

MiramoPDF 2.0

DITA-OT plugin

Getting Started Guide

Contents

1	Introduction.....	1
1.1	GUI template controlled formatting.....	1
2	Installing.....	3
2.1	System requirements.....	3
2.2	Installing the Miramo DITA-OT plugin for the first time.....	3
2.2.1	Installing from the setup.exe [Windows only].....	3
2.2.2	Installing from ZIP archive [all platforms].....	4
2.3	Files and folders.....	4
3	Upgrading.....	7
3.1	Upgrading the Miramo DITA-OT plugin.....	7
3.2	Upgrading existing MFD files.....	7
3.3	Upgrading oXygen Transformation scenarios.....	7
3.4	Using the oXygen 'MiramoPDF MFD upgrade' transformation.....	8
3.5	Using the 'mfdupgrade' command line tool.....	8
3.6	Upgrading existing plugin customizations.....	8
4	Publishing to PDF.....	11
4.1	Workflow for developing customized PDFs.....	11
4.2	Creating a new MFD file.....	11
4.3	Publishing DITA content with PDF tooltips enabled.....	11
4.4	Editing the MFD template using MiramoDesigner.....	12
4.5	Ant build parameters.....	12
4.6	Controlling formatting using DITA @outputclass values.....	15
4.7	Mapping DITA elements to PDF document objects.....	17
4.7.1	Cover pages.....	17
4.7.2	Table of Contents (TOCs).....	17
4.7.3	Indexes.....	17
4.7.4	Table borders and rulings.....	18
4.7.5	Metadata elements to <VarDef> mapping.....	19
4.7.6	Adobe Acrobat PDF properties populated from DITA elements.....	20
4.7.7	DITA equation domain (<mathml>) support.....	21
4.8	Further information and help.....	22
5	Command line interface.....	23
5.1	Running the demo [Windows].....	23
5.2	Creating a PDF using dita2mmpdf [Windows].....	23
5.2.1	Using a different template and/or DITaval filter.....	23
5.2.2	Using the -dev option to display formatting properties in the output PDF.....	24
5.2.3	Using a different DITA input file.....	24
5.3	Creating a PDF using the <i>dita</i> command.....	24
5.4	Using ant to build the dita2mmpdf target.....	25
6	Multilingual publishing.....	27

Contents

6.1	How many MFD or template files will I need?.....	27
6.2	Language specific variables.....	28
6.2.1	Changing localization text.....	29
6.2.2	Example: extract from the 'cfg/strings/fr.xml' file.....	29
6.2.3	Changing existing language variables.....	30
6.2.4	Creating and using a new language variable.....	30
6.3	Configuring supported xml:lang values: langlist.xml.....	31
6.3.1	Modifying langlist.xml to add a new language.....	32
7	Customization.....	35
7.1	Customizing the com.miramo.mmpdf plugin.....	35
7.1.1	Creating a custom MiramoPDF DITA-OT plugin using oXygenXML Author/Editor.....	35
7.1.2	Manually creating a custom MiramoPDF DITA-OT plugin.....	38
7.1.3	com.example.my-mmpdf files and folders.....	40
7.1.4	Setting default parameter values.....	40
7.1.5	Modifying the dita2mmpdf default transformation.....	41
8	Integrating with oXygen.....	43
8.1	Installing and integrating the plugin with oXygen.....	43
8.1.1	Installing from the setup.exe (recommended).....	43
8.1.2	Installing from the ZIP file (advanced).....	43
8.1.3	Running the DITA-OT integrator [oXygen].....	43
8.1.4	Files and folders.....	44
8.2	Configuring oXygen.....	44
8.2.1	Import MiramoPDF transformation scenarios [oXygen].....	44
8.2.2	Make oXygen's content completion present the @outputclass values.....	44
8.3	Integrating with oXygen.....	45
9	Installing and integrating the plugin with XMetaL.....	47
9.1	Installing from the setup.exe.....	47
9.2	Files and folders.....	47
9.3	Configuring XMetaL.....	47
9.3.1	Create MiramoPDF 'deliverable types' in XMetaL.....	47
9.4	To test the integration.....	48

Chapter 1: Introduction

Thank you for your interest in the Miramo DITA Open Toolkit (OT) plugin, **com.miramo.mmpdf**, which provides a simple method for generating high-quality PDFs from DITA on Windows system using **MiramoPDF**, where formatting is controlled via a GUI template design tool, **MiramoDesigner**.

This Getting Started guide provides the information you need to get up and running publishing your DITA content, fast.

The plugin maps DITA to the MiramoXML model, then applies template-controlled formatting using **MiramoPDF**. It is designed to be easily integrated with your chosen XML authoring tool or content management system, or run with the standalone DITA-OT using a simple command line interface described in Command line interface on page 23.

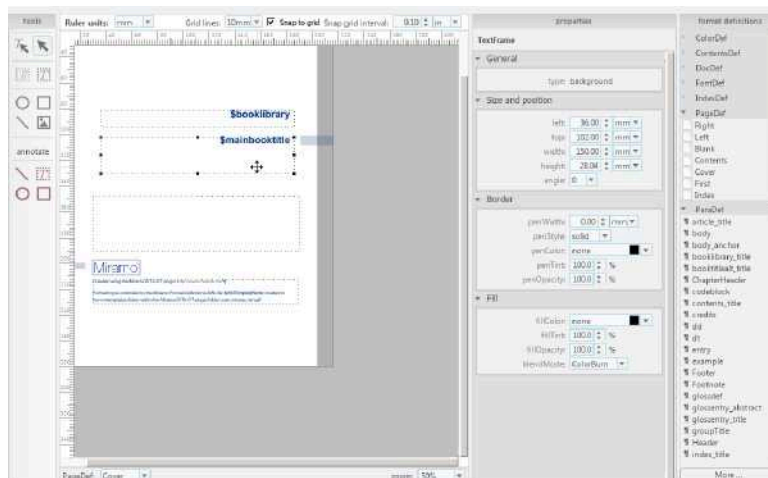
See Appendices for information on Installing and integrating the plugin with oXygen on page 43, and Installing and integrating the plugin with XMetaL on page 47.

The **com.miramo.mmpdf** plugin is designed to cover standard formatting requirements for DITA document content out of the box, but for bespoke requirements it may be customized to suit your needs - for information see Customization on page 35.

If you would like more information about server versions of Miramo, or if you need to publish non-DITA structured data, or would like to produce different output formats (FrameMaker, ePub, Kindle, HTML5 ...) please contact us (email miramo@datazone.com, phone +353 64 66 28964), and/or visit our website: <http://www.miramo.com>.

1.1 GUI template controlled formatting

The appearance of the output PDF is controlled by a **Miramo Format Definitions** (MFD) template - *the separation of form and content*. Templates are created and modified using **MiramoDesigner** (MiramoDesigner.exe), the GUI template design tool component of **MiramoPDF**.



The default template (default.mfd) supplied with the plugin provides a single-column layout for rendering MiramoXML to PDF. Here is an example: [flowers_default.pdf](#).

Chapter 2: Installing

The MiramoPDF DITA-OT plugin **com.miramo.mmpdf** is designed to be seamlessly integrated with any CMS or XML authoring tool such as oXygen or XMetaL. Alternatively it may be used with a standalone DITA-OT installation via the command line interface. This chapter gives an overview of installing or upgrading the plugin.

2.1 System requirements

Before installing the Miramo DITA-OT plugin the following software must be installed and working on the target system:

- DITA-OT version 2.4 or above
- **MiramoPDF** or Miramo Enterprise vs 2.0 or above

MiramoPDF Personal edition runs on the following operating systems:

- Windows 7 Professional (64 bit only)
- Windows 8.1 (64 bit only)
- Windows 10

Miramo Enterprise is available to run on Windows Server and Linux operating systems - contact us for more information.

2.2 Installing the Miramo DITA-OT plugin for the first time

These instructions relate to installing and integrating the Miramo DITA-OT plugin into an existing DITA-OT installation folder for the first time. For information on upgrading the plugin see [Upgrading the Miramo DITA-OT plugin on page 7](#).

To integrate with your XML authoring tool or DITA CMS, follow the instructions given by your vendor. See Appendices for information on Installing and integrating the plugin with oXygen on [page 43](#) and Installing and integrating the plugin with XMetaL on [page 47](#).

See Files and folders on [page 4](#) for a list of files and folders installed in the DITA-OT plugins/ com.miramo.mmpdf folder.

2.2.1 Installing from the setup.exe [Windows only]

Double-click on the MiramoDITA-OTplugin<vs>setup.exe and navigate to the DITA-OT install root folder, which contains the 'integrator.xml' file. The MiramoDITA-OT plugin source (including samples, command scripts and templates) will be located in the *[dita-ot install dir]/ plugins/ com.miramo.mmpdf* folder, for example:

C:\program files (x86)\dita-ot-3.3\plugins\com.miramo.mmpdf

The MiramoDITA-OTplugin<vs>setup.exe will run the DITA-OT integrator and will append the com.miramo.mmpdf folder to the PATH environment variable. The plugin is now ready to use.

See www.dita-ot.org for more information on installing and integrating DITA-OT plugins.

2.2.2 Installing from ZIP archive [all platforms]

Download the Miramo DITA-OT plugin ZIP file from www.miramo.com into a temporary folder *tmpdir*, then run the 'dita -install MiramoDITA-OTplugin<vs>.zip' command to install and register the plugin with the DITA-OT, for example:

```
cd tmpdir
dita-ot-installdir/bin/dita -install MiramoDITA-OTplugin200p01.zip
```

This will create a new folder,

```
dita-ot-installdir/plugins/com.miramo.mmpdf
```

See www.dita-ot.org for more information on installing and integrating DITA-OT plugins.

2.3 Files and folders

The **com.miramo.mmpdf** plugin includes the following files and folders, located in the DITA-OT installation folder selected during installation:

<dita-ot-installdir>/plugins/com.miramo.mmpdf/

Source for **com.miramo.mmpdf** plugin:

cfg/

XSL stylesheet placeholders and language specific localization strings. **Do not change the contents of this folder** - see Customizing the com.miramo.mmpdf plugin [on page 35](#) for information on generating a custom plugin based on com.miramo.mmpdf and [mmpdf:localization-strings.dir](#)

com.example.my-mmpdf/

Contains source files for making a specialization of the com.miramo.mmpdf plugin. See Customizing the com.miramo.mmpdf plugin [on page 35](#) for more details.

docs/Miramo_DITA-OT_GettingStarted.pdf

This document

images/

Images used by the com.miramo.mmpdf plugin

integration/oxygen/MiramoPDF.scenarios

The oXygen transformation scenario for producing a PDF from the currently edited DITA file. Import into your oXygen tool using Options->import transformations

integration/xmetal/*

Files needed for integrating with XMetaL.

logos/

Images and logos used by the com.miramo.mmpdf plugin

mmtemplates/

Miramo Format Definitions default template for DITA publishing:

Miramo Format Definitions templates are XML files which can be edited using the MiramoPDF GUI template design tool, **MiramoDesigner**.

default.mfd

default template used for publishing DITA content out of the box

samples/flowers/

The oXygen DITA flowers sample

samples/taskbook/

A modified version of the DITA-OT taskbook (IT book) sample

xsl

XSL stylesheets for mapping DITA to MiramoXML

dita2mm.cmd

Command script for preprocessing a DITA input file to MiramoXML, ready for rendering to PDF.
Called by:

dita2mmpdf.cmd

Command script for running the flowers demo, or for producing a PDF from a specified DITA map.

build_dita2mmpdf.xml

XML file which contains the ant targets for performing topicmerge and translating the DITA content to MiramoXML (Miramo Simple Markup, which represents the document content), and for rendering that document content to PDF using the format definitions in the MFD template.

Chapter 3: Upgrading

3.1 Upgrading the Miramo DITA-OT plugin

Double-click on the MiramoDITA-OTplugin<vs>setup.exe and navigate to the DITA-OT install root folder. If the target DITA-OT folder already contains an instance of the Miramo DITA-OT plugin com.miramo.mmpdf, the installer will automatically uninstall the existing plugin from the com.miramo.mmpdf plugin folder, and existing configuration files and XSL stylesheets in the com.miramo.mmpdf folder will be overwritten. If you wish to install multiple versions of the plugin, navigate to a different DITA-OT folder and choose to proceed without uninstalling the previous version when prompted.

If you have made changes to the 'default.mfd' template file located in the com.miramo.mmpdf \mmtemplates folder, the installer will make a backup copy of the 'default.mfd' to 'default-pre<vs>mfd', where <vs> is the version being installed for example **200p01**. Any other MFD files located in the mmtemplates folder will be left unchanged.

➤ **Important:** *Once the installation has completed, upgrade any existing MFD files (including the renamed 'default.mfd' if present) by following the instructions given below.*

3.2 Upgrading existing MFD files

Several new features and improvements have been introduced in version 2.0 of the MiramoPDF DITA-OT plugin, some of which have implications for upgrading existing MFD template files.

For example, the interparagraph spacing mode has changed since version 1.4 to remove inconsistencies. In the new version, spacing between paragraphs is exclusively controlled using @spaceAbove and @spaceBelow. If the values of these attributes is 0, there will be no space between paragraphs.

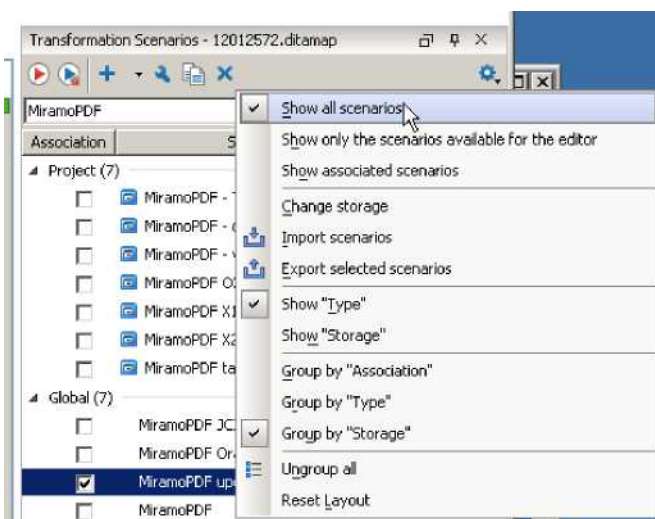
As a consequence the @spaceBelow values in ParaDef format definitions with a non-zero @leading value in MFD templates need to be adjusted to achieve the correct interparagraph spacing.

If you are using oXygen, a new transformation scenario 'MiramoPDF 2.0.0 MFD upgrade' has been added to streamline the upgrade process. See Using the oXygen 'MiramoPDF MFD upgrade' transformation [on page 8](#) for more information. Alternatively the MiramoPDF distribution includes a command line utility 'mfdupgrade.cmd' which automates the upgrade process.

Running the automated upgrade process on an MFD file '**filename.mfd**' results in a new MFD file '**filename_upgraded.mfd**'. Once you are happy with the results of the upgrade, this may be used in place of the original MFD file (but it is recommended that you make a backup copy first).

3.3 Upgrading oXygen Transformation scenarios

Before importing the transformation scenarios, open the *Transformation Scenarios* window and ensure that the **Show all scenarios** is set:



Import the 'MiramoPDF.scenarios' from the `<dita-ot>plugins/com.miramo.mmpdf/integration/oxygen` folder (see [Integrating with oXygen on page 43](#) for more information).

3.4 Using the oXygen 'MiramoPDF MFD upgrade' transformation

To use the MiramoPDF 2.0.0 MFD upgrade scenario:

- open the target MFD template as an XML file in oXygen editor
- apply the MiramoPDF 2.0.0 MFD upgrade transformation

This creates a new MFD file 'template_upgraded.mfd' and opens it using MiramoDesigner. The console window will display diagnostic information describing the changes made.

3.5 Using the 'mfdupgrade' command line tool

The 'mfdupgrade.cmd' script is installed in `c:\program files\Miramo\MiramoPDF` which is added to the PATH environment variable by the installer.

To upgrade your MFD file, start a new DOS command window and enter the following commands:

```
cd "foldercontainingMFDfile"
mfdupgrade template.mfd
```

The system will produce a log file 'mfdupgrade.log' in the current folder, and will automatically open the new **template_upgraded.mfd** using MiramoDesigner.

The MM_DITA_DIR environment variable is set to the location of the com.miramo.mmpdf plugin folder during installation of the DITA-OT plugin. The mfdupgrade utility copies any missing or updated Format Definitions from the DITA-OT plugin default template 'default.mfd' located in `%MM_DITA_DIR%/plugins/ com.miramo.mmpdf/mmtemplates`.

3.6 Upgrading existing plugin customizations

The MiramoPDF DITA-OT plugin customization structure has been significantly reworked in version 1.6, so the recommended upgrade procedure for upgrading from earlier versions is to create a new customization from scratch using the instructions given in [Manually creating a new](#)

custom plugin on page 38, then reapply any changes you have made to the 'cfg/custom.xml' to the new plugin folder.

For upgrading from version 1.6 and above, when installing a new version of the MiramoPDF DITA-OT plugin it may be necessary to update the 'build.xml' file for all existing customizations, so that parameters passed to the <xslt> target (eg 'my-mmpdf.merged2miramoxml') which maps from merged DITA to MiramoXML match those in the com.example.mmpdf/build.xml file to take advantage of any new 'ant' build parameters.

An example of the <param> values given in version 1.6 **com.example.my-mmpdf** is shown below. As new build parameters are introduced, the list of <param> values passed to the custom XSL stylesheet will need to be updated to reflect the new parameters:

```
<xslt in="${topic.merged.file}" out="${miramoxml.file}"
style="${dita.plugin.com.example.my-mmpdf.dir}/xsl/dita2miramo.xsl">
<xmlcatalog refid="dita.catalog"/>
<param name="transtype" expression="${transtype}"/>
<param name="images.dir" expression="${mmpdf:images.dir}"/>
<param name="images.suffix" expression="${mmpdf:images.suffix}"/>
<param name="defaultImageUnits"
expression="${mmpdf:defaultImageUnits}"/>
<param name="showStatus" expression="${mmpdf:showStatus}"/>
<param name="includeCover" expression="${mmpdf:includeCover}"/>
<param name="includeTOC" expression="${mmpdf:includeTOC}"/>
<param name="includeIndex" expression="${mmpdf:includeIndex}"/>
<param name="metadataprecedence"
expression="${mmpdf:metadataprecedence}"/>
<param name="defaultCellVerticalAlignment"
expression="${mmpdf:defaultCellVerticalAlignment}"/>
<param name="defaultImageAlignment"
expression="${mmpdf:defaultImageAlignment}"/>
<param name="args.rellinks" expression="${args.rellinks}"/>
<param name="args.draft" expression="${args.draft}"/>
<param name="formatSectionTitleAsTopicTitle"
expression="${mmpdf:formatSectionTitleAsTopicTitle}"/>
<param name="mfd.file.path" expression="${mfd.file.path}"/>
<param name="tableBorders" expression="${mmpdf:tableBorders}"/>
<param name="localization-strings.dir"
expression="${mmpdf:localization-strings.dir}"/>
</xslt>
```


Chapter 4: Publishing to PDF

This chapter covers using the MiramoPDF DITA-OT plugin and the GUI template designer, MiramoDesigner, to produce custom PDFs from DITA content. It includes information on using MiramoDesigner to customize a Miramo Format Definitions (MFD) template together with information on using ant build parameters and @outputclass attributes to customize the appearance of PDF output.

The plugin includes a default MFD template which can be used to publish DITA out of the box, either via your CMS, or using the MiramoPDF output transformation (oXygen) or deliverable (XMetaL), or via a simple command line interface. If the default template suits your needs, you can skip to Ant build parameters [on page 12](#). For information on using the command line interface see Command line interface [on page 23](#).

4.1 Workflow for developing customized PDFs

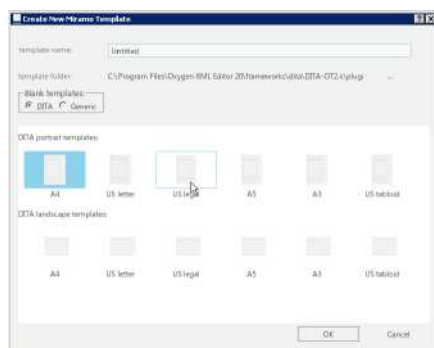
The workflow for developing a GUI template for customized PDFs is as follows:

- 1 Create a new DITA template (MFD) in the `com.miramo.mmpdf/mmtemplates` folder
- 2 Publish DITA content using the new MFD file with PDF tooltips enabled [`showProperties=Y`]
- 3 Review the PDF tooltips and decide which format definitions you would like to change
- 4 Use MiramoDesigner to modify the format definitions in the MFD template
- 5 Repeat steps 2 to 5 until you're happy with the output
- 6 Publish to production PDF with tooltips disabled [`showProperties=N`]

4.2 Creating a new MFD file

Use **File** > **SaveAs** to save your MFD file to the `com.miramo.mmpdf/mmtemplates` folder under a new name, eg `myTemplate.mfd`.

Start MiramoDesigner and either browse to the default.mfd located in `com.miramo.mmpdf/mmtemplates`, or click **New template** and choose the DITA template page size which suits best:



4.3 Publishing DITA content with PDF tooltips enabled

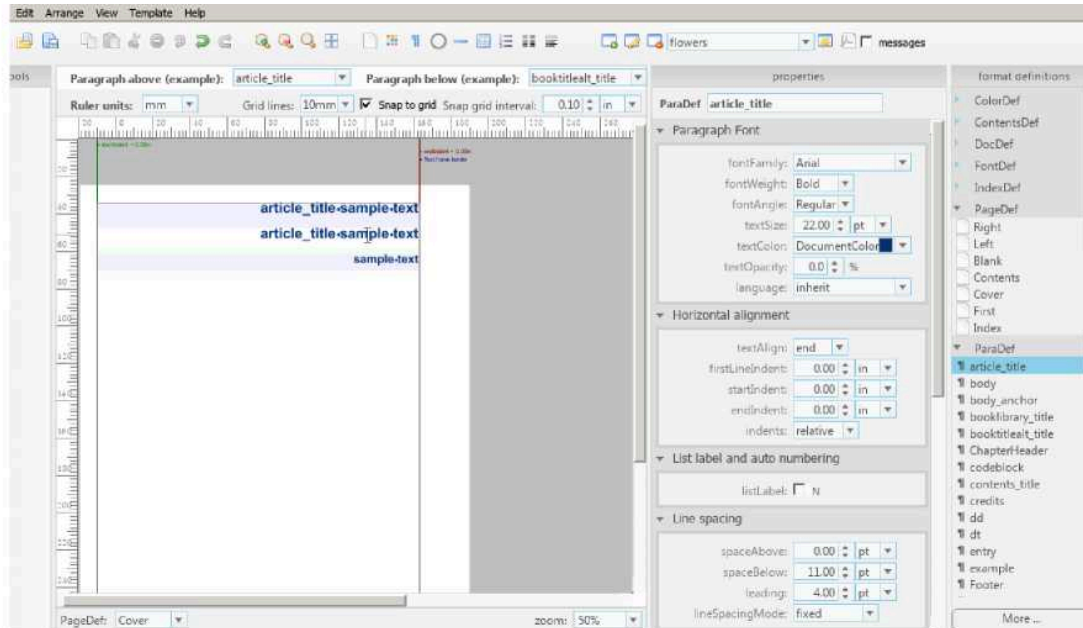
Set the ant build parameter `mmpdf:mfd.file` to the name of the template (eg `myTemplate.mfd`) and the `mmpdf:showProperties` build parameter to 'Y' to include PDF tooltips in the generated output, as allowed by your CMS or authoring tool. For example, If using oXygen, make a copy of the MiramoPDF - Dev output transformation, change the `mmpdf:mfd.file` parameter to 'myTemplate.mfd'

Publish your DITA content to produce a PDF with tooltips which indicate which format are applied, here's an example showing the 'article_title' <ParaDef> properties:

Once you have reviewed the PDF and decided which format definitions require updating you're ready to edit the MiramoDesigner template and republish.

4.4 Editing the MFD template using MiramoDesigner

Here is a screenshot of the main *MiramoDesigner* window, showing the properties of the *article_title* paragraph, which can be altered as required.



See videos on the [MiramoPDF YouTube channel](#) for tutorials on using MiramoDesigner.

4.5 Ant build parameters

The table below gives a list of the ant build parameters supported by the MiramoPDF DITA-OT plugin, which may be used to configure aspects of the PDF publishing process..

Table 1: ant build parameters used by the Miramo DITA-OT plugin

ant parameter	Default value	Description
args.input	\${cf}	Path to the source content being published.
args.draft	no	Set to 'yes' to include draft comments as 'comment' paragraphs in the PDF. If @author is specified the comment is wrapped in the correspondingly named FontDef (ignored unless FontDef is defined in the MFD file)
args.filter		Specifies a filter file to be used to include, exclude, or flag content, as part of the DITAOT preprocessing

Table 1: ant build parameters used by the Miramo DITA-OT plugin

ant parameter	Default value	Description
args.rellinks		Specifies which links to include in the output. Allowed values: "none" - No links are included. "all" - All links are included. "noparent" - Parent links are not included. "nofamily" - Parent, child, next, and previous links are not included.
clean.temp	yes	Specifies whether to remove temporary intermediate files from \${dita.temp.dir} Useful for debugging/development, merged DITA content is located in \${dita.temp.dir}. Allowed values: "yes" (default) "no"
dita.dir	\${configured.ditaot.dir}	Absolute path to the DITA Open Toolkit that is being used.
dita.temp.dir	\${cfd}/mmtmp	Specifies temporary folder for intermediate files.
output.dir	\${cfd}	Path to folder containing output PDF \${pdf.file}.
pdf.file	input file with file extension replaced by .pdf	Name of PDF file to be created in \${output.dir}.
transtype	mmpdf	Name of transformation type, modify only if using a customization of the com.miramo.mmpdf plugin
mmpdf:chapterContentsDef	chapterTOC	Set ContentsDef to use for formatting chapter TOC (miniTOC) - ignored unless mmpdf:includeChapterTOC = 'Y'
mmpdf:contentsDef	contents1	Set ContentsDef to use for formatting TOC - ignored if mmpdf:includeTOC = 'N'
mmpdf:copyImagesToOutput	Y	Set to Y to enable copy-image preprocessing step which copies image files to \$output.dir Allowed values: "Y" (default) "N"
mmpdf:cropMarks	N	Set to Y to include cropmarks in PDF output Allowed values: "N" (default) "Y"
mmpdf:cwd	\${cfd}	Current working directory for running MiramoPDF, defaults to folder containing dita input
mmpdf:defaultCellVerticalAlignment	middle	Set default vertical alignment of table cell contents Allowed values: "" (default) "top" "middle" "bottom"
mmpdf:defaultImageAlignment		Set default horizontal alignment for images Allowed values: "" (default) "start" "center" "end"
mmpdf:defaultImageUnits	px	Default units for image width/height values if none specified. Allowed values: "px" (default) "pt" "mm" "pc" "cm" "dd" "cc"
mmpdf:fontPath	default	Set folder or folders to search for fonts in addition to system fonts (default); use path separator ';' (windows) or ':' (linux/mac)

Table 1: ant build parameters used by the Miramo DITA-OT plugin

ant parameter	Default value	Description
mmpdf:formatDefinitionsAsTable	N	Set to Y to format definitions list as table rather than paragraphs Allowed values: "N" (default) "Y"
mmpdf:formatGlossaryAsTable	N	Set to Y to format glossary as table rather than paragraphs Allowed values: "N" (default) "Y"
mmpdf:formatSectionTitleAsTopicTitle	Y	Set to N to format section title elements as section_title* paragraphs, set to Y to format section title elements as topic_title* paragraphs
mmpdf:inlineImageVerticalOffset	Opt	Set default vertical shift for inline images - positive value moves image baseline down, percentage is relative to image height
mmpdf:images.dir	\${dita.dir}\plugins\com.miramo.mmpdf\images	Folder containing images for notes, defaults to \${dita.dir}/plugins/com.miramo.mmpdf/images
mmpdf:images.suffix	svg	File extension for images for notes, defaults to svg
mmpdf:includeBackCover	N	Set to Y to include back cover
mmpdf:includeChapterTOC	N	Set to Y to include chapter level TOC (minitoc)
mmpdf:includeCover	Y	Set to Y to include cover page, N to suppress cover generation
mmpdf:includeIndex	Y	Set to N to suppress index generation
mmpdf:includeTOC	Y	Set to N to suppress Table of Contents generation
mmpdf:indexDef	index1	Set IndexDef to use for formatting index - ignored if no indexterms or mmpdf:includeIndex = 'N'
mmpdf:jobID		<i>MiramoEnterprise only:</i> unique identifier for queued job as viewed in mmVisor
mmpdf:localization-strings.dir	\${plugin.dir}/cfg/strings	Location of strings folder containing langlist.xml and *.xml language files
mmpdf:logMetrics	N	Set to Y to create a metrics report detailing format definitions which are used but not defined in the specified MFD template Allowed values: "N" (default) "Y"
mmpdf:maxMemory	1GB	Set maximum memory usage for Miramo composer
mmpdf:metadataprecedence	map	Set to 'topic' to allow topic metadata data element values to override equivalent map metadata data element values. Default is 'map', where topic metadata is only used where no equivalent map data is defined Allowed values: "" (default) "map" "topic"
mmpdf:mfd.dir	\${dita.dir}\plugins\com.miramo.mmpdf\mmtemplates	Folder containing Miramo MFD template file, defaults to \${dita.dir}/plugins/com.miramo.mmpdf/mmtemplates
mmpdf:mfd.file	default.mfd	Miramo MFD template file, located in \${mfd.dir}, eg default.mfd

Table 1: ant build parameters used by the Miramo DITA-OT plugin

ant parameter	Default value	Description
mmpdf:pdfType	untagged	Set to UA to create structured PDF. Allowed values: "untagged" (default) "UA"
mmpdf:processGroup		<i>MiramoEnterprise only:</i> Name or number of MiramoEnterprise processing group as viewed in mmVisor
mmpdf:showStatus	N	Set to Y to highlight @status attributes; N for production Allowed values: "N" (default) "Y"
mmpdf:showProperties	N	Set to Y to include PDF tooltips showing Miramo format definitions; N for production Allowed values: "Y" (default) "N"
mmpdf:shrinkImagesToFit	N	Set to Y to shrink overlarge images with no @width or @height to fit container width/height Allowed values: "N" (default) "Y"
mmpdf:tableBorders	fromDITA	Set table border properties fromDITA (support @frame, @colsep and @rowsep) or fromT-bIDef (use TblDef rulings)
mmpdf:userVariable1		Set xslt user variable value 'userVariable1' for use in customizations
mmpdf:userVariable2		Set xslt user variable value 'userVariable2' for use in customizations
mmpdf:whiteSpaceMode	trimLeadingNewlines	By default, leading newline characters in paragraph text sections are trimmed; set to 'preserve' to treat as whitespace

4.6 Controlling formatting using DITA @outputclass values

The Miramo DITA-OT plugin allows page layouts, covers, TOC and Index generation to be controlled using one or more space separated 'mmpdf:' @outputclass values, as described in the table below:

Table 2: DITA @outputclass values

@outputclass value	DITA elements	Description
mmpdf:span	title, p, note, cmd	spans title across all columns in multi-column document
mmpdf:pageBreak	title, p, note, cmd	breaks page before title/p/note paragraph
mmpdf:columnBreak (alias for break-before)	title, p, note, cmd	force title/p/note paragraph to top of next column (or page, for single-column documents)
mmpdf:fontDef:name	ph	apply fontDef name to this phrase
mmpdf:keepTogether:bool	table, title, p, note, cmd	set keepTogether to 'Y' to prevent page or column breaks within the paragraph or table, for example to prevent page breaks before an <image> anchored within a paragraph with @placement = 'break'
mmpdf:paraDef:name	title, p, note, ol, ul	override default paraDef with paraDef name for rendering generated paragraph. For , elements, use named paraDef for child list items

Table 2: DITA @outputclass values

@outputclass value	DITA elements	Description
mmpdf:listName:name	ol, ul	specify basename for generating listItem paragraph names. default: orderedlistitem () or unorderedlistitem()
mmpdf:tblDef:name	fig, table, simpletable, reltable, hazardstatement	use tblDef name for rendering table, figure or hazardstatement
mmpdf:topRule:name	fig, table,	set top, bottom, start or end rule to ruleDef name.
mmpdf:bottomRule:name	simpletable, reltable, hazardstatement, row, strow,	If applied to a <fig>, <table>, <simpletable>, <reltable> or <hazardstatement>, the ruling applies to the outside edges of the table
mmpdf:startRule:name	entry, stentry	If applied to a <row> or <strow> element, the ruling establishes the default start/end/top/bottom ruling for all contained table cells.
mmpdf:endRule:name		If applied to a <entry> or <stentry> element, the ruling applies to the edges of the table cell.
mmpdf:maximumHeight:dim	row, strow	set maximum row height
mmpdf:minimumHeight:dim	row, strow	set minimum row height
mmpdf:spaceAbove:dim	title, p, note, cmd, table, simpletable, image	set spaceAbove
mmpdf:spaceBelow:dim	title, p, note, cmd, table, simpletable, image	set spaceBelow
mmpdf:splitRow:bool	row, strow	allow page or column breaks within row
mmpdf:splitStraddledRow:bool	row, strow	allow page or column breaks between straddled rows
mmpdf:colorDef:name	entry, stentry	use colorDef name for table cell fill
mmpdf:fillOpacity:percent	entry, stentry	use fillOpacity percent for cell fill color (ignored unless mmpdf:colorDef:name used) default: 100%
mmpdf:fillTint.percent	entry, stentry	use fillTint percent for cell fill color (ignored unless mmpdf:colorDef:name used) default: 100%
mmpdf:withNext: bool	title, p, note, cmd	set withNext to 'Y' or 'N' to set the paragraph keep with next property
mmpdf:withPrevious: bool	title, p, note, cmd	set withPrevious to 'Y' or 'N' to set the paragraph keep with previous property
mmpdf:xRefDef:name	xref	Use XRefDef name for this cross reference
mmpdf:section:name	map, bookmap, title, topic, topicref, chapter, appendix, booktitle, table	start new section using SectionDef 'name'. SectionDef is defined in the MFD template or using <SectionDef sectionDef="name"../> and starts a new page layout sequence
mmpdf:coverSection:name	map, bookmap	Use sectionDef 'name' for cover and subsequent pages. Ignored if mmpdf:includeCover is set to N. default: coverSection
mmpdf:backCoverSection:name	map, bookmap	Apply sectionDef 'name' to end of PDF. Ignored unless mmpdf:includeBackCover is set to Y. default: backCoverSection

Table 2: DITA @outputclass values

@outputclass value	DITA elements	Description
mmpdf:contentsSection:name	map, bookmap	Use sectionDef name for TOC and subsequent pages. Ignored if mmpdf:includeTOC is set to N. default value: contentsSection
mmpdf:indexSection:name	map, bookmap	Use sectionDef name for Index and subsequent pages. Ignored if mmpdf:includeIndex is set to N, or document contains no indexterms. default: indexSection

4.7 Mapping DITA elements to PDF document objects

This section covers some general rules for the default mapping of DITA elements to PDF document objects and format definitions. To override aspects of the default mapping which can't be achieved using ant build parameters, MFD template changes or @outputclass values, it may be necessary to create a customized plugin - see Customizing the com.miramo.mmpdf plugin on [page 35](#) for more information.

4.7.1 Cover pages

By default, front cover pages are included under the following conditions:

- map: map element has @title attribute or child <title> element
- bookmap: bookmap element contains child <booktitle> element

The front cover is added as a result of applying the coverSection SectionDef from the specified MFD file. To suppress cover generation in the above cases, **set mmpdf:includeCover=N**

Back cover pages may be added to the PDF generated from a map or bookmap if and only if the ant build parameter mmpdf:includeBackCover is set to Y. In this case the backCoverSection SectionDef is applied to the end of the generated PDF.

4.7.2 Table of Contents (TOC)s

By default, TOCs are included under the following condition:

bookmap: bookmap element contains a <booklists> <toc> element, for example:

```
<frontmatter>
  <booklists>
    <toc/>
  </booklists>
```

To suppress TOC generation in the above case, set **mmpdf:includeTOC=N**

4.7.3 Indexes

By default, Indexes are included under the following conditions:

- DITA source contains one or more <indexterm> elements
- bookmap: bookmap element contains a <booklists> <indexlist> element, for example:

```
<backmatter>
```

```
<booklists>
  <indexlist/>
</booklists>
```

To suppress index generation in the above cases, set **mmpdf:includeIndex=N**

4.7.4 Table borders and rulings

The **mmpdf:tableBorders** ant build parameter value controls whether table rulings are controlled by the MFD template (**mmpdf:tableBorders** = 'fromTblDef') or whether they can be overridden via the DITA **@frame**, **@colsep** and **@rowsep** parameters (**mmpdf:tableBorders** = 'fromDITA') as illustrated in the tables below. The appearance of the 'all', 'none', 'topbot', 'sides', 'top', 'bottom' and 'Very Thin' is controlled by the MFD template **<RuleDef>** settings.

Table 3: <table> @frame attribute to MiramoXML <Tbl> RuleDef mapping [mmpdf:tableBorders = fromDITA]

@frame attribute value	topRule	bottomRule	startRule	endRule
all	all	all	all	all
none	none	none	none	none
topbot	topbot	topbot	none	none
sides	none	none	sides	sides
top	top	none	none	none
@frame attribute value	topRule	bottomRule	startRule	endRule
bottom	none	bottom	none	none

If the **mmpdf:tableBorders** ant build parameter value is set to 'fromDITA' (default) the following rules apply for mapping inner table rulings:

Table 4: <tgroup> @colsep, @rowsep attributes to MiramoXML <Tbl> RuleDef mapping [mmpdf:tableBorders = fromDITA]

	columnRule	rowRule
@colsep = "0"	none	from <TblDef>
@colsep = "1"	Very Thin	from <TblDef>
@rowsep = "0"	from <TblDef>	none
@rowsep = "1"	from <TblDef>	Very Thin

Additional **@outputclass** values such as **mmpdf:topRule:namemay** be specified to override top, bottom, start and end rulings on a table, row, or entry (cell), regardless of the value set for **mmpdf:tableBorders**, or the value of the **@frame**, **@colsep** or **@rowsep** attributes.

See Controlling formatting using DITA **@outputclass** values on page 15 for further information.

4.7.5 Metadata elements to <VarDef> mapping

Metadata elements in the DITA are mapped to MiramoXML <VarDef> elements which are used to populate MiramoDesigner variable placeholders inserted in background (running header/footer) text frames in the MFD template. The variables listed in Table 5 on page 19 are available in the default template and are populated from metadata elements, if present and not empty.

In addition, certain DITA metadata elements are mapped to MiramoXML <MetaData> elements which are used to populate PDF metadata.

For example, the following DITA <author> element:

```
<author>Howe Tudit</author>
```

is mapped to this <MiramoXML>:

```
<VarDef varDef="author">Howe Tudit</VarDef>
<MetaData name="creator">Howe Tudit</MetaData>
```

PDF metadata may be viewed using a tool such as Adobe Acrobat, or interrogated by third-party software. See Table 6 on page 19, and Adobe Acrobat PDF properties populated from DITA elements on page 20.

Metadata <data> or <othermeta> elements may be used to extend the set of variables and/or PDF metadata:

- <data> elements are mapped to a <VarDef> whose name is mapped from the <data> element @name attribute.
- a <data> element with @outputclass attribute set to 'mmpdf:metadata:name' may be used to map to a MiramoXML <MetaData> element, which is used to populate PDF metadata.

Table 5: MiramoDesigner metadata variables

author	featnum
booklibrary	mainbooktitle
brand	prognum
booktitlealt	prodinfo
category	prodname
component	platform
copyrholder	series
copyryear	vrmlist (space separated list of versions)
emailaddress	vrmlversion, vrmlrelease, vrmlmodification

See Adobe Acrobat PDF properties populated from DITA elements on page 20 for a screenshot illustrating the PDF document properties populated from the taskbook.ditamap supplied in the samples/ taskbook folder.

Table 6: DITA to PDF xmp metadata mapping

DITA source element/attribute	MiramoXML <MetaData> @name	PDF xmp metadata element
-------------------------------	----------------------------	--------------------------

Table 6: DITA to PDF xmp metadata mapping

author	creator	dc:creator
keyword	keywords	pdf:Keywords dc:subject
bookmap/@title bookmap/title bookmap/mainbooktitle map/@title map/title	title	dc:title
created/@date	createdDate YYYY-MM-DD	xmp:CreateDate
revised/@modified	modifiedDate YYYY-MM-DD	xmp:ModifyDate
copyright/copyrholder copyyear/@year	rights	dc:rights

4.7.6 Adobe Acrobat PDF properties populated from DITA elements

Below is an annotated screenshot illustrating the mapping from DITA markup to PDF document properties, as viewed using Adobe Acrobat:



4.7.7 DITA equation domain (<mathml>) support

The plugin supports the following DITA math elements:

- <equation-block>
Use the <equation-block> element to represent an equation that is presented as a separate block within a text flow. Block equations can be numbered. <equation-block> elements are wrapped in an 'equation-block' FontDef.
- equation-figure
Use the <equation-figure> element to represent an equation that functions as form of figure or display. Display equations are intended to be numbered when numbering is desired. <equation-figure> equations are wrapped in a 'equation-figure' table, which allows for different numbering sequences for equation titles.
- equation-inline
Use the <equation-inline> element to represent an equation that is presented inline within a paragraph or similar context. Inline equations are not intended to be numbered. <equation-inline> equations are wrapped in an 'equation-inline' FontDef.

'equation-inline' and 'equation-block' FontDefs may be used to set default font characteristics (font family, pointsize and text color) for <mathml> equations.

4.8 Further information and help

See the [Miramo mmComposer Reference Guide \(PDF\)](#) for more information about the MiramoXML simple markup.

See the [MiramoDesigner videos on the Miramo Datazone YouTube](#) channel for more information about MiramoDesigner

Check our website at: <http://www.miramo.com> or phone us on +353 64 66 28964.

And you are always welcome to email miramo@datazone.com with comments, criticisms and questions - your feedback would be much appreciated.

Chapter 5: Command line interface

This chapter covers using the MiramoPDF DITA-OT plugin via a simple command line interface [Windows only] or ant, suitable for use with a standalone DITA-OT installation.

5.1 Running the demo [Windows]

Start a console window, or powershell window and enter:

```
dita2mmpdf -dev
```

If your PATH environment variable is set correctly this will run the *dita2mmpdf.cmd* script in the *[dita-ot install dir][dita-ot install dir]/plugins/com.miramo.mmpdf* folder.

i Note: *NOTE that the first time you run the demo it may take several minutes as mmComposer builds its internal font cache*

This calls the **dita2mm** script to perform a topic merge on the flowers.ditamap, and to produce an intermediate flowers_miramo.xml file. The resulting XML is processed to PDF by Miramo.

When Miramo has finished creating the PDF you should see a message like this in the console window:

```
mmComposer: 1 : Completed with 0 errors and 0 warnings.  
PDF file <com.miramo.mmpdf>\samples\flowers\flowers.pdf createdFinished.
```

5.2 Creating a PDF using dita2mmpdf [Windows]

The dita2mmpdf script has the following usage:

```
dita2mmpdf [-dev] [-ditaval file.ditaval] [ditafile [template.mfd [pdffile]]]
```

5.2.1 Using a different template and/or DITAVAL filter

To run the flowers sample through using a different 'twocol.mfd' template (not supplied) which you have created in the mmtemplates folder, use this command from the plugins/com.miramo.mmpdf subfolder in the DITA-OT install folder:

```
dita2mmpdf samples/flowers/flowers.ditamap twocol.mfd
```

This will create the file samples/flowers/flowers.pdf (the PDF file defaults to the basename of the DITA file, with a .pdf extension) using the 'twocol.mfd' template (*not supplied*).

To create a differently named PDF, specify it as the last argument to dita2mmpdf, for example:

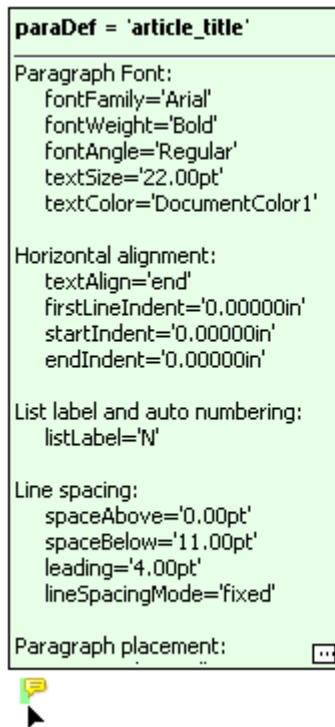
```
dita2mmpdf samples/flowers/flowers.ditamap twocol.mfd ff.pdf
```

To apply a DITAVAL filter 'print.ditaval' (not supplied) use the -ditaval command line option, eg:

```
dita2mmpdf -ditaval print.ditaval samples/taskbook.ditamap
```

5.2.2 Using the -dev option to display formatting properties in the output PDF

The PDF files created using dita2mmpdf can be set to contain tooltips which give formatting information for paragraph tags (green), font tags (red), and table tags (blue). Here is an example showing the article_title paragraph in the flowers.pdf file produced using the default template:



Growing Flowers

The tooltips are shown if the -dev command line option is supplied to the dita2mmpdf.cmd. (The dita2mmpdf -dev option sets the MiramoPDF command line option '-showProperties Y'). By default tooltips are not shown.

5.2.3 Using a different DITA input file

To run the DITA file through to a pdf with the same basename as the DITA file, use:

```
dita2mmpdf <path to ditafile>
```

or add the appropriate template file and output PDF name, eg:

```
dita2mmpdf sample.dita mytemplate.mfd mysample.pdf
```

In this case 'mytemplate.mfd' must exist in the *plugins/com.miramo.mmpdf/mmtemplates* folder.

5.3 Creating a PDF using the *dita* command

You can use either the dita command-line tool or ant to transform DITA content to PDF using the Miramo DITA-OT plugin:

```
dita -f mmpdf -i <path to ditafile> [options]
```

The above assumes that folder containing dita.bat is included in the PATH environment variable, for example using the following powershell command:

```
$env:PATH += ";$env:MM_DITA_DIR/bin"
```

Here is an example of using the 'dita' -f mmpdf command line tool to publish flowers.ditamap to a PDF in the default output folder, './out':

```
dita -f mmpdf -i flowers.ditamap
```

Any of the Ant build parameters [on page 12](#) may be specified, for example to include PDF tooltip properties in the output PDF, and to write to pdf file 'myflowers.pdf' in the current folder:

```
dita -f mmpdf -i flowers.ditamap "-Dmmpdf:showProperties=Y" "-Dpdf.file=myflowers.pdf"
```

See [dita command arguments and options](http://www.dita-ot.org) on the <http://www.dita-ot.org> website for more information, and Ant build parameters [on page 12](#) for a list of supported parameter values.

5.4 Using ant to build the dita2mmpdf target

Alternatively you can use ant to build the 'dita2mmpdf' target using the following syntax (powershell):

```
ant -f "$env:MM_DITA_DIR/build.xml" ["-Dparam=value" "..."] dita2mmpdf
```

The above assumes that folder containing ant.bat is included in the PATH environment variable, for example using the following powershell command:

```
$env:PATH += ";$env:MM_DITA_DIR/bin" # powershell
```

Here is an example creating flowers.pdf from the samples/flowers folder by running this command from the powershell:

```
cd "env:MM_DITA_DIR"
ant -f "$env:MM_DITA_DIR/build.xml" "-Dargs.input=samples/flowers/flowers.ditamap" "-Dmmpdf:showProperties=Y" "-Dpdf.file=flowers.pdf" dita2mmpdf
```

Here is another example, where the test.ditamap is located in "c:\users\joanne\documents", and the MFD file is called 'test.mfd' and is located in c:\Users\Joanne\documents:

```
cd c:\Users\Joanne\documents
ant -f "$env:MM_DITA_DIR/build.xml" "-Dargs.input=samples/flowers/flowers.ditamap" "-Dmmpdf:showProperties=Y" "-Dmmpdf:mfd.file=test.mfd" "-Dmmpdf:mfd.dir=c:\Users\Joanne\documents" "-Dpdf.file=flowers.pdf" dita2mmpdf
```

See Ant build parameters [on page 12](#) for a list of parameters supported by the MiramoPDF DITA-OT plugin.

Chapter 6: Multilingual publishing

Multilingual publishing presents challenges when deciding whether to use a single MFD template to control the appearance of multilingual documents, or to use one MFD file per document language: both approaches have advantages and disadvantages.

The Miramo DITA-OT plugin handles multilingual publishing through language specific variables (see Language specific variables [on page 28](#)), localization text configuration (see) and configuring the mapping of xml:lang attribute values to MiramoXML (see Configuring supported xml:lang values: langlist.xml [on page 31](#)).

Here are some points to consider when creating MiramoDesigner templates for use within a multilingual publishing system:

6.1 How many MFD or template files will I need?

The choice of font used for publishing is a key consideration for multilingual publishing: no single font supports all unicode characters, and some languages (eg Traditional vs Simplified Chinese) require different fonts, as the same unicode number may be rendered as a radically different glyph in the different languages.

For example, here is the same phrase published in Simplified Chinese, Traditional Chinese and Japanese:

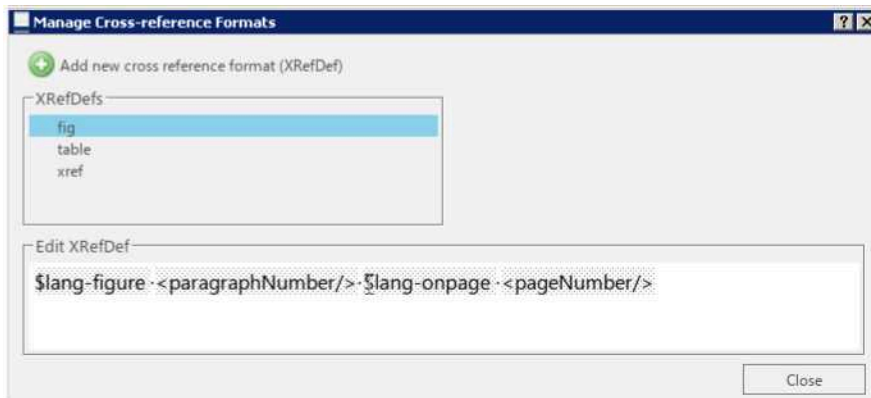
Simplified Chinese 朝辞白帝彩云间

Traditional Chinese 朝辭白帝彩雲間

Japanese 朝に辞す白帝彩雲の間

Generally, it is usually appropriate to use one MFD template for 'latin' languages (eg Italian, French, English, Spanish, Dutch) and a separate MFD template file for each other language which would require a different base font (eg Simplified Chinese, Arabic, Hebrew).

6.2 Language specific variables



The language specific variables above (eg \$lang-figure, \$lang-onpage) are mapped to the appropriate language text, depending on the @xml:lang value specified for the DITA document or chapter.

Language specific variables may be used to create cross-reference formats, running headers and footers and figure/table numbering appropriate for multilingual publishing. For example, here is a language independent cross-reference format:

The following are some of the variables available in the default template which are populated from language specific strings in plugin folder *cfg/strings/*.xml*.

Table 7: Examples of MiramoDesigner language specific variables

MiramoDesigner variable name	Language specific text (english)
lang-appendix	Appendix
lang-contents	Contents
lang-figure	Figure
lang-index	Index
lang-onpage	on page
lang-page	page
lang-table	Table
lang-related	Related Links

For a complete list of language specific variables available by default, see the langlist.xml file in com.miramo.mmpdf/cfg/strings/langlist.xml, and the corresponding language files in the com.miramo.mmpdf/cfg/strings/*.xml

► **Important:** Do not change the localization strings in the original com.miramo.mmpdf/cfg/strings/*.xml files, as these changes will be overwritten if you upgrade. Always make changes to a copy of the strings folder, either by manually copying the folder to a new location outside the com.miramo.mmpdf and setting the ant build parameter *mmpdf:localization-strings.dir* to the new folder, or add a new custom plugin (Customizing the com.miramo.mmpdf plugin on page 35) and change the strings in the cfg/strings folder within the new plugin.

6.2.1 Changing localization text

By default, localization text strings are located in the plugin folder's `cfg/strings/*.xml` files. If you wish to keep a copy of the strings folder elsewhere, copy the folder and set `thempdf.localization-strings.dir` ant build parameter to the new location.

► **Important:** Do not change the localization text in the original `com.miramo.mmpdf/cfg/strings` folder, the changes will be overwritten if you upgrade.

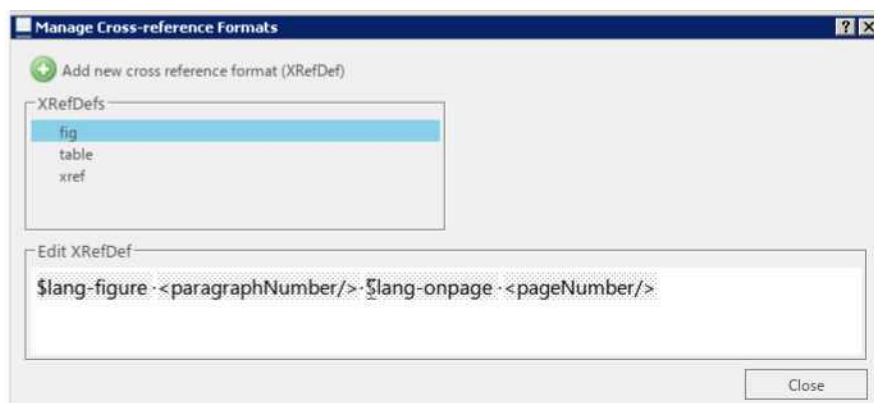
Each language specific .xml file contains a set of `<str>` elements, each of which maps to a MiramoXML language dependent `<VarDef>` whose name is prefixed with 'lang-'.

Every MiramoDesigner DITA template contains a set of language independent placeholder variables (`<VarDef>`s), which can be inserted into background text frames, or used as a building block, for example within a cross-reference (`<XRefDef>`) format definition.

Here is a screenshot of a language specific variable 'lang-contents' which has been inserted into the header of a Contents PageDef:



Below is a language independent Cross Reference format which uses the lang-figure and lang-onpage `<VarDef>`s:



When the DITA is rendered to PDF, the placeholder variable will be replaced by the language specific string taken from the appropriate `cfg/strings//language.xml`, determined by the `xml:lang` setting applicable to the current document or chapter.

6.2.2 Example: extract from the 'cfg/strings/fr.xml' file

For example, here is an extract from the start of the default 'fr.xml' supplied with the `com.example.my-mmpdf` plugin:

```

<strings xml:lang="fr">
<str id="lang-fig">Illustration </str>
<str id="lang-table">Tableau </str>
<str id="lang-chapter">Chapitre </str>
<str id="lang-appendix">Annexe </str>
<str id="lang-page">la page</str>
<str id="lang-optional">Facultatif :</str>
<str id="lang-required">Obligatoire :</str>
<str id="lang-onpage"> a la page </str>
<str id="lang-prereq" dita-ot-id="task prereq"/>
<str id="lang-postreq" dita-ot-id="task postreq"/>
<str id="lang-example" dita-ot-id="task example"/>

```

This file is used to populate a set of MiramoDesigner language independent <VarDef>s as follows.

If present, @dita-ot-id specifies the DITA-OT language variable or string id to look up, defined in:

```

plugin:org.dita.base:xsl/common/strings-*.xml
plugin:org.dita.pdf2/cfg/common/vars/*.xml

```

If there is no @dita-ot-id attribute, the value is taken from the <str> element contents.

6.2.3 Changing existing language variables

To change the value used to populate a <VarDef> which retrieves the value of a DITA-OT language string, remove the @dita-ot-id attribute and add the required text to the <str> element.

For example, to suppress the standard note separator ':' which is added as a label suffix for <note> paragraphs, change from:

```
<str id="lang-note-separator" dita-ot-id="#note-separator"/>
```

to:

```
<str id="lang-note-separator"/>
```

Here is another example where the standard text used to label a <prereq> paragraph is changed from the default DITA-OT string 'Avant de commencer' to 'Avant:' change from:

```
<str id="lang-prereq">Avant de commencer </str>
```

to:

```
<str id="lang-prereq">Avant: </str>
```

Be sure to run the DITA-OT integration tool to apply the changes: Running the DITA-OT integration tool [on page 39](#).

6.2.4 Creating and using a new language variable

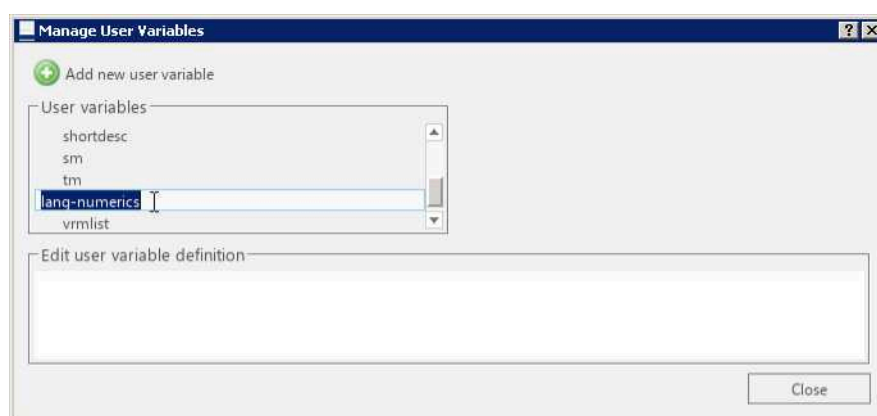
First, add new entry for the new <VarDef> to each of the language.xml files you support for publishing your DITA content.

Here are two examples added to the language file 'nl.xml' (Dutch language). The first exposes the DITA-OT language string Numerics into a <VarDef> named 'lang-numerics'. The second creates a new <VarDef> named 'lang-languagename':

```
<strings xml:lang="nl">
<str id="lang-numerics" dita-ot-id="Numerics"/>
<str id="lang-languagename">Nederlands</str>
```

Next, create matching empty placeholder <VarDef>s 'lang-numerics' and 'lang-languagename' in your MFD template:

Click on **More** at the bottom of the *Format Definitions* pane and click on **VarDef** to bring up the *Manage Variables* dialog, then click on the **Add new user variable** button to add the new lang-numerics and lang-languagename placeholder variables:



These placeholder variables can now be inserted in background text frames. Double-click on a background text frame to locate the insertion point, then right-click to bring up the context menu, choose 'Insert variable' and click on 'lang-languagename'. The screenshot on the right shows the \$lang-languagename placeholder in a rotated background text frame:

6.3 Configuring supported xml:lang values: langlist.xml

The com.miramo.mmpdf/cfg/strings/langlist.xml file controls the mapping of xml:lang values specified in DITA markup, and configures the filename containing localization text. The 'cfg/strings/*.xml' contain language specific strings used to construct language specific <VarDef>s which are mapped to the language string appropriate to the xml:lang value as part of the publication process.

If you are considering adding support for an xml:lang value which is not in the DITA-OT by default, always make changes to a copy of the strings folder, either by manually copying the folder to a new location outside the com.miramo.mmpdf and setting the ant build parameter `mmpdf.localization-strings.dir` to the new folder, or add a new custom plugin (Customizing the com.miramo.mmpdf plugin on page 35) and change the strings in the *cfg/strings* folder within the new plugin.

Here are some extracts from the default langlist.xml file, starting at line 169. See comments for an explanation of attribute values:

```

<!--
<!-- MiramoPDF plugin config starts here: add or remove entries below

Each xml:lang value below includes the following attributes:
  filename
    specifies file containing language specific strings
  rtl
    set to "true" for RTL languages such as Arabic (ar) and Hebrew (he)
  requiresFontSwitch
    set to 'true' if language requires a switch to a different (non-latin
    script) font for example, if a phrase with @xml:lang='zh-cn'
    (simplified chinese) is encountered within a DITA document with
    @xml:lang='us-en', this would require a switch to a different fontDef:
    lang-zh-cn
  miramoLanguage
    MiramoXML @language attribute value: controls font script, text
    direction and hyphenation characteristics
-->
<lang xml:lang="en-us" filename="en.xml" miramoLanguage="usEnglish"/>

<lang xml:lang="ar" filename="ar.xml" rtl="true" requiresFontSwitch="true"
miramoLanguage="Arabic"/>
<lang xml:lang="ar-eg" filename="ar.xml" rtl="true"
requiresFontSwitch="true" miramoLanguage="Arabic"/>
<lang xml:lang="ar-ae" filename="ar.xml" rtl="true"
requiresFontSwitch="true" miramoLanguage="Arabic"/>
<lang xml:lang="be" filename="be.xml"/>
<lang xml:lang="be-by" filename="be.xml"/>
<lang xml:lang="bg" filename="bg.xml" miramoLanguage="Bulgarian"/>
<lang xml:lang="bg-bg" filename="bg.xml" miramoLanguage="Bulgarian"/>
<lang xml:lang="bs" filename="bs.xml" miramoLanguage="Bosnian"/>
<lang xml:lang="bs-ba" filename="bs.xml" miramoLanguage="Bosnian"/>
...

```

6.3.1 Modifying langlist.xml to add a new language

Note: This assumes you are working on a copy of the 'strings' folder, either in a customization or a folder located using the 'mmpdf:localization-strings.dir' build parameter.

To add a support for a new xml:lang value which is not in the DITA-OT by default, first create a new language localization files in the strings folder, and register them by adding <lang> entries to the langlist.xml file. This section describes adding support for Tamil, language code 'ta' but the principle can be applied to any language supported by the MiramoPDF composition engine.

6.3.1.1 Reusing existing localization variables from org.dita.pdf2

If you have already created a PDF2 localization variables file for the new language 'ta.xml' located in the DITA-OT/plugins/org.dita.pdf2/common/vars folder, you can reuse it by copying it to a 'ta-dita-vars.xml' file in the target strings folder, alongside the 'langlist.xml' file and the MiramoPDF 'ta.xml' strings file. (note: copying the file to cfg/common/vars, as is commonly done for PDF2 transformations, will have no effect).

To register the 'ta-dita-vars.xml' file add the following entry to the end of the 'langlist.xml' file:

```
<lang xml:lang="ta" filename="ta-dita-vars.xml"/>
```

6.3.1.2 Reusing generated text strings from org.dita.base

If you have already added a set of HTML localization strings located in DITA-OT/plugins/org.dita.base/xsl/common, eg strings-ta-ta.xml, you can reuse it by copying it to 'ta-dita-strings.xml' file in the target strings folder. To register the 'ta-dita-strings.xml' file, add the following entry to the end of the 'langlist.xml' file:

```
<lang xml:lang="ta" filename="ta-dita-strings.xml"/>
```

6.3.1.3 Creating a new MiramoPDF language file

To add a support for a new xml:lang value which is not in the DITA-OT by default, first create a new MiramoPDF localization strings file by copying an existing file (eg cfg/strings/en.xml), then modify the <string> entries as required.

For example, to add support for Tamil, make a copy of the strings/en.xml to 'ta.xml' and make changes to that file. Note that the 'dita-ot-id' attribute may be used to reference an existing PDF2 variable, or HTML language string located in the 'ta-dita-vars.xml' or 'ta-dita-strings.xml' files. Alternatively, the @dita-ot-id attribute may be removed in which case the <string>content defined in ta.xml file will be used. See [Changing existing language variables on page 30](#) for more information.

Register the MiramoPDF language file 'ta.xml' by adding a new entry to end of the langlist.xml file, with the filename attribute set to the location of the new language file, and the @miramoLanguage attribute set to a supported language name (see MiramoDesigner <DocDef> @language attribute for a list of values).

Here is an example of a language entry for Tamil, which has a language code 'ta', or 'ta-ta':

```
<!-- Add ta.xml which has MiramoPDF strings to vardefs mappings -->
<lang xml:lang="ta" filename="ta.xml" miramoLanguage="Tamil"/>
<lang xml:lang="ta-ta" filename="ta.xml" miramoLanguage="Tamil"/>
```

6.3.1.4 @xml:lang to MiramoXML mapping

The DITA-OT processes @xml:lang values encountered in the inputstream as follows:

When the DITA-OT plugin encounters an @xml:lang attribute value which differs from the document @xml:lang value, it checks the langlist.xml file to see if a font switch is required (@requiresFontSwitch="true"); if so, it applies the FontDef named 'lang-lower-case(@xml:lang)' - for example, if an Traditional Chinese table like this is encountered in a us-en (US english, default) document:

```
<table xml:lang="zh-TW">
```

it is mapped to this MiramoXML:

```
<Tbl language="TraditionalChinese"><!-- language attribute determines
hyphenation-->
```

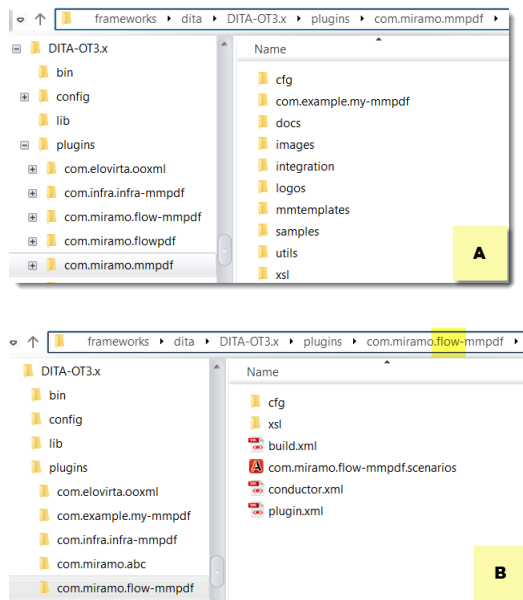
and all the paragraphs within the <table> contents of the <table> are formatted with the Font-Def named 'lang-zh-tw'.

Similarly, if a `` (bold) or `<i>` (italic) element is encountered within the context of an `@xml:lang` value which differs from the document `@xml:lang`, it is mapped to a `FontDef` named `'b-lower-case(@xml:lang)'` or `'i-lower-case(@xml:lang)'` - for example, `b-zh-tw` or `i-ar`. However, if the language context of the `` or `<i>` element is the same as the document language context, the mapping is to `FontDef 'b'` or `'i'`, as usual.

Chapter 7: Customization

7.1 Customizing the com.miramo.mmpdf plugin

The **com.miramo.mmpdf** plugin is intended to cover most formatting requirements, but in some cases it may be necessary to create a custom plugin which overrides or adds to some of the features supported by the standard plugin. If so, **do not** modify the files in the default MiramoPDF DITA-OT plugin installation folder (*DITA-OT3.x\plugins\com.miramo.mmpdf*), because these files will be overwritten when you install a new version of the MiramoPDF DITA-OT plugin. Create a custom MiramoPDF DITA-OT plugin instead.



A
Default MiramoPDF DITA-OT plugin folder

B
Custom MiramoPDF DITA-OT plugin folder

If you have access to oXygenXML Author/Editor see [Creating a custom MiramoPDF DITA-OT plugin using oXygenXML Author/Editor on page 35](#), otherwise follow the instructions in [Manually creating a new custom plugin on page 38](#) or [Manually creating a company specific plugin on page 39](#)

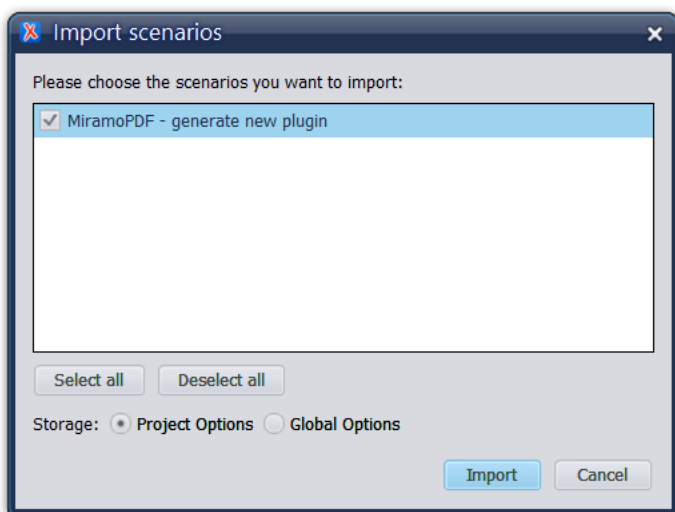
It also includes a simple example of working with a new plugin (com.example.my-mmpdf) to create a new transformation type my-mmpdf which includes changes to localization text, default parameter values and to the DITA to MiramoXML transformation.


7.1.1 Creating a custom MiramoPDF DITA-OT plugin using oXygenXML Author/Editor

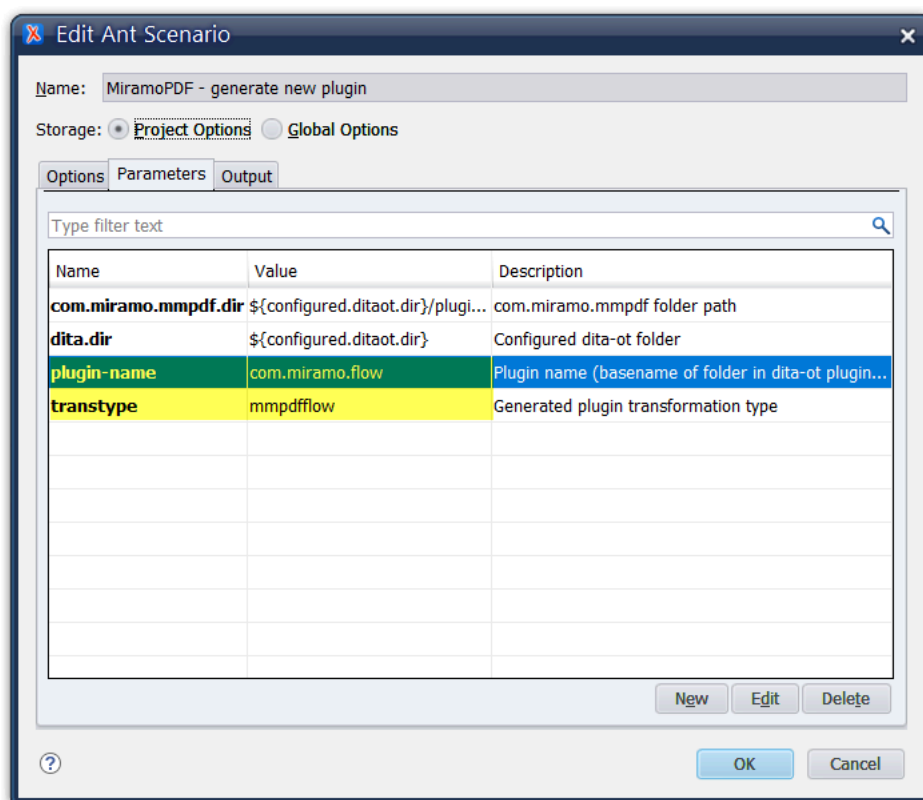
This topic describes how to use the **MiramoPDF - generate new plugin** transformation scenario in oXygenXML Author/Editor to create a new custom plugin.

- 1 In oXygenXML Author/Editor, choose **Options > Import Transformation Scenarios**.

- 2 Navigate to your MiramoPDF DITA-OT plugin installation folder, for example `C:\Program Files\Oxygen XML Editor 21\frameworks\dita\DITA-OT3.x\plugins\com.miramo.mmpdf\integration\oxygen`.
- 3 Select `MiramoPDF-generate-plugin.scenarios` and click **Open**.
- 4 In the *Import scenarios* dialog, select **Project Options** and click **Import**.

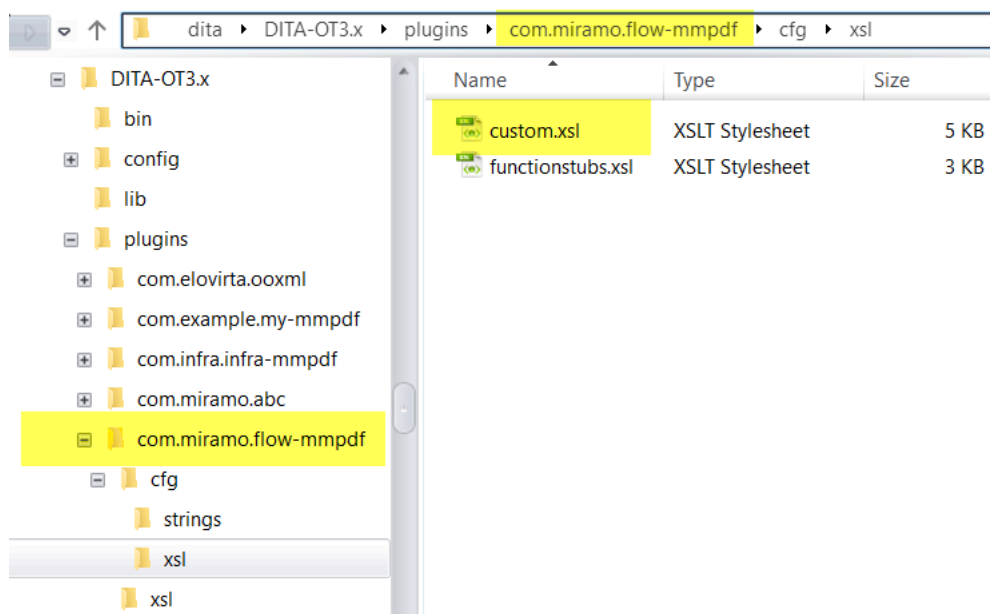


- 5 Create or open any DITA file.
- 6 You can open a DITA file in the following *Samples* folder:
`C:\Program Files\Oxygen XML Editor 21\frameworks\dita\DITA-OT3.x\plugins\com.miramo.mmpdf\samples\flowers`
- 7 Click the **Configure Transformation Scenario**  button.
- 8 In the *Configure Transformation Scenarios* dialog, select the **MiramoPDF - generate new plugin** transformation scenario and click **Edit**.
- 9 Click the **Parameters** tab and change the **plugin-name** and **transtype** parameters, as shown in the example below.
- 10 A **transtype** is a new transformation type with any new custom parameters that are supported. In this case, the **transtype** is the MiramoPDF transformation with customizations done in a *custom.xsl* file.



11 Click **OK** and then **Apply associated** to run the transformation scenario.

In your DITA-OT installation folder, you will now find your custom Miramo plugin folder where you can make the necessary customizations in the `cfg\xsl\custom.xml` file.



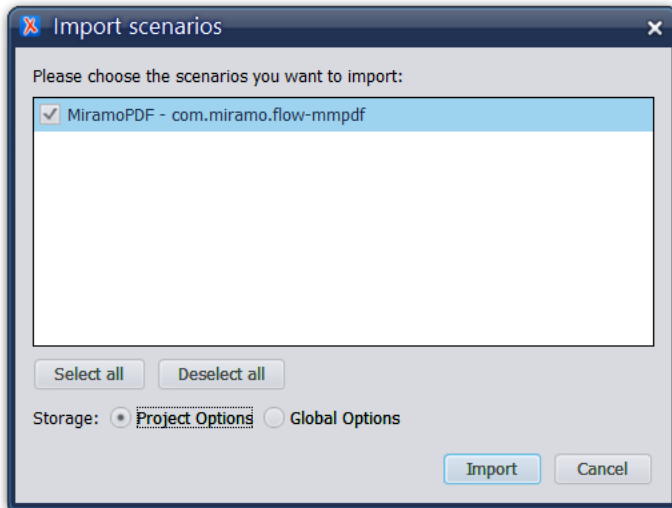
12 Choose **Options** > **Import Transformation Scenarios**.


13 Navigate to your newly created, custom MiramoPDF DITA-OT plugin folder.

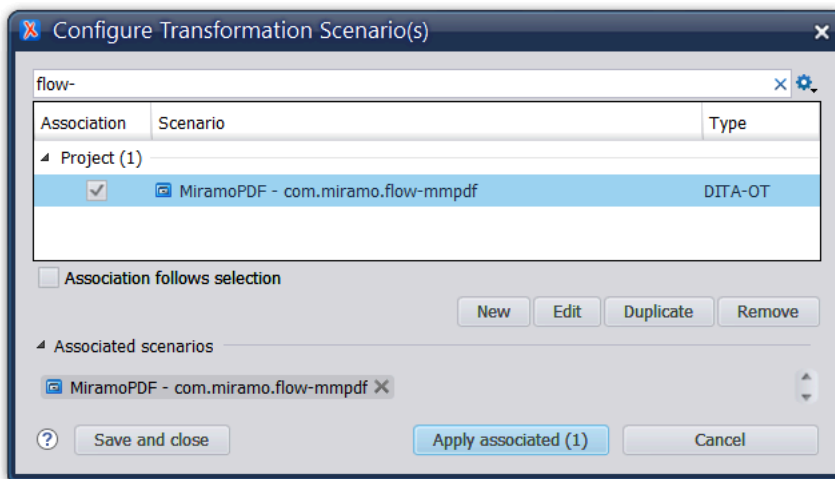
For example: `C:\Program Files\Oxygen XML Editor 21\frameworks\dita\DITA-OT3.x\plugins\com.miramo.flow-mmpdf`.

14 Select the .scenarios file, for example `com.miramo.flow-mmpdf.scenarios`, and click **Open**.

15 In the *Import scenarios* dialog, select **Project Options** and click **Import**.



16 Click the **Configure Transformation Scenario**  button again.
Your new custom Miramo transformation scenario is now ready for use.



17 You can click **Apply associated** to test your custom transformation scenario.

7.1.2 Manually creating a custom MiramoPDF DITA-OT plugin

7.1.2.1 Manually creating a new custom plugin

The `com.miramo.mmpdf` folder includes a sub-folder called `com.example.my-mmpdf`, and creates a transformation type 'my-mmpdf'. See [Manually creating a company specific plugin](#) on

[page 39](#) for information on creating a company specific plugin with a differently named plugin and transformation type.

To create a new custom plugin 'com.example.my-mmpdf', start by copying this folder to the DITA-OT plugins folder so it is at the same level as the original com.miramo.mmpdf folder.

7.1.2.2 Manually creating a company specific plugin

In some cases it may be necessary to create one or more new plugins with a different names, targets, and transformation types.

For example, company ABC Corp might need to create two specializations, com.abc.mmpdf1 and com.abc.mmpdf2, which create transformation types mmpdf1 and mmpdf2 respectively.

To do this, copy the *plugins/com.miramo.mmpdf/com.example.my-mmpdf* folder to *plugins/com.abc.mmpdf1* and *plugins/com.abc.mmpdf2*.

Then edit the following files in each of the two new folders com.abc.mmpdf1, com.abc.mmpdf2:

- plugin.xml
- conductor.xml
- build.xml

Replace all occurrences of 'com.example.my-mmpdf' with com.abc.mmpdf1 or com.abc.mmpdf2

Replace all occurrences of 'my-mmpdf' with 'mmpdf1' or 'mmpdf2'

Modify *xsl/dita2miramo.xsl*, *cfg/xsl/custom.xsl*, *cfg/strings/langlist.xml*, *cfg/strings/*.xml* as required

Run the DITA-OT integration process

The ant targets 'dita2mmpdf1' and 'dita2mmpdf2' will now be available, along with the transformation types 'mmpdf1' and 'mmpdf2'.

7.1.2.3 Running the DITA-OT integration tool

Next, install the new com.example.my-mmpdf plugin by running the DITA-OT ant integration tool as enabled by your CMS or authoring tool, or using one of the commands below.

For oXygen, see Running the DITA-OT integrator [oXygen] [on page 43](#).

For standalone DITA-OT installations, use:

```
ant -f integrator.xml
```

or use the dita command:

```
dita-ot-dir/bin/dita --install
```

This will create a new transformation type 'my-mmpdf' which you can use in place of the 'mmpdf' transformation type. It will also add a new ant target 'dita2my-mmpdf'.

The next stage is to modify the files in the *plugins/com.example.my-mmpdf* folder.

7.1.3 com.example.my-mmpdf files and folders

The com.example.my-mmpdf plugin includes the following files and folders:

<dita-ot-installdir>/plugins/com.example.my-mmpdf

Source for my-mmpdf plugin:

plugin.xml

Configuration file for the com.example.my-mmpdf plugin which creates a new transformation type 'my-mmpdf'.

conductor.xml

Imports local build.xml

build.xml

Contains the dita2my-mmpdf target, applied by the 'my-mmpdf' transformation type.

xsl/dita2miramo.xsl

Top level stylesheet - modify this file to set default values for xslt parameters

cfg/strings/langlist.xml

Language configuration file including a list of files containing localization strings

cfg/strings/*.xml

Miramo specific localization strings, supplied in addition to the standard DITA-OT localization strings (including those in pdf2)

cfg/xsl/custom.xsl

Modify this file to add xsl stylesheet template match modifications; included by xsl/dita2miramo.xsl

cfg/xsl/functionstubs.xsl

Modify this file to add to or override templates and functions with extension points defined in the com.miramo.mmpdf/xsl/utilities.xsl. Included by cfg/xsl/custom.xsl

7.1.4 Setting default parameter values

Modify the <xsl:param> and <xsl:variable> definitions in *custom/xsl/dita2miramo.xsl* as required. For example, to set all non-inline graphics to be centered rather than start-aligned by default, include this:

```
<!-- Default left/right alignment of images within text column or text frame.
Allowed values: start center end -->
```

```
<xsl:variable name="defaultImageAlignment">center</xsl:variable>
```

7.1.5 Modifying the dita2mmpdf default transformation

Edit the 'cfg/xsl/custom.xsl' stylesheet to add <xsl:template>s to achieve the required mapping from DITA elements and attributes to MiramoXML elements and attributes. (See the mmComposer reference guide for a detailed description of the MiramoXML file format).

<xsl:template>s added here may either override or add to the <xsl:template>s defined in the default DITA to MiramoXML stylesheets, map2miramo.xsl and topic2miramo.xsl. Here is a simple example where if the <p> @outputclass value is specified, it is used as the <ParaDef> format name. If the @outputclass value is empty, the default template match is applied using <xsl:apply-imports/>.

```
<xsl:template match="*[contains(@class,' topic/p ')]">
<xsl:choose>
<xsl:when test="string-length(@outputclass) != 0">
<xsl:call-template name="makePara">
<xsl:with-param name="paraDefName">
<xsl:value-of select="@outputclass"/>
</xsl:with-param>
</xsl:call-template>
</xsl:when>
<xsl:otherwise>
<xsