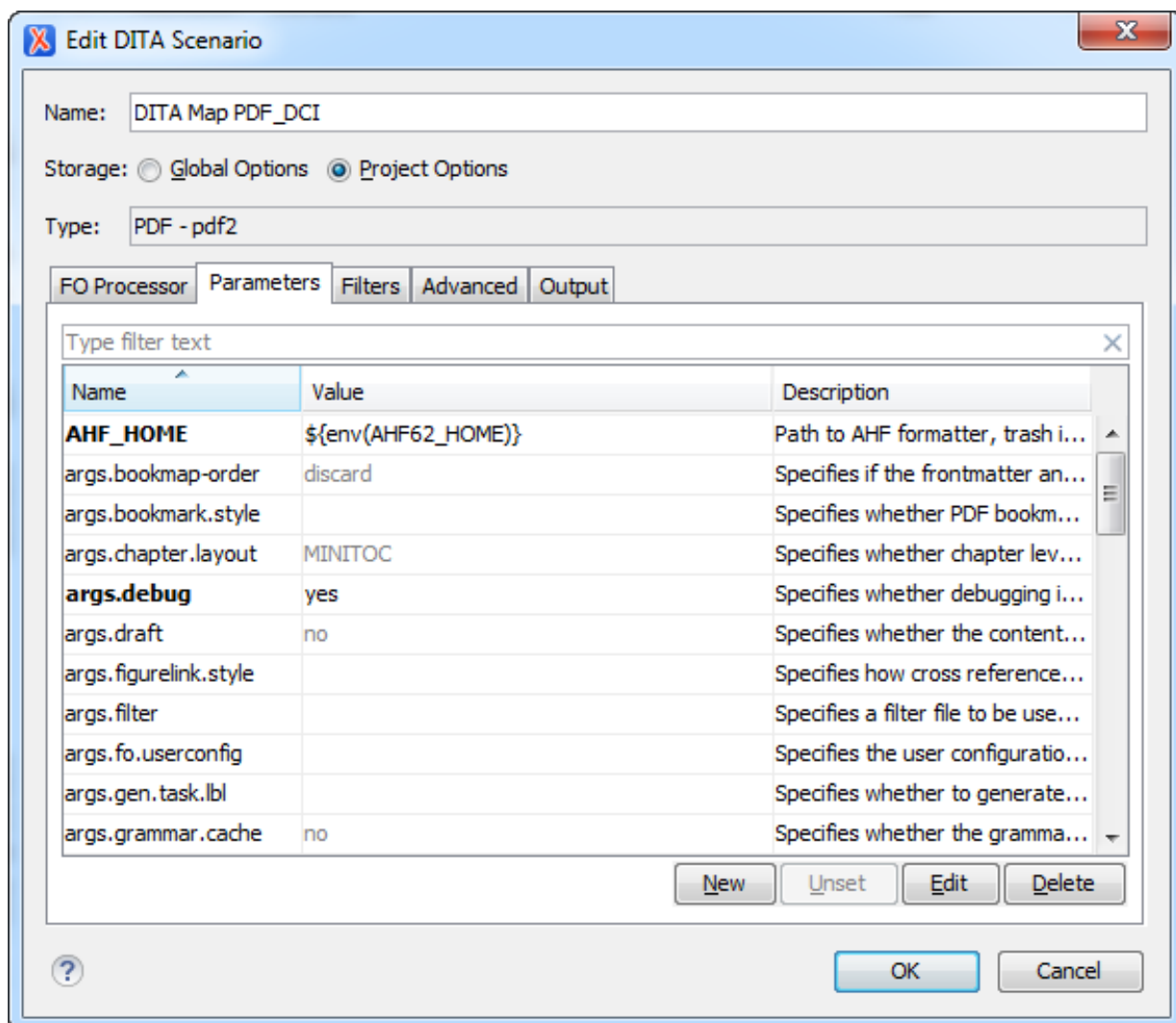


Figure 11: Parameters settings

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

Table 1: ANT parameters



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

Table 1: ANT parameters

Parameter	Value
dita.dir	<code>\${configured.ditaot.dir}</code> Alternatively: <code>\${env(Dita-OT)}</code>
dita.temp.dir	<code>\${cfd}/../build/temp</code>
env.AXF_OPT	<code>\${env(AXF_OPT)}/AHFSettings.xml</code>
 Note: You must create this entry with New	
gfxPath	<code>\${cfd}/gfx</code>
output.dir	<code>\${cfd}/../pdf</code>
org.dita.pdf2.use-out-temp	true
pdf.formatter	ah
ProjDocRel	<code>\${env(ProjDoc)}</code>
retain.topic.fo	yes
transtype	xPower

9. Go to the **Advanced** tab

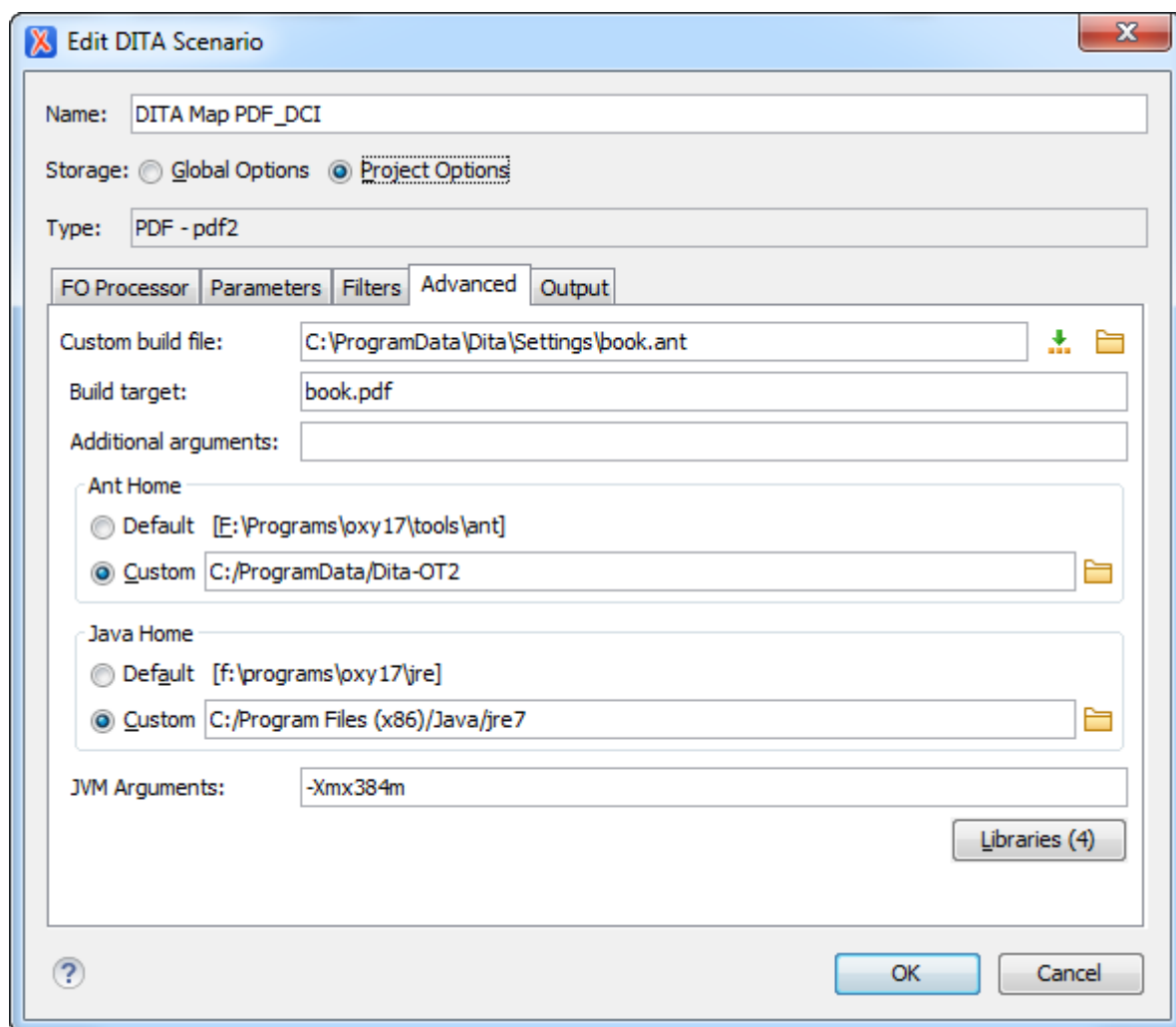


Figure 12: Advanced settings

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

10., click the **Libraries** button

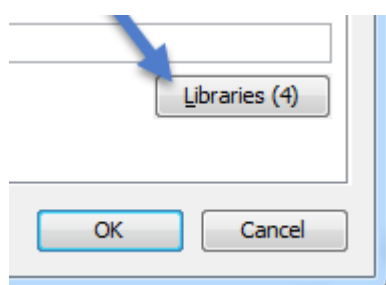


Figure 13: Library settings

and there will be a panel suggesting a lot of default libraries.

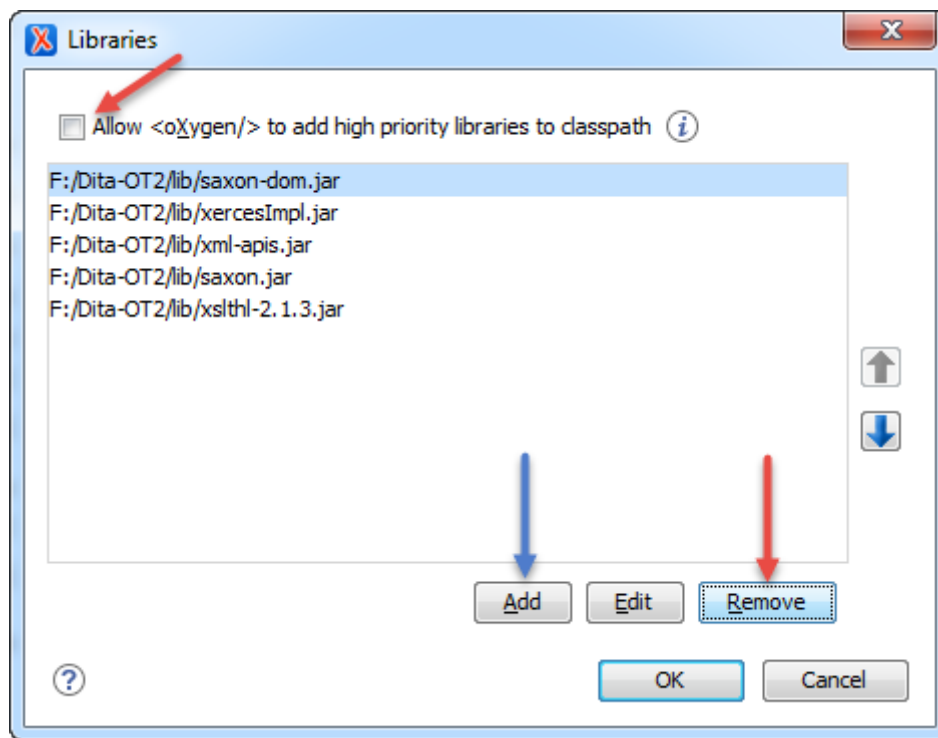


Figure 14: Adding JAVA libraries

- 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](#).



Important: You might not find the exact names as in the above image because the might had a *version suffix*. In this case you may either add the libraries by their name or (recommended) copy-rename them to the plain names as suggested above.

11. Go to the **Output** tab

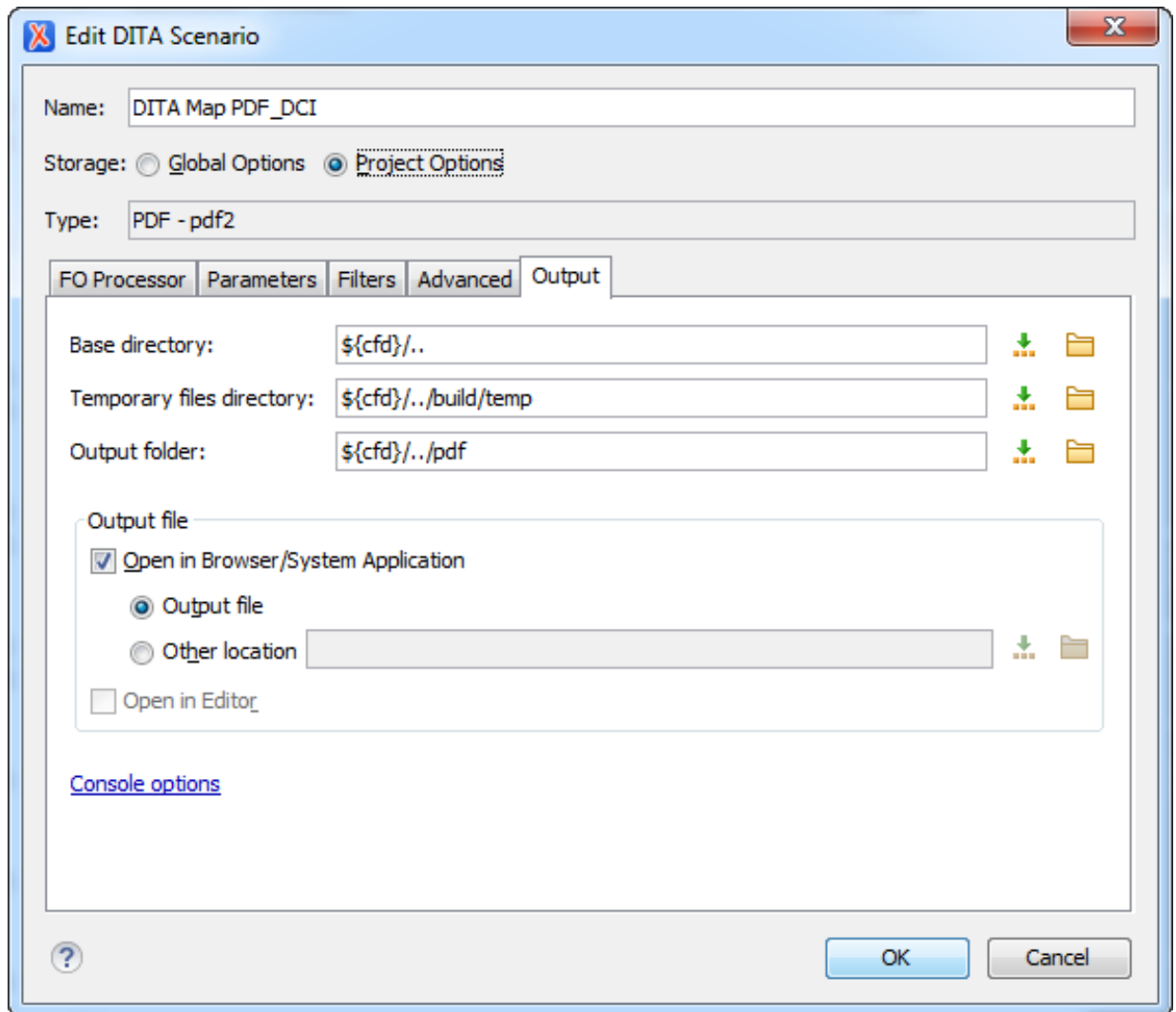


Figure 15: Output settings

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

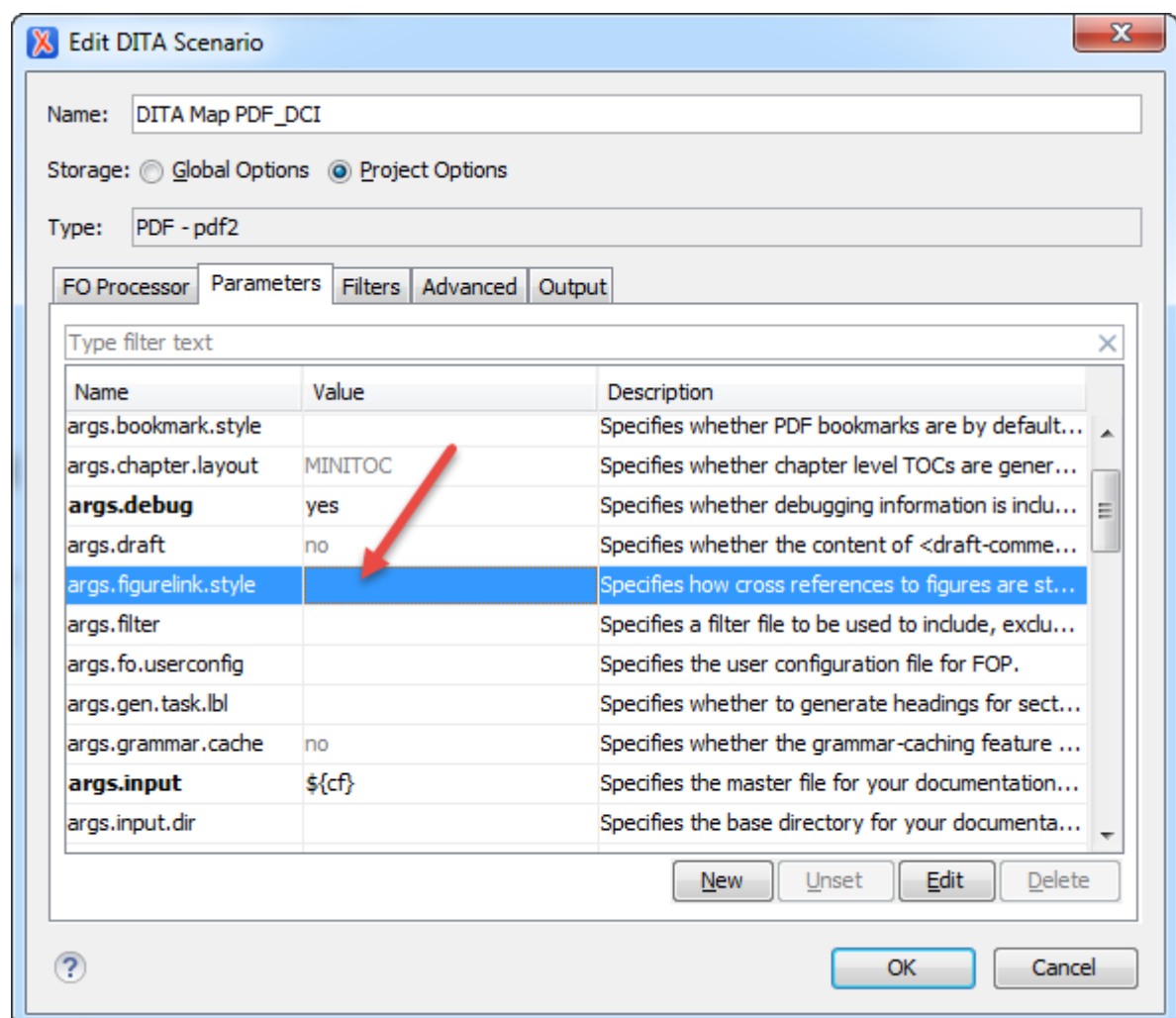


Figure 16: Edit the args.figurelink.style

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

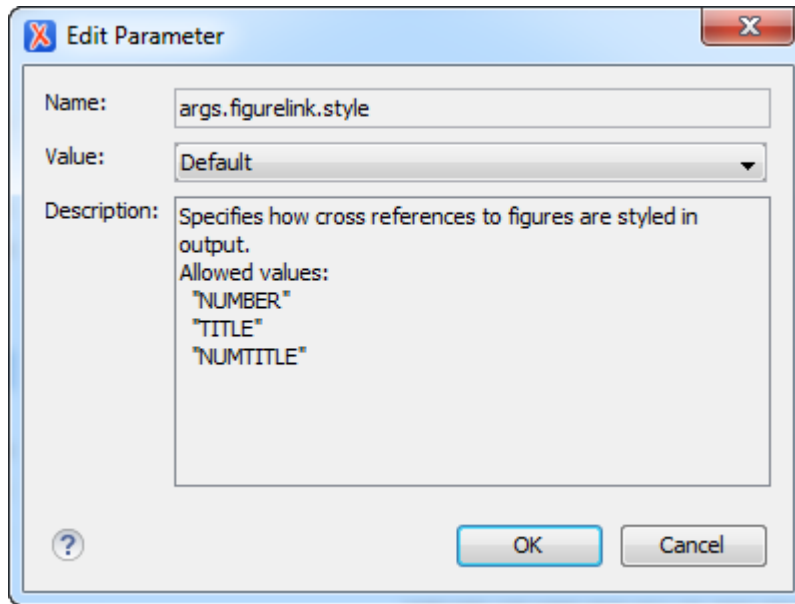


Figure 17: Empty xref options

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 [4.6.1 Linking figures and tables](#)

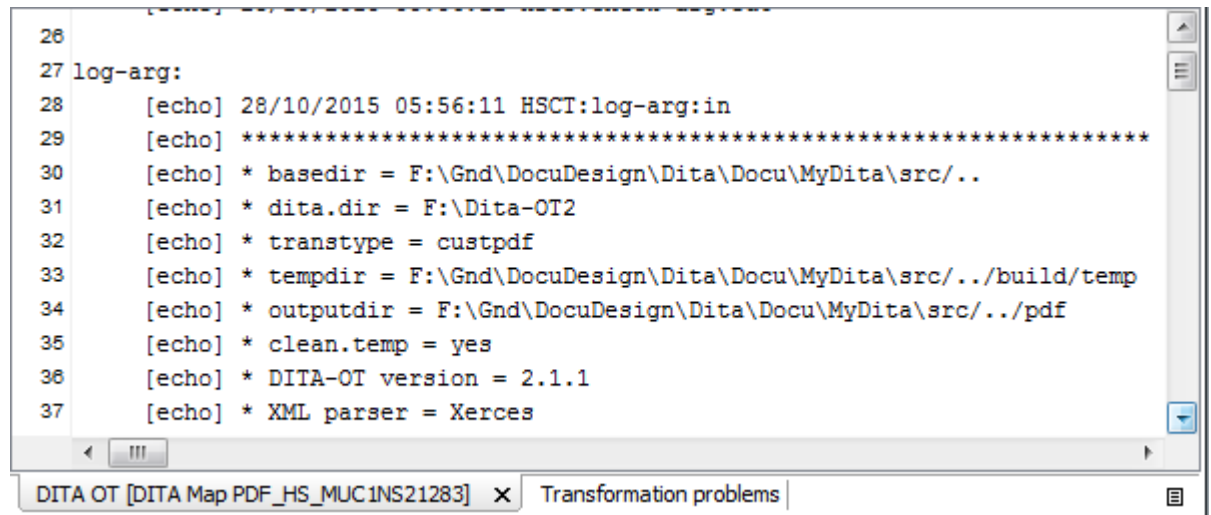
2.4 Running oxygen scenarios

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
  
```

DITA OT [DITA Map PDF_HS_MUC1NS21283] x Transformation problems

Figure 18: Progress log during oxygen-scenario

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

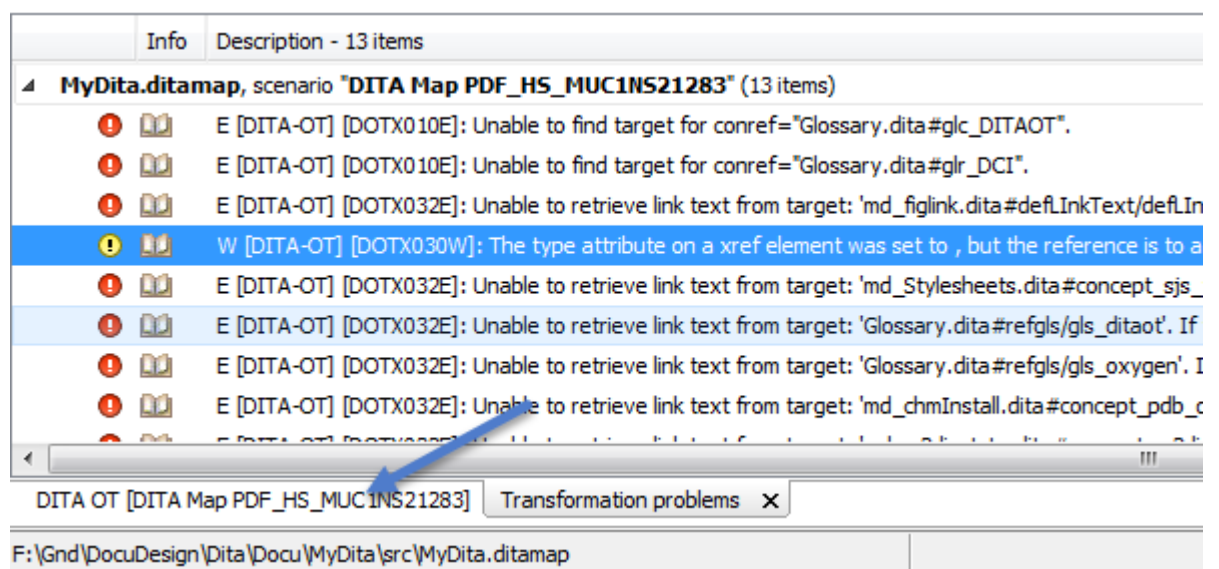


Figure 19: Result window

- 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 19](#)

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

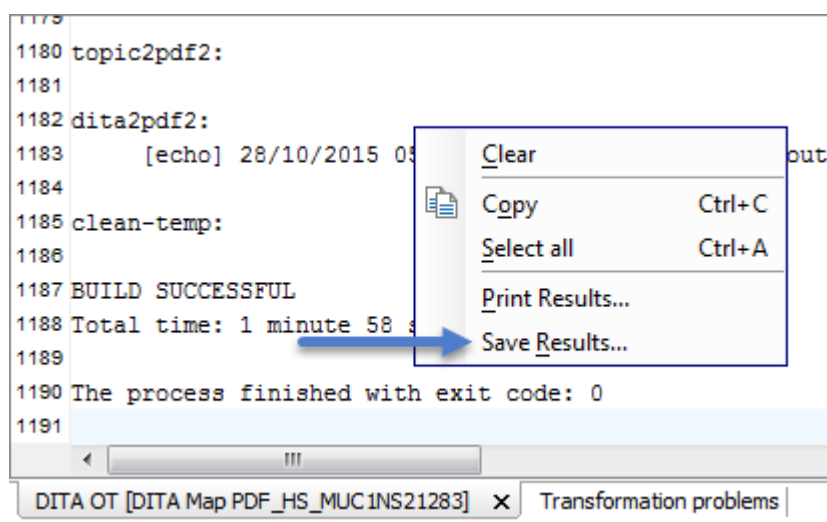


Figure 20: Log window context dialog after right mouse click

- 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 [1.5 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`.

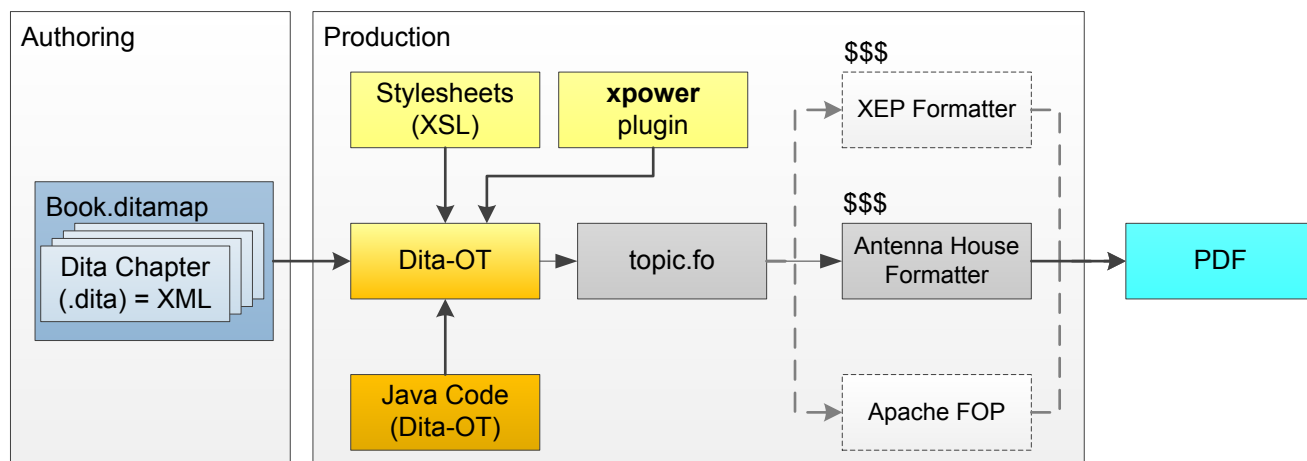


Figure 21: Dita Process

The following stages occur:

- **Authoring:** The author writes the "book" with the usual DITA statements.
- **Production:** The new book is processed by the DITA-OT with the output `topic.fo` which is well described in [DTPrt#3.9.1]. This intermediate file `topic.fo` is also an XML file with all formatting information required to produce a PDF. The associated language is described in [XslFo]. It requires a *formatter program* to do all the computations for the PDF.

The present system works best with the Antenna House Formatter which is commercially available at around \$1250. Optionally other formatters are available

- The XEP Formatter is JAVA-based and does not have as many features as the Antenna House Formatter (at the time of the last investigation approx. 2015), The price is around \$500.
- The Apache FOP formatter is a free formatter delivered with every Dita Open Toolkit distribution. It has, however, the disadvantage that a log of features are not available.

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

Semi-optimal bookmarks from Antenna House Formatter

Antenna House Formatter produces semi-correct bookmarks. In the PDF, the bookmarks do work properly. Using the [ND.API](#) plugging revealed, however, that the [collapsing function](#) did not work because the bookmarks are not created perfectly.



Note: The exact technical detail has not yet been scrutinized, however the [ND.API](#) fixes the problem using **ND-xtreme** → **Links** → **DITA post processing**

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 → [\[ezRead#1.1\]](#)

For the *graphics post-processing* see also [Chapter 4.5.1 "PDF post processing"](#)

3 Important Topics

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

Topics	3.1 Using figures	38
	3.2 Basic figure aspects	39
	3.3 Working with the Glossary	40
	3.4 Working with equations	41

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 22: 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`
- `expanse`



Figure 23: 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 can just be refered by e.g. `gls/gls_<name>` whereas `<name>` is any term e.g `gls/gls_ezRead`. References are case-sensitive.



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

The ID's in the glossary are standardized

glossentry

`gle_<name>`

glossterm gls_<name>

glossdef glc_<name>

where <name> is the term you are referring to.



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 24: 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. More information about the DITA-OT can be read in the official user guide that accompanies the distribution. [otUsrGde#1]

Topics	4.1 Extensions on the Paragraph.	42
	4.2 Extensions on Section	43
	4.3 oxygen Annotations	44
	4.4 Extensions to Tables	45
	4.5 Extensions to fig/image	47
	4.6 Extension to Links	56
	4.7 Extensions to Notes	63
	4.8 Extensions to Lists	64
	4.9 Extensions to Mini-Toc	65
	4.10 Page Control	66
	4.11 System Variables	71
	4.12 Equations	72
	4.13 CHM extensions	75

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.xsl.



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.



*image
with text*

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.



*image
with
valign=top*

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 to 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

Empty section title


You can also create a section with an **empty title** or **no title**. If either of those is found, the output will be just like there was no section ever. So why would you do this ?

The answer is ... a section is a perfect idea to group several paragraphs of an e.g. "concept" in order to assign it a `product` or `audience` attribute.

`conbodydiv` would promise such grouping but it does not accept other than `section` or `example` anyway. So `section` is the best grouping idea, by using an empty title.

4.3 oxygen Annotations

oxygen supports  up that allows intermediate formatting and change history

- **Insertions**  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.

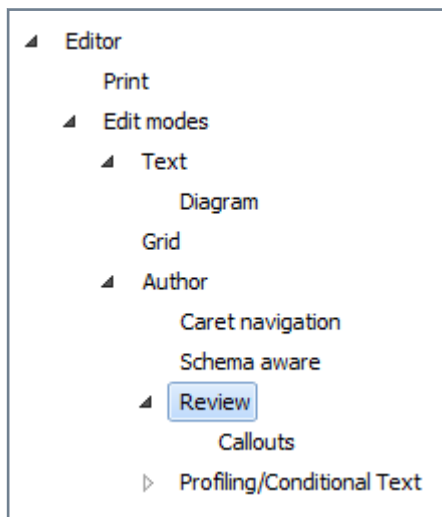




Figure 25: Preview references in oxygen

- 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  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`)
- **entry:outputclass:cellcolor=#1280FF | yellow | red ...** specifies background color for the single cell, it overrides a parent `row:outputclass:rowcolor=` definition for the cell
- **row:outputclass:rowcolor=#1280FF | yellow | red ...** specifies background color for the entire row
- **p:outputclass=compact** for table paragraphs
- **table:outputclass:rowcolor=#1280FF | yellow | red ...** specifies rowcolors 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 with left:0mm	→	Receive Data Compute random number R = RNG(1024)
	Get Result	←	Return result
	row:outputclass with rowcolor=yellow		Overriding with cellcolor=#FF8080
	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.xsl`:

```
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		

Table 3: Row colors

Sender		Receiver
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 (see 2.6)

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)

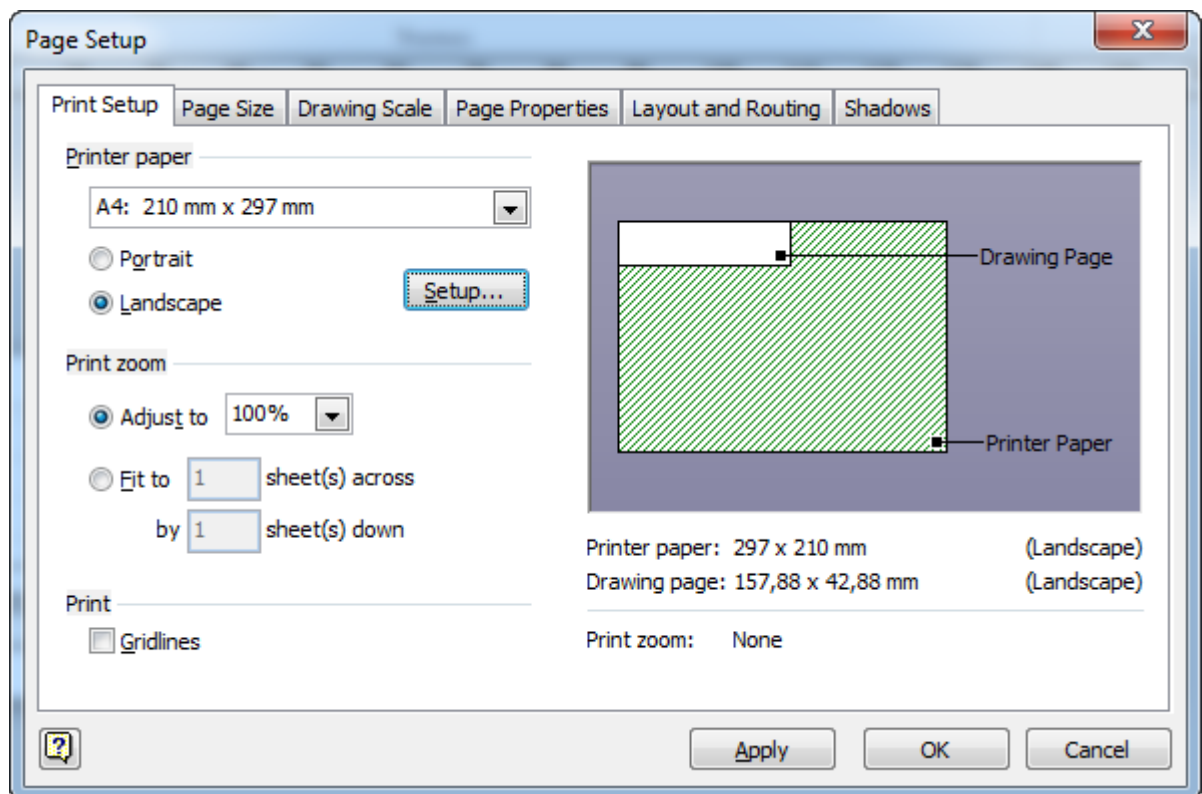


Figure 26: Selecting Margins

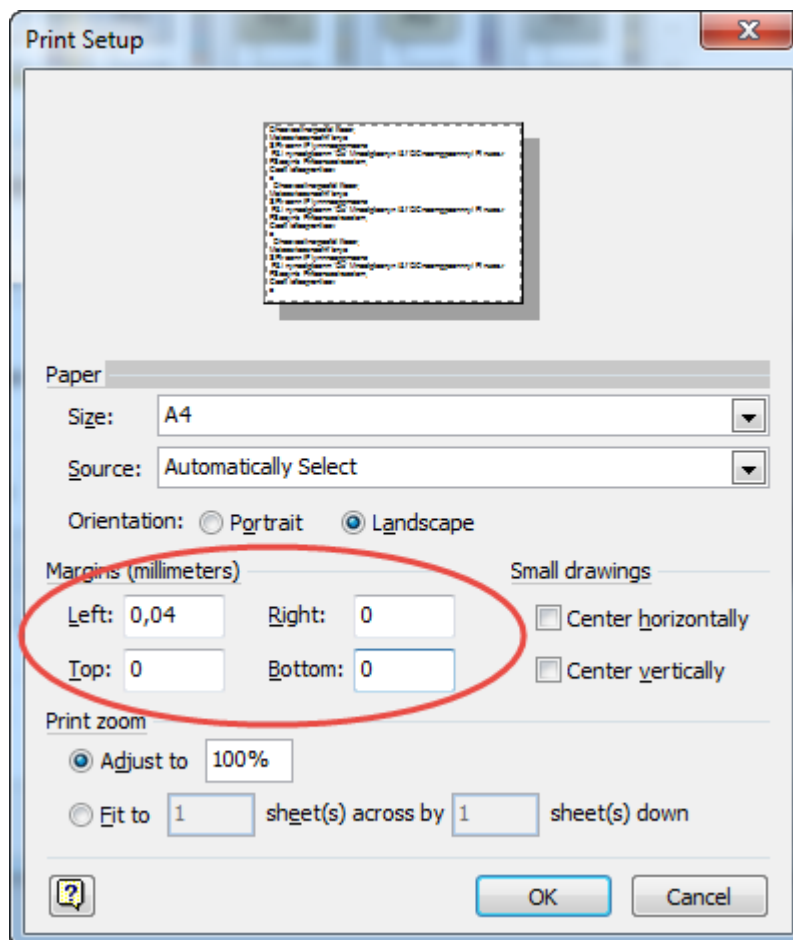


Figure 27: Setting margins

and

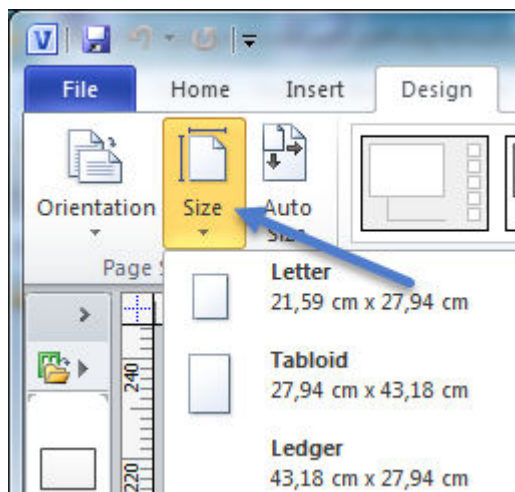


Figure 28: Set Page width

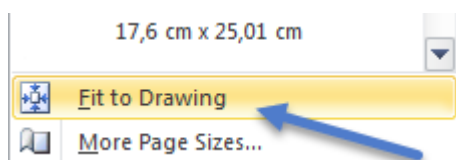


Figure 29: Set "Fit to Drawing"

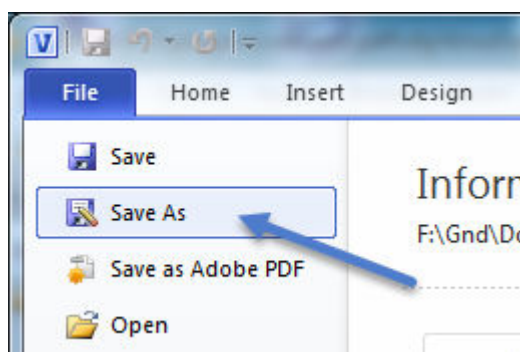


Figure 30: Set "Fit to Drawing"

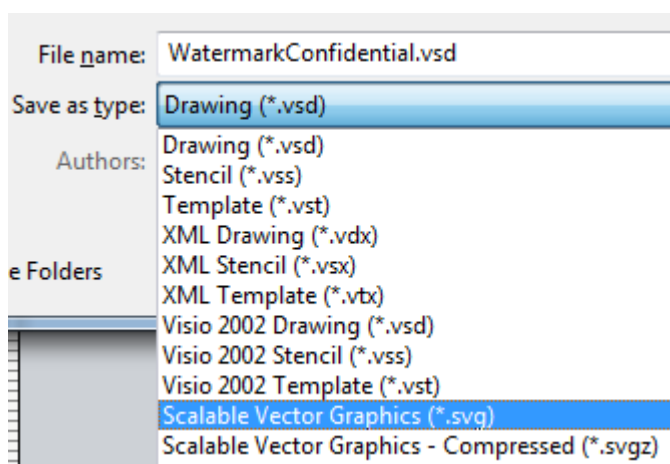


Figure 31: Set "Fit to Drawing"



Warning: Never **Save As** when you have objects selected. First use **ESC** to deselect 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.

PDF post processing

There are two issues which require SVG- or PDF post processing:

1. Two-or-more line text in boxes is not exported properly by VISIO, the link areas are way too small → this requires a post-processing step for the exported .svg file.
2. The VISIO links do not match the DITA-OT's PDF destinations (and it is nearly impossible to give'm the right links) → this requires a post-processing step

Process

to be continued

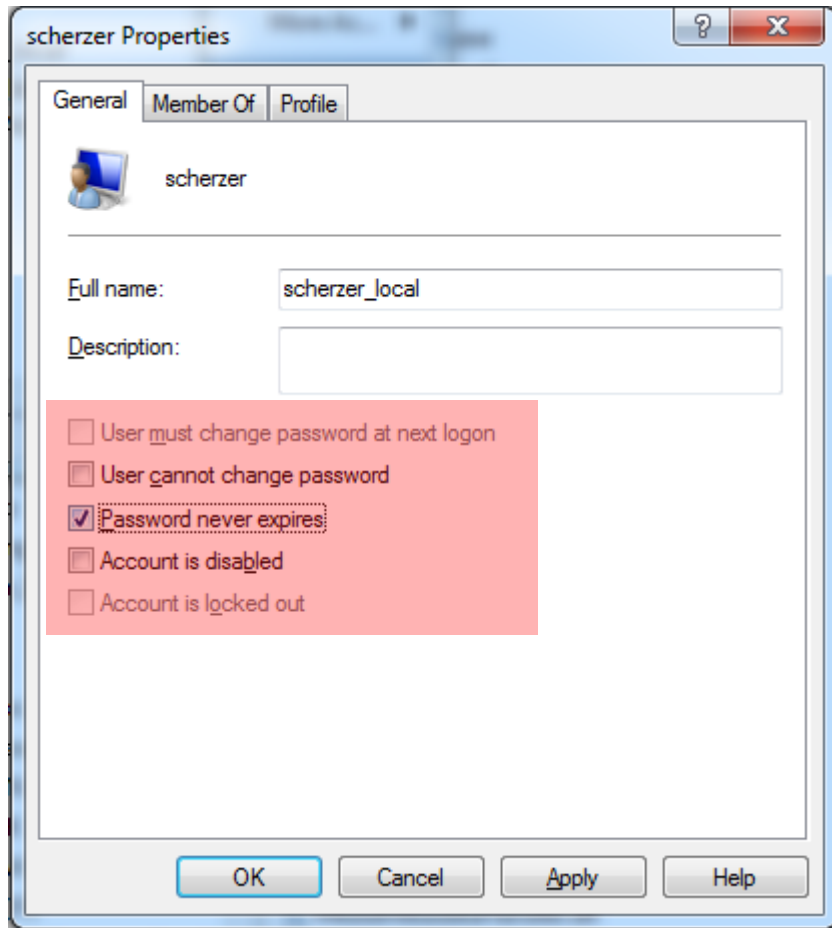
4.5.2 Links within graphics

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*.

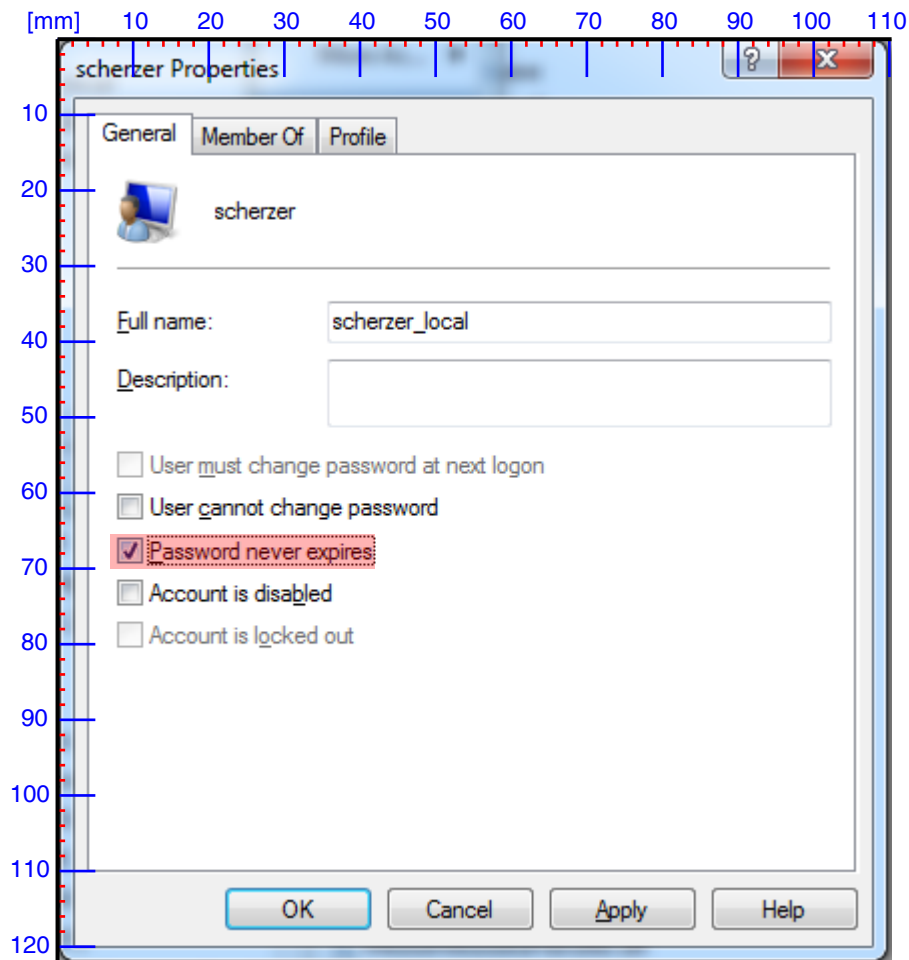


Attention: For a cross reference to a document beyond the scope of your main book (.ditamap) i.e. a file located in another than the .ditamap's directory or its children, you shall set the `xref/@scope=external`. Otherwise the DITA-OT will fail to process the final PDF because it gets lost in the location of directories.



Attention: For a cross reference to a document beyond the scope of your main book (.ditamap) i.e. a file located in another than the .ditamap's directory or its children, you shall set the `xref/@scope=external`. Otherwise the DITA-OT will fail to process the final PDF because it gets lost in the location of directories.

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>

```



Important: The scale and the area will **always** be computed from the **left margin**. Even if the image has `placement=break` and therefore allows and processes `align=center` the scale nor the areas will follow the alignment. Hence the area coordinates will not be relative to the picture but relative to the current left margin.

The advantage of this method is that the measurements through the scale are exact. Trying to follow the alignment would imply to know the side offset for an even or odd page. That, however, is complex and not worth to follow.



Note: Unfortunately on fully qualified (absolute) pathes 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 32: 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 33: Plain figure, image:width=50mm

The figure below expands to the entire page



Figure 34: 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 35: 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 36: image:outputclass=page, placement=break

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

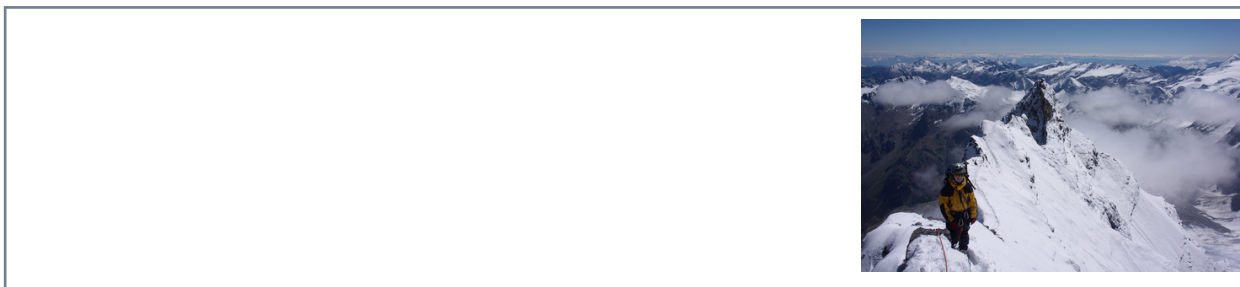


Figure 37: `image:outputclass=flow, align=right, width=50mm, placement=break`

Left alignment will take place in the text flow

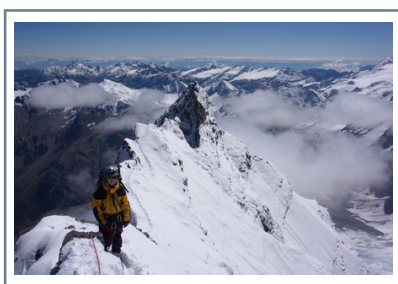


Figure 38: `image:width=50mm, align=left, placement=break`

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

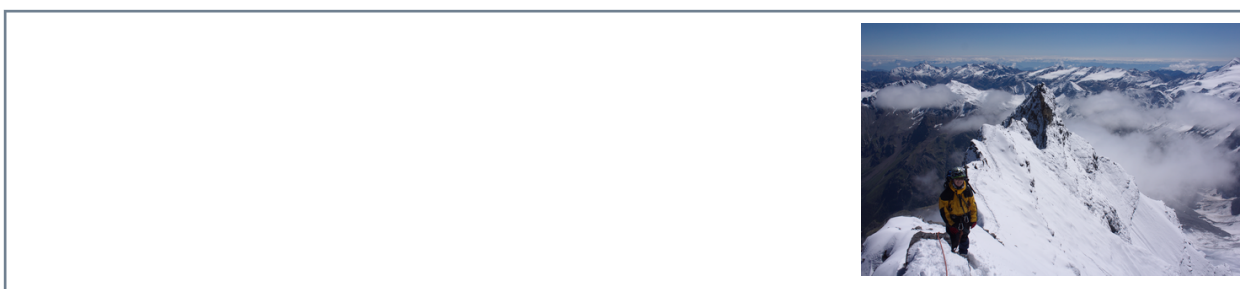


Figure 39: `image:align=right, width=50mm, placement=break`

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.

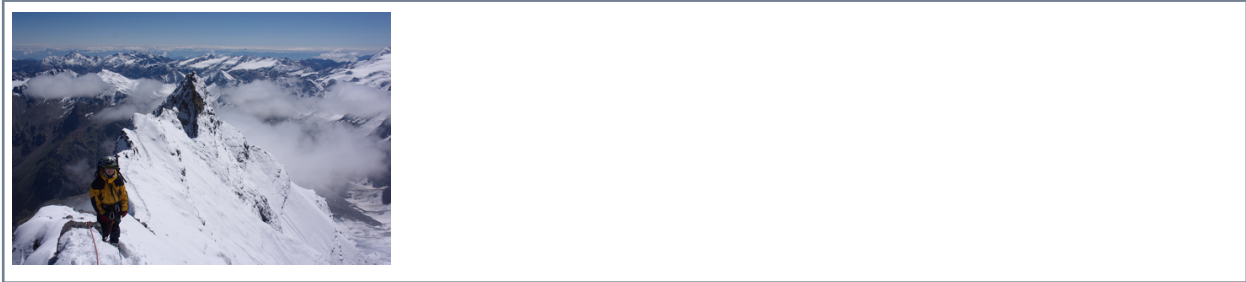


Figure 40: image:align=right, width=50mm, but placement=inline (inline does not obey 'align')

Links in figures

A specific formatter can be set.
Here is an image with links [ISO4#5.3] to different targets

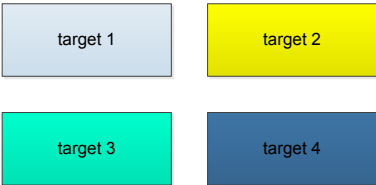


Figure 41: Image with #hstarget1..4

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
- title

- 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")

title

prints the target's title

label

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



Note: This is also the default in the latest DITA-OT, but we keep it as it allows mixed use with other tags

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] <text> [label] <text> [title] <text> [onpage|pagenum|pagenumonly]
               <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 [num] and [chp] together

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 26](#)

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 <code><label> <number></code> text the <code>[label]</code> macro can still be used, if more than the label shall be displayed. The combination <code>outputclass=[label] <text> [title]</code> 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 <code>outputclass</code> 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 [Figure 26](#)

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

<code>outputclass = Find more in [num]</code>	test Find more in 4.6
<code>outputclass = Find more in [chp] "[title]"</code>	test Find more in Chapter 4.6 "Extension to Links"
<code>outputclass = Find more in [chp] "[title]" [onpage] ff.</code>	test Find more in Chapter 4.6 "Extension to Links" on page 56 ff.
<code>outputclass = [see] in [num]</code>	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 56 of many)
outputclass = [num]	test 4.6
outputclass = [chp] "[title]"	test Chapter 4.6 "Extension to Links"
Feel free to play with other combinations	

xrefs with linktext

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 7 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
here is test	
this is a paragraph next para with compact	
next para next para with compact	

- Table in List

Table 6: Table in list

Header 1	Header 2
here is test	
this is a paragraph next para with compact	
next para next para with compact	

Table 7: Table in normal text flow

Header 1	Header 2
here is test	
this is a paragraph next para with compact	
next para next para with compact	

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.

Extensions to the auto-link feature

Linktext

The construct

```
prior text [any text followed by two colons::ezRead#9.3.3] ... more text
```

allows to use the text before the double-colon (::) to be used as `linktext`. By default the resulting `linktext` will not be put in brackets.

Table 8: Example for the linktext construction

```
prior text any text followed by two colons ... more text
```

Linktext in brackets

If the Linktext-Feature is used, then a **preceding exclamation mark**

```
prior text ![any text followed by two colons::ezRead#9.3.3] ... more text
```

will indicate to put the `linktext` construct into brackets. This is often useful when reference to literature are made with their original biblio tag, whereas you actually want to link to a biblio tag that follows the naming conventions of [ezRead#9.3.3].

Table 9: Example for the ![...] construction

```
prior text [any text followed by two colons] ... more text
```



Tip: You might ask yourself, how it was possible to write ![...] since it should be caught by the translation and as such it cannot appear here. The trick is to actually write this as

```
!<ph> [...] </ph>
```

The use of the `ph` tag prevents the processor from recognizing the ![...] as one text and hence the match in the DITA-OT plugin extension will not catch the construct. Since the `ph` does not create any formatting, it is a perfect element to be used for such purposes (also to give even a word an id for referencing).

Avoid link

Finally you might have some text to write of which you know that you do not want an automatic linkage from the [...] format. There you can use the construction

```
~ [...]
```

which indicates to the tool that you don't like the interpretation of the brackets as link. In contrast to the [Ignore Lists](#) which care about the bibliography, this avoids the automatic linking function and prints the text right as you type it after the ~.



Note: The construct

```
~[linktext]
```

seems to be useless because with the ~ you indicated "do not link" and with the `:ezRead#9.3.3` term you specify a link. This may, however, be of relevance if you are using the feature of [Generating a Local Bibliography](#).

The associated stylesheet takes the `bibliotag` - not the `linktext` - (here "ezRead") to identify the document to be taken to the generated bibliography. So if you want to "show" the `linktext` but nevertheless get this non-referencing token to be automatically added to you bibliography, then you need this construct.

4.6.4 ezRead Main-Book flag

The ezRead technology will export a tree of associated PDF documents into two directories

- Books
- References

whereas `Books` holds the few mainbooks and `references` may contain many (even 100+) documents that are referenced by the main books. The separation is obviously done to maintain focus on the important documents.

However - how can the ezRead plugin know that a book is supposed to be an important document.?

Therefore the `.ditamap/bookmap/bookmeta/shortdesc/keyword` may indicate through the verb `mainbook` that this book belongs to the main important books and shall be exported to the `books` directory.

```
<bookmeta>
  <shortdesc>
    <keyword>mainbook</keyword>
    <data name="style">dsgn_smart</data>This is the short title</
shortdesc>
  <authorinformation>
    <personinfo>
```

contains the keyword `mainbook`.

During the export function [[ezRead#12.1.26](#)] every document in the link tree is visited and tested for the keyword `mainbook` in its metadata. If found, the book will be stored in the BOOKS directory regardless from which target directory it was supposed to be stored by its position in the link hierarchy.

4.6.5 Manual Named Destinations

Any id-attribute given in DITA that starts with an underscore `_` will be converted into an *M8.newlink*.`<name>` named destination. The underscore will not be part of the final named destination, it is used to make the ND.API plugin recognize that this destination shall be converted → [[ezRead#ditapost](#)].

To create an `ezRead` named destination, use an **underscore** as first letter of any id-attribute

```
<concept id="_manDest">
  <title id="_manDest">Manual Named Destinations</title>
  <shortdesc></shortdesc>
  <conbody>
```

will convert the title's id "_manDest" into an [ezRead](#) (FrameMaker compatible) named destination "M8.newlink.manDest" destination which can later be addressed as `[MyDita#manDest]`.

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

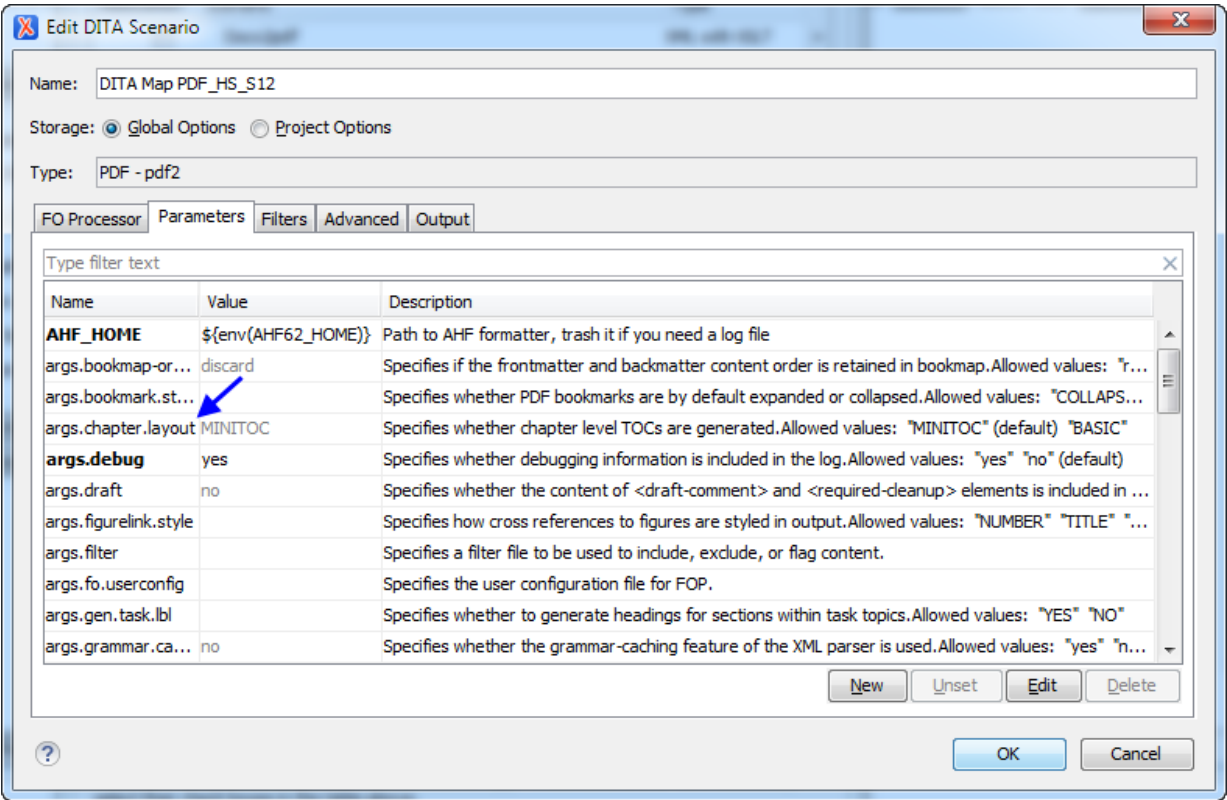


Figure 42: Configuring mini-TOC in runtime parameters

4.10 Page Control

Four important extensions were made to the page layout.

4.10.1 Landscape page format for sub-chapters

Landscape layout

Landscape layout has suggested in [DtPrt#7.12.8] available in the Dita-OT for `Head1` chapters only by assigning

```
concept:outputclass = landscape
```

The same chapter indicates that landscape-layout on any subchapter is difficult and require intensive understanding and modification of the DITA-architecture.

[DtPrt#7.12.8] states the following difficulties on the landscape assignment for other than `Head1` level topics:

Once you create a landscape page master, you can assign it to topics you want displayed in landscape mode. In word-processing applications, you can simply scroll through your document and **assign the page wherever it's needed** (although this approach can produce some unexpected results if your document becomes longer or shorter and pages reflow).

It's **almost impossible** to do the same in XSL-FO because, of course, there is no document to scroll through—page master assignment is automatic. But there are a few methods you can use to assign specific page masters during the build. These methods range from the slightly complicated to the mindblowingly complicated.

As you might guess, the more flexibility you need, the more complicated the approach. Within the current PDF2 architecture, there are **serious limitations** to how page sequences are generated. Without changing this architecture it is **essentially impossible** to do anything other than change the page sequence **at top-level-topic boundaries**. Changing that architecture would be one of those mindblowingly complicated solutions just mentioned and is beyond the scope of this book.

Nevertheless this problem could be solved with medium effort. The DITA architecture didn't need to be changed deeply, however, several rearrangements for the `page-sequence` were done. Changes were done in `commons.xml` and around the code which can be found by searching *group-adjacent*.

To get any subchapter as landscape, you may just use the same notation on subchapters

```
concept:outputclass = landscape
```

whereas `concept` is a representative for any main topic type (`task`, `reference`, `topic`).

Landscape within a chapter

Some authors might want to go further and have only a particular part of a chapter to be shown in landscape format. Although this is not recommended as good writing style - situations may occur that required such mechanism. This can be achieved with a trick.

Example Let us assume, an author wants to have the mid part of a chapter to be landscape. Then s(he) shall do the following:

1. If the present (concept) chapter is X, create two other adjacent chapter Y, Z.
2. Put the text before the landscape into concept chapter X
3. Put the text for landscape into new concept chapter Y - Leave the title empty
4. Put the text after the landscape new into chapter Z - Leave the title empty

The plugin supports empty titles as described in [Chapter 4.10.5 Extension on the Title Element](#). This will therefore virtually generate one chapter X because the titles of Y and Z are empty and their content is added adjacent to X.



Note: The empty title feature is build such that neither the TOC, nor the bookmarks or the mini-TOC will recognize the empty chapters, this makes the empty titled chapter like a simple body extension of the previous chapter.

4.10.2 Page Heading Control

The current plugin prints the current chapter's title on every page's heading. However, you might not want to go this as deep as e.g. to *Level-6* chapters, right?

The system variable

```
<xsl:variable name="topicHeader.numLevel">3</xsl:variable> <!-- values
from 1..n -->
```

in the plugin's file `basic_attr.xsl` allows you to control the maximum heading level until which the heading shall follow the current chapter. Hence if we are in an *Level-6* chapter, for the above setting `topicHeader.numLevel = 3` the heading in the *Level-6* chapter pages will be the latest *Level-3* ancestor's title.



Important: If you are using `PART` notation where the `Head1` chapters are already children of a `PART` (I don't like that too much, yet it is common use) then the `PART` does count as a level. This implies a small misunderstanding. Any *Level-2* chapter as a child of a major `PART` will actually be recognized as a *Level-3* chapter.

This effect requires you to set the desired heading value **+1** the level you like to consider as maximum heading level. As the `PART` counts as a level, you pay it with the **+1**.

4.10.3 Newpage Enforcement

new page

To create a new page for the present topic's `topicname` shall get the `outputclass=newpage`

```
task:outputclass = newpage
```

The newpage feature is available for

- topic
- concept
- reference
- task
- fig
- table



Important: The landscape `subchapter` feature was originally only available for subtopics whose parent is a chapter (`bookmap/chapter`) i.e. those subtopics that are in the body part of a `ditamap`, which are all `topicrefs` that follow `frontmatter`. After the changes in [Landscape page format for sub-chapters](#) the landscape features is available to any chapter depth.

In the implementation `outputclass=landscape` is not relevant for subtopics under `preface`, `notices` `abstract`, `glossary` which typically are `Head1` - where it works, or do not contain subchapters (although this is supported, but not as landscape option)

4.10.4 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 → [\[xslfo#keep-with-previous\]](#).

Strength

The strength of the keep condition is set to '1'. It shall not be `always` because that would yield too many brute force results.

4.10.5 Extension on the Title Element

The title element has two extensions

- empty titles
- plain named destinations


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 target) chapter into invisible sub-chapters. That may also be attractive for the purpose of reusing text blocks on file level.

**Important:** There is a special use for an empty title if a text block shall be printed in `landscape` format within a chapter. Find more in [Chapter 4.10.1](#)

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

Plain named destinations

If you give the title an `@id` attribute with a preceding underscore (e.g. `_mydest`), then the `ezRead` post-processor (as an Acrobat Plugin) [\[ezRead#12.1.20.1\]](#) will remove the DITA-OT generated prefix `unique_42_...` and create a plain named destination from the id - removing the underscore `_`.

Find more in [Chapter 4.6.5](#)

4.11 System Variables

System Variables are expressions (e.g. `Dita-OT`) that you want to reuse in your text, but for some good reason you do not want to write them explicitly but pull from some global location.

This case can occur, e.g. for

- special names
- version numbers
- product abbreviations
- special words in different languages
- and more ...

The correct method to pull text from a central file is the DITA `conref` or `conkeyref` attributes which "buy" the content of the system variable for the present position.

So it seems practical if you write such word once in your document and all other places refer to it using `conref` or `conkeyref`.

There is a major disadvantage ... Rearranging chapters your "sources" will be distributed and might not be part of your next book. Then you need to find another "source" and having many system variables you can easily mess up with rather frustrating results.

The remedy would be ONE file only (e.g. `sysvars.dita`), that keeps all system variables. However, what, if you do not want to print that chapter with the system variables in the final document ? If you try to use the `ditaval` mechanism with some `audience=noprint` and give your `sysvars.dita` chapter the tag `audience=noprint` then you can expect a bad surprise.

The DITA-OT removes all non-printed chapters in an early pass even before evaluating everything else. If your `sysvars.dita` is excluded through the `ditaval` mechanism, your `conrefs` into the `sysvars.dita` will not be resolved as the target already disappeared before.

Solution

The solution comes with an `outputclass = noprint` attribute in your `sysvars.dita` and it is not going to be filtered using the `ditaval` mechanism.

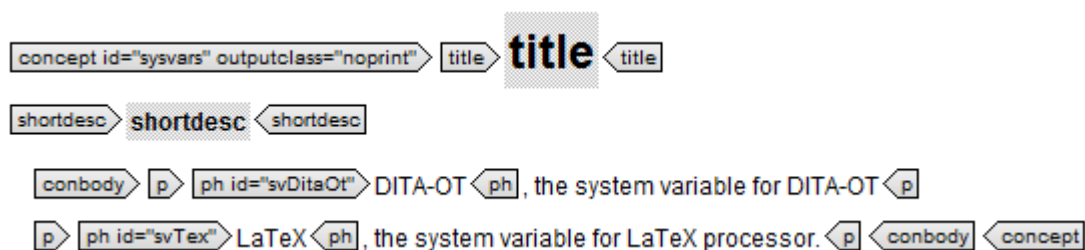


Figure 43: Example for a correct `sysvars.dita`

where the coding is

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA Concept//EN" "concept.dtd">
<concept id="sysvars" outputclass="noprint">
  <title/>
  <shortdesc/>
  <conbody>
    <p><ph id="svDitaOt">DITA-OT</ph>, the system variable for DITA-OT</p>
    <p><ph id="svTex">LaTeX</ph>, the system variable for LaTeX

```

```
processor.</p>
  </conbody>
</concept>
```

The above `sysvars.dita` is a simple `concept` which lists all system variables. The `<ph>` ("phrase" element) is very practical here because it does not have any formatting and therefore is quite versatile as a wrapper for content to be reused.



Important: Of course you can use any other element e.g. `xref` which is also practical to organize links.

Reusing

The actual reuse is obvious ... the client will use its `conref` or `conkeyref` attribute to "buy" the content from the system variable.

The `outputclass=noprint` avoids that anything from the chapter is being printed.



Warning: It is important to keep the `title` **empty**. If you do not follow this rule, the DITA-OT will consider the `sysvars.dita` for chapter numbering and you will find your `sysvars.dita` in the Table Of Contents and the bookmarks although the chapter is not printed. For the empty title feature see [4.10.5 Extension on the Title Element](#)

4.12 Equations

Equations have been added since DITA 1.3, the DITA-OT version 2.2.2 does not yet support equations. However the present plugin already supports equations in the way they are specified in DITA 1.3.

The main challenge for equations is their creation. Certainly the most popular way of creating equations from simple script language is the [LaTeX](#) approach. An example for a [LaTeX](#) equation is

Example (to be done)

However [LaTeX](#) language cannot be taken into XML.

An equation can also be written in an XML representation whereas the most appropriate language would be [MathML](#) which is specified by the [w3-consortium](#). An example for [MathML](#) shows its disadvantage:

```
<m:math xmlns:m="http://www.w3.org/1998/Math/MathML">
  <m:semantics>
    <m:mstyle mathsize="12.0pt">
      <m:mi mathvariant="italic">X</m:mi>
      <m:mo>&#x3d;</m:mo>
      <m:msup>
        <m:mi mathvariant="italic">M</m:mi>
        <m:mi mathsize="50.0%" mathvariant="italic">e</m:mi>
      </m:msup>
      <m:mi>mod</m:mi>
      <m:mi mathvariant="italic">N</m:mi>
    </m:mstyle>
  </m:semantics>
</m:math>
```

The code is XML, but nearly impossible to understand, while [LaTeX](#) actually allows to write equations right from the text mode.

To ease the creation of equations, several powerful tools are on the market, my personal recommendation goes to the [MagicMath](#) product because it has quite a rich set of features and import/export functions, very suitable for the purpose.

Rendering? Another point is the question, whether your formatter (Apache FOP, XEP, Antenna House) renderer is able to accept [#unique_7/unique_7_Connect_42_gls_mathml](#) code in order to create the proper equations with all its fonts and micro-adjustments. As of 21th March 2016, only the Antenna House Formatter is capable to render embedded MathML to equations in PDF.

Which points to the question, why to use MathML representation for equations if the renderer (except Antenna House) does not accept it anyway?

Recommendations Given you can afford the investment into an equation editor tool e.g. [MagicMath](#) you will be able to export your result into [SVG](#) format. You can then insert the [SVG](#) drawing as image into your equation-block like

```
<equation-block>
  <image placement="break" href="../../../Book/src/eqx/rsa01.svg"
  id="image_dz1_zdr_jv"/>
</equation-block>
```

which delivers

$$X = M^e \bmod N$$

This method works in all renderers and you have the full comfort of proportional fonts [SVG](#). Another advantage is, that you can well see your result and your XML code is not spoiled by unreadable MathML code. And of course you can keep your equations in a separate directory all together.

The disadvantage of using [SVG](#) is related to the proper size. Typically your equation editor tool does not allow a scaling of its [SVG](#)-result so you might experience unwanted size. Of course you can use the `scale` attribute in the `image` which will fix the problem. But if you ever decide that all your equations should be a bit smaller, you need to fix all your `scale` specifications in all your equations.

Therefore a system variable has been created which does the scaling for you generally as part of the DITA-OT plugin.

4.12.1 Using embedded MathML

MathML can be used several ways, Antenna House Formatter does even allow embedded MathML as follows

$$X = M^e \bmod N \tag{Eq. 1}$$

with

$$N = p \cdot q \tag{Eq. 2}$$

and

$$\begin{aligned} p &\rightarrow \text{prime}(\text{Rand}) \\ q &\rightarrow \text{prime}(\text{Rand}) \\ \text{Rand} &\rightarrow \text{random}(1..2^{1024}-1) \end{aligned} \tag{Eq. 3}$$

The code has to be setup as follows:

```
<equation-block>
  <mathml>
    <m:math>
      ... further statements coded in mathML using m: prefix
    </m:math>
  </mathml>
  <equation-number/>
</equation-block>
```

4.12.2 MathML within XSL-FO

MathML is treated as a vector image. XSL-FO V1.0 specification only permits embedding foreign object via `fo:instream-foreign-object` and `fo:external-graphic`.

When using a DTD in the `fo:instream-foreign-object`, it is necessary to treat its entirety as a CDATA section.

Since the DTD is not required when a special mathematical symbol is described by Unicode, the MathML markup can be placed into `fo:instream-foreign-object`.

When using `fo:external-graphic` to place the equation, the external MathML file should be described as the `src` attribute.

Inline and Block

The expression marked up by MathML can also be used in a line like an ordinary character. This is called an inline object.

Also, a paragraph of only a math expression can be made by surrounding it with `fo:block`. This is called a block object.

Interface

When formatting math expressions marked up using MathML, the values specified in the XSL-FO body text such as the size of a character, the font family, etc. are inherited by the math expression.

Thus if the characters of the body text increase in size, the characters of the MathML also increases in size. This keeps the balance of the text and expression styles.

On the other hand, the information as to the baseline of math expressions is passed from the renderer of the MathML to the XSL-FO engine. This arranges the baseline of the math expression and the text.

Related Information

<https://www.antennahouse.com/antenna1/mathml-option/>

4.13 CHM extensions

The CHM processing is explained in .

An important change was made to support Auto-Extension conversion. Any xref-reference to a <filename>.SVG file will be converted into a xref-reference to a <filename>.PNG. This is necessary because we highly recommend to use .SVG wherever it is possible, but CHM does not support SVG files.

Hence for every available .SVG file, for CHM output, the .SVG shall be exported to a .PNG file with a corresponding image conversion program.

The use is explained in

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	76
	5.2 Company Logo	76
	5.3 Security Class	78
	5.4 Watermark	78
	5.5 Front Picture	79
	5.6 Second Page layout	80

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 44: Front page definition with logos



Important: The path to the logo-file has to be counted relative to the position of the current .ditamap

The `data:height` attribute is optional and allows to change the default height (10mm) of the Logo in case it appears optically smaller/larger than the specified 10mm as result of an optical effect that may depend on the brain's perception of the logo.

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

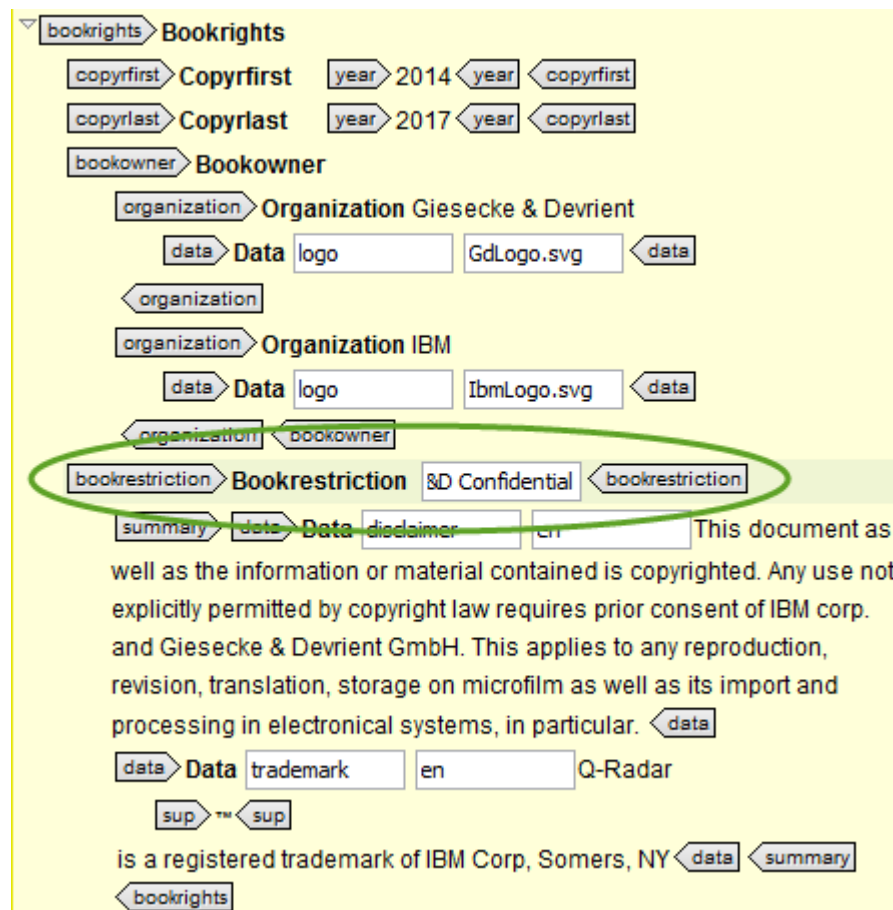


Figure 45: Security Class definition in Author mode

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

```
<proinfo>
  <prodname>Version
  <data name="image" value="front.png"/>
  <data name="top-left-width" value="100mm 80mm 80mm"/>
```

```
</prodname>
<vrmlist>
  <vrml version="0.2" release="20150928" modification="wmConf.svg"/>
</vrmlist>
</prodinfo>
```

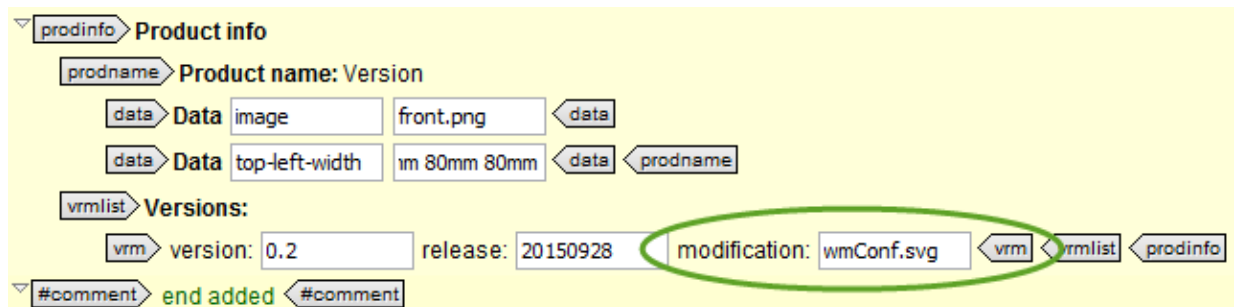


Figure 46: Watermark specification in the front page

Watermark on
front page
only

It might be required to have a watermark on the front page only. In this case, simply use the front page picture feature → [Chapter 5.5](#)

The watermark drawing

Location

As the watermark pictures are more of a general nature, they are expected to be located under %Dita-OT%\plugins\com.refl.pdf\cfg\common\artwork. Therefore you shall not enter a path definition in the file name.

More watermarks

Create your own watermarks under %Dita-OT%\plugins\com.ref1.pdf\cfg\common\artwork.
Currently the supported watermarks are

- **Draft:** wmDraft.svg
- **Comments:** wmComments.svg
- **Confidential:** wmConf.svg
- **Restricted:** wmRestricted.svg
- **Secret:** wmSecret.svg

To add your own watermark, edit an existing VISIO file (.VSD) in the artwork folder and **Save As** corresponding .SVG. If you don't have Microsoft-VISIO the you may directly create .SVG file using the Inkscape open source SVG editor..



Important: The watermark drawing shall be transparent to maintain readability. This can be achieved with appropriate 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 vrml="0.2" release="20150928" modification="wmConf.svg"/>
</vrmlist>
</prodinfo>
```

The oxygen author mode shows the front picture definition as follows

prodinfo Product info 2015-01-22

prodname Product name: Version

data Data image gfx/front.png data

data Data top-left-width n 60mm 100mm data prodname

vrmlist Versions:

vrml version: 1.0 release: 100.25003.230 modification:

bookid Bookid

Figure 47: Front picture definition in oxygen author mode

!

Important: The path to the front-picture file has to be counted relative to the position of the current `.ditamap`

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	81
	6.2 Creating a Glossary	86

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 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 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'"/>
```

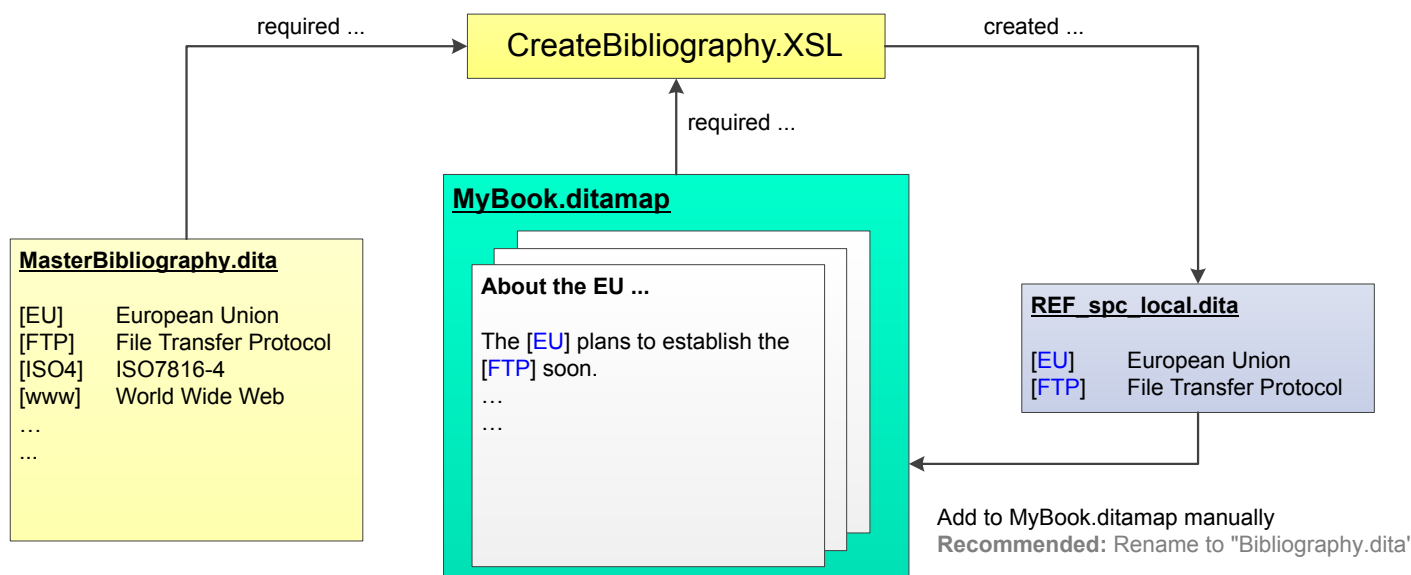


Figure 48: Creating a local bibliography

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**

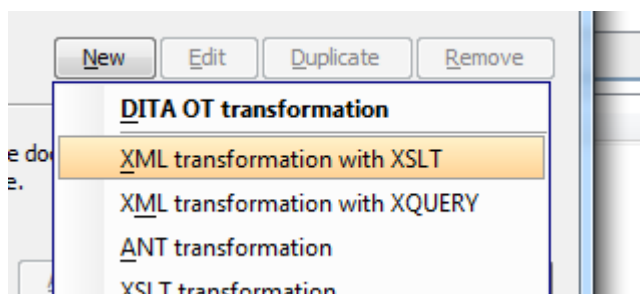


Figure 49: New scenario

Be sure to use the right Transformer type

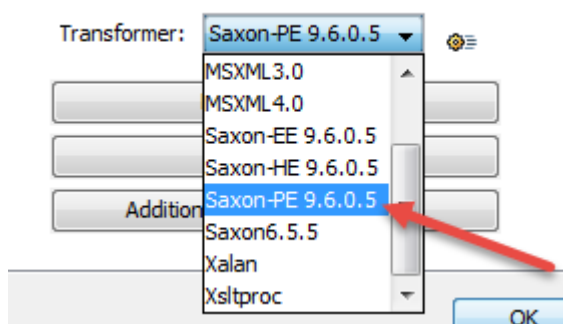


Figure 50: Select scenario type

Create the scenario according to Figure 51

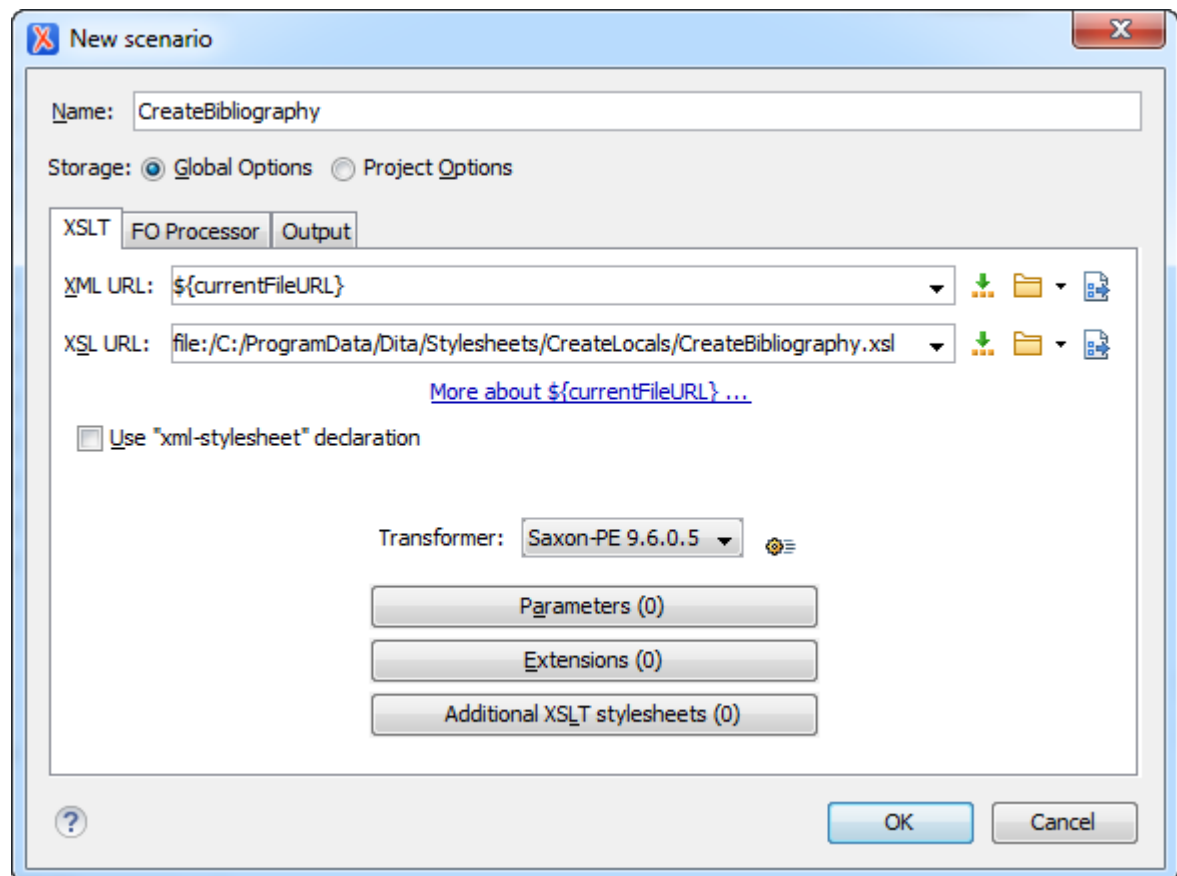


Figure 51: Create Scenario



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.xsl` 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	87
	7.2 Running the docx2dita conversion	91

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

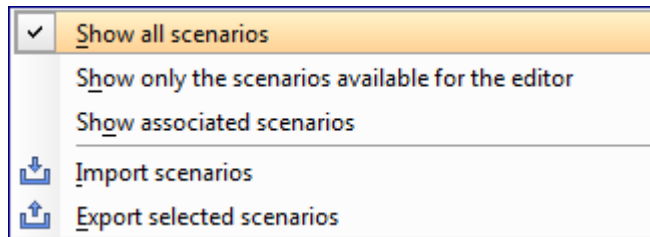


Figure 52: All scenarios view

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

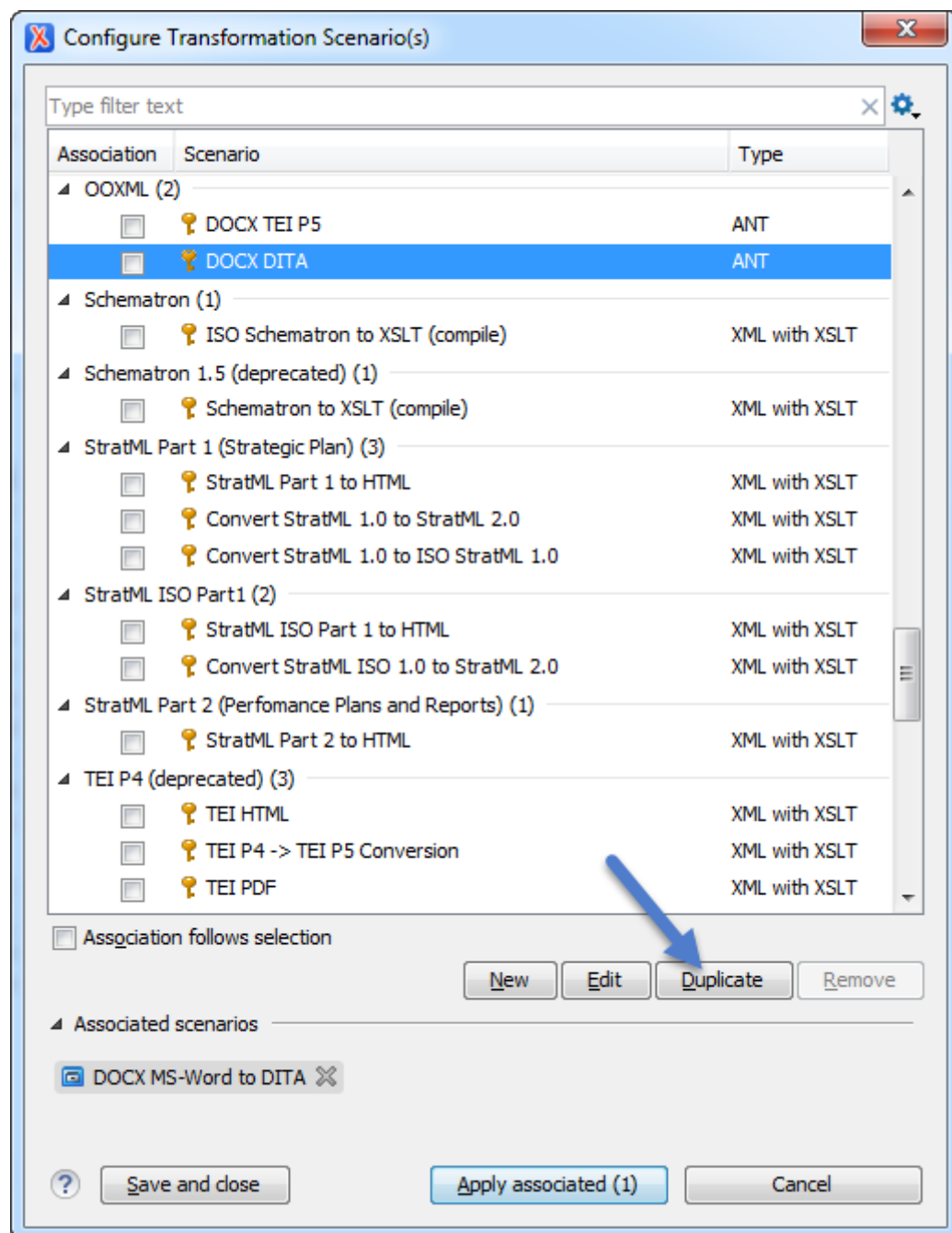


Figure 53: Open oxygen scenarios

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**

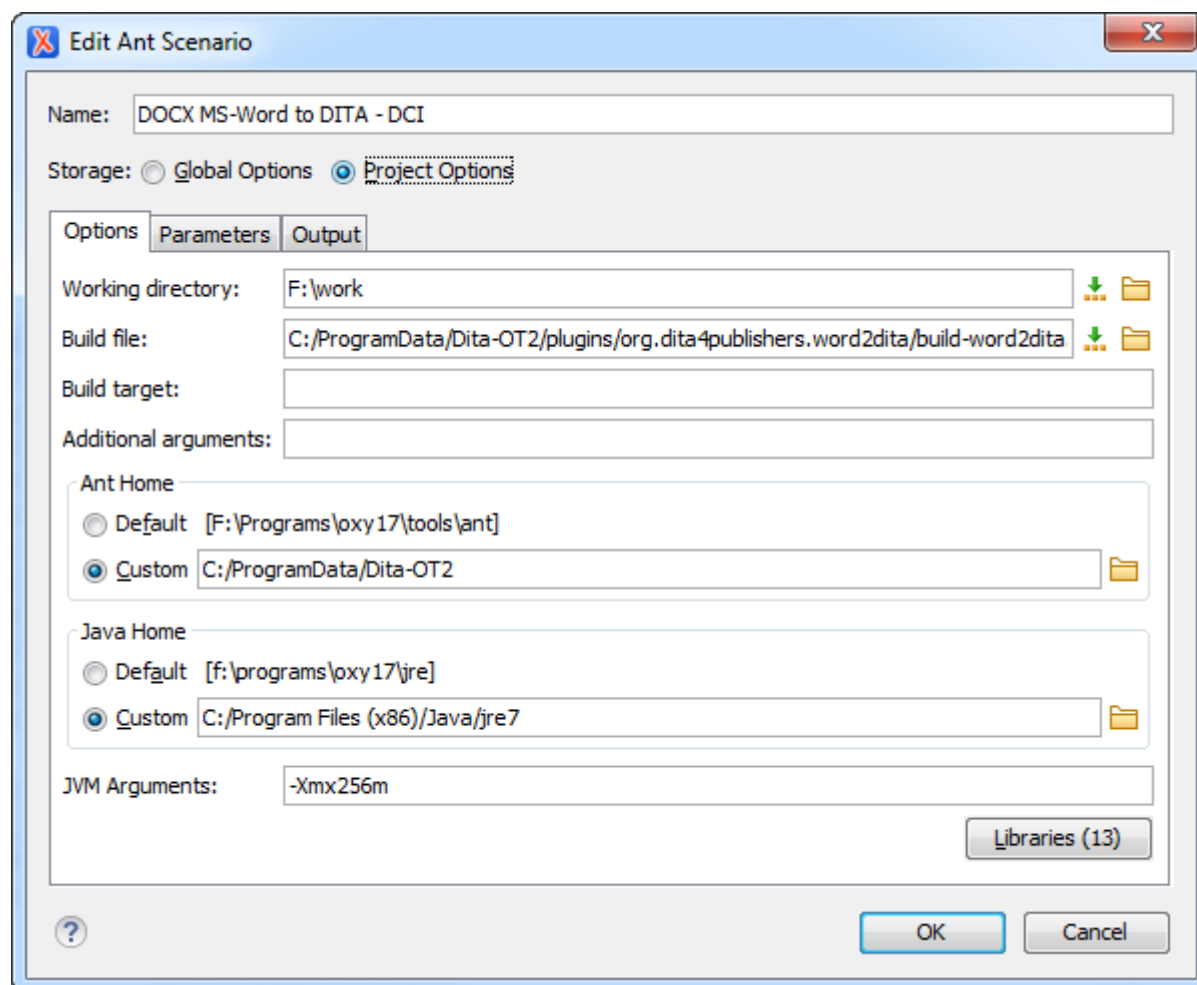


Figure 54: Change the scenario options

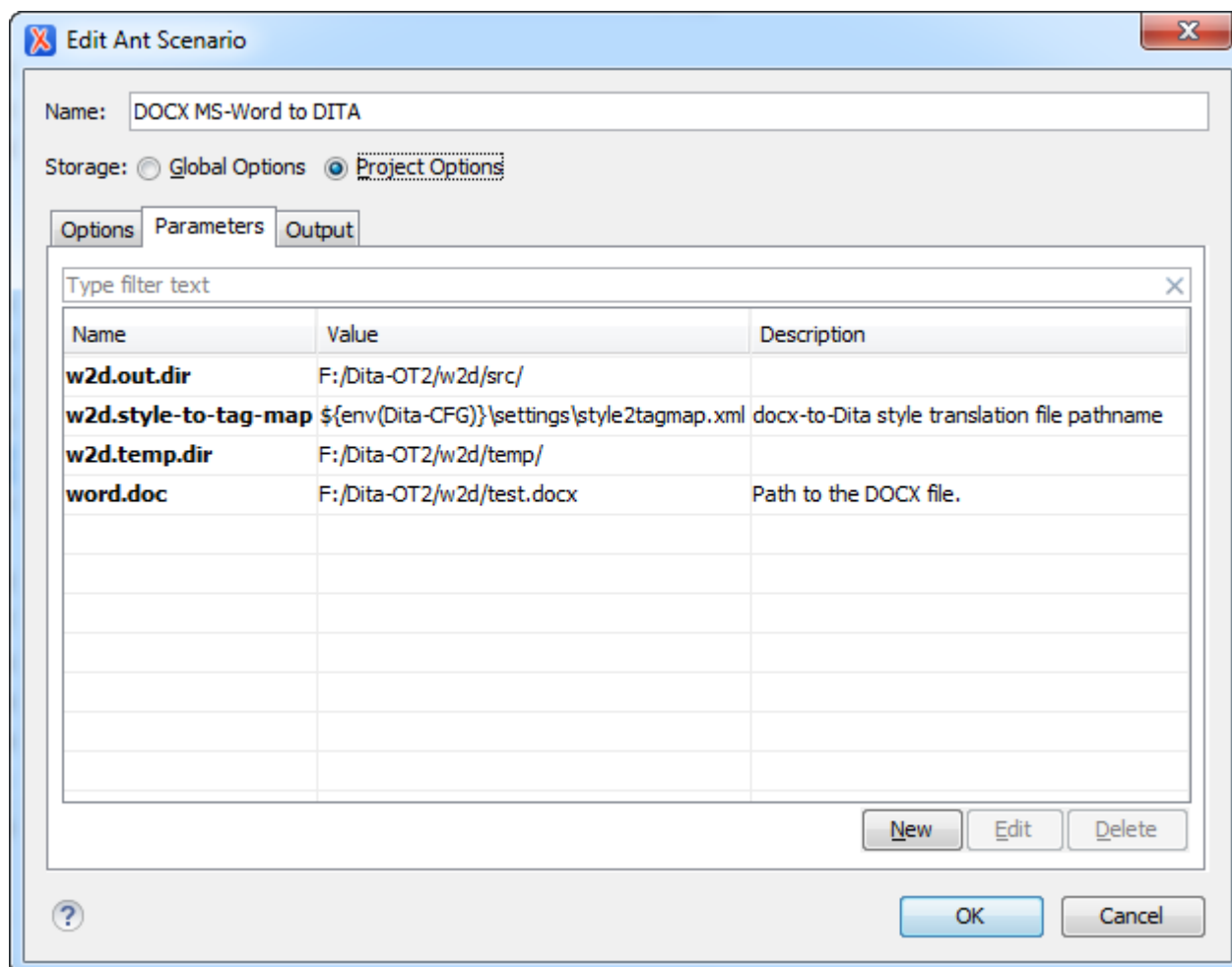
4. Change the **Parameters** as follows

Figure 55: 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 pathes or even the oxygen variables.

5. Change the Output accordingly

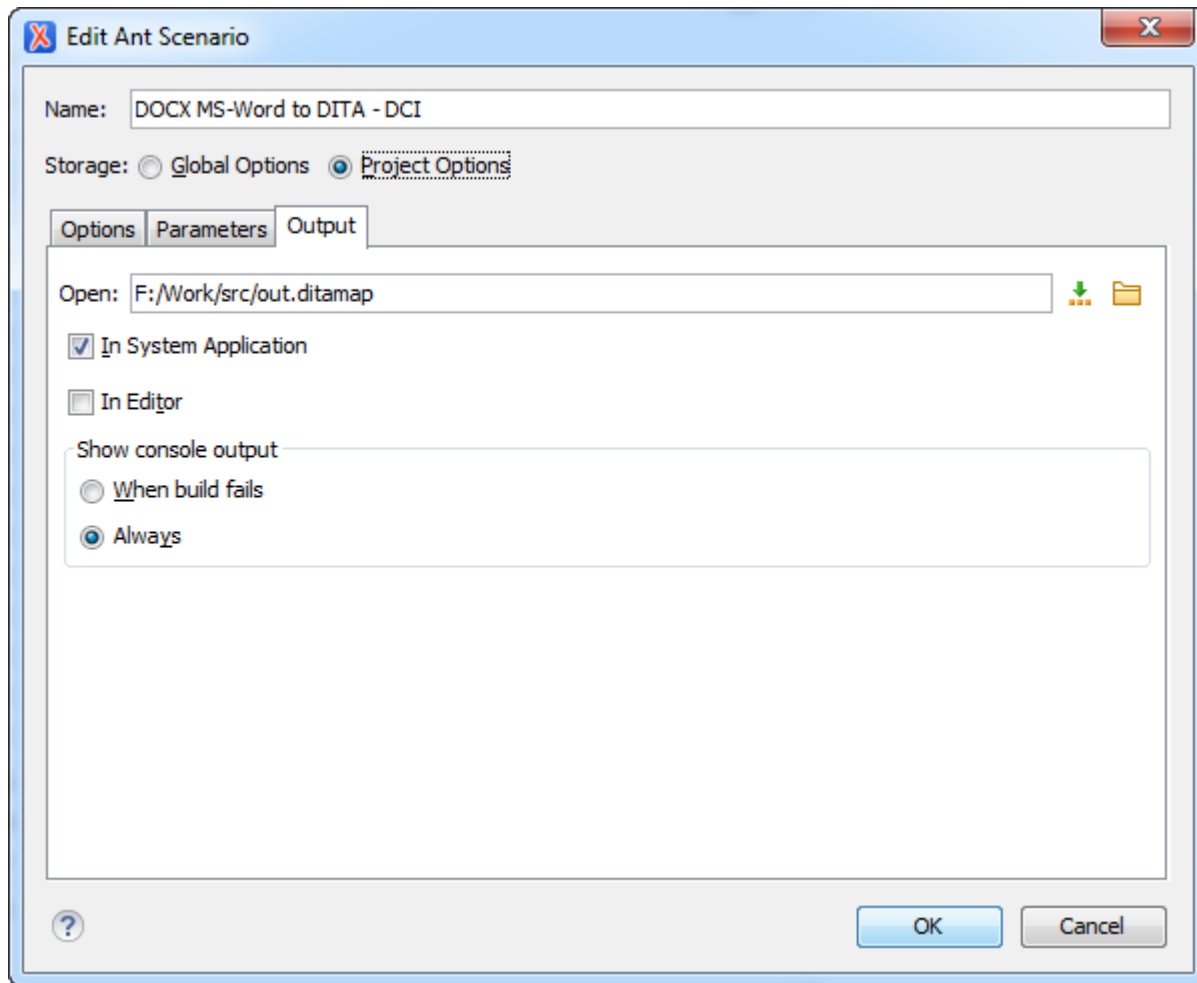


Figure 56: Edit output parameters

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

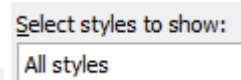
The DOCX2DITA conversion uses the first paragraph with `style=Title` in order to start the conversion. Hence you need to

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

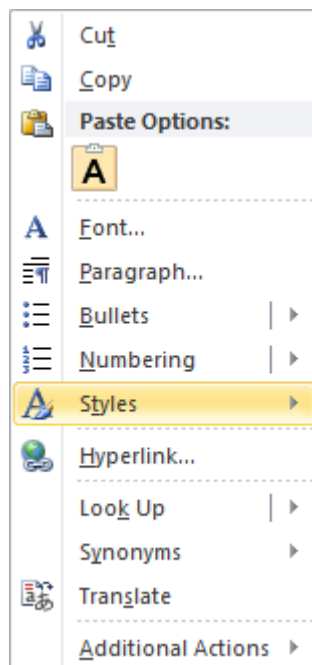


Figure 57: Select the 'styles' context menu

Select **Styles** and

then

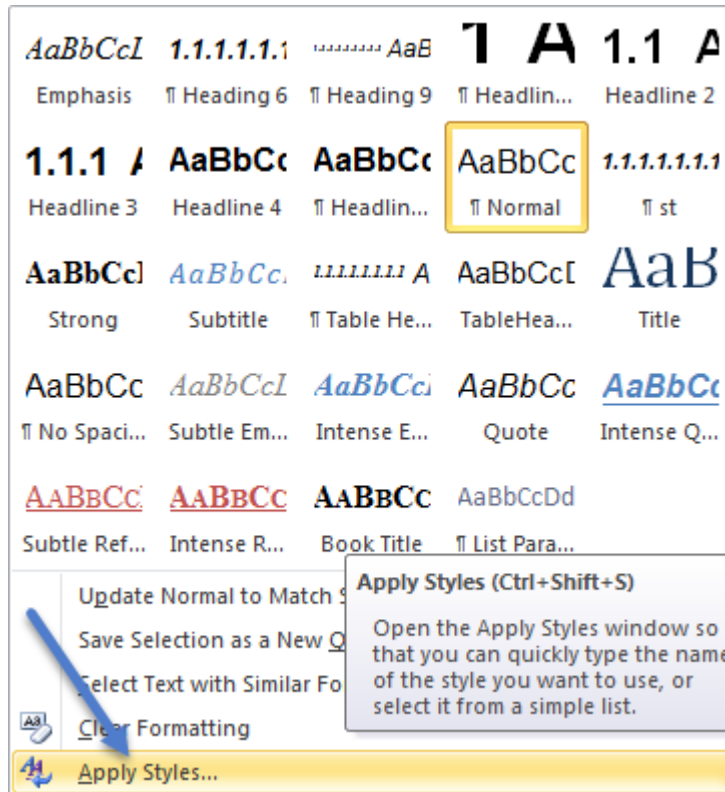


Figure 58: Apply styles

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

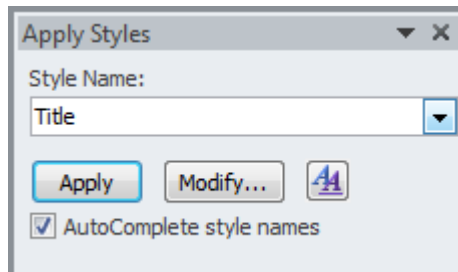


Figure 59: Style Dialog

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 55](#)
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\setings\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\setings\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\setings\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 , 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 Other Tools

Topics	9.1 Plant UML	97
--------	---------------------	----

9.1 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](#)]

Appendix A - Bibliography

Table 11: Bibliography

BibEnt	Description	Publisher
[AHF]	Antenna House Formatter V6 User Guide	Antenna House, Inc
[Ant]	Apache Ant 1.8.1 Manual	APACHE
[DitaSpec]	Darwin Information Typing Architecture (DITA) Version 1.2 OASIS Standard 1 December 2010 http://docs.oasis-open.org/dita/v1.2/os/spec/DITA1.2-spec.pdf	OASIS
[DtPrt]	DITA for Print: A DITA Open Toolkit Workbook DITA Open Toolkit 1.8 by Leigh W. White http://xmlpress.net	XmlPress
[ezRead]	ezRead Documentation Guide ezRead Documentation Technology and Tools, Helmut Scherzer http://www.hscherzer.de/dita.html/	none
[oxy17]	Oxygen XML Editor 17.0 User Guide Oxygen XML Editor User Manual http://www.oxygenxml.com	Syncro Soft SRL.
[PlantUML]	Drawing UML with PlantUML Language Reference Guide (Version 8031)	PlantUML project
[svgspec]	Scalable Vector Graphics (SVG) 1.1 Specification W3C Recommendation, 14 January 2003 http://www.w3.org/TR/2003/REC-SVG11-20030114/	W3C
[xslfo]	Extensible Stylesheet Language (XSL) Version 1.1 W3C Recommendation, 05 December 2006, Anders Berglund (IBM), mailto:alrb@us.ibm.com http://www.w3.org/TR/xsl11/	W3C

Table 11: Bibliography

BibEnt	Description	Publisher
[XslTut]	Beginning XSLT 2.0 From Novice to Professional, Jeni Tennison http://www.it-ebooks.info	Apress
[Xslt]	XSLT Mastering XSLT transformations, Doug, Tidwell http://www.oreilly.com	O'Reilly

Terms and Abbreviations

BMP	<p>The BMP file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows operating systems.</p> <p>The BMP file format is capable of storing two-dimensional digital images of arbitrary width, height, and resolution, both monochrome and color, in various color depths, and optionally with data compression, alpha channels, and color profiles.</p>
CHM	<p>Microsoft Compiled HTML Help is a Microsoft proprietary online help format, consisting of a collection of HTML pages, an index and other navigation tools. The files are compressed and deployed in a binary format with the extension .CHM, for Compiled HTML. The format is often used for software documentation. It was introduced as the successor to Microsoft WinHelp with the release of Windows 98 and is still supported in Windows 7. Although the format was designed by Microsoft, it has been successfully reverse-engineered and is now supported in many document viewer applications.</p>
DITA-OT	<p>DITA Open Toolkit</p> <p>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.</p>
ezRead	<p>ezRead is a product name for an Acrobat Plugin that allows sophisticated treatment of documents in particular with the focus to named destination. Find more in [ezRead].</p>
Oxygen	<p>oxygen XML editor</p>
PNG	<p>bla bla</p>
SVG	<p>Scalable Vector Graphic</p>
empty	<p>empty</p>

Index

S

security

sub 1

sub2 **60**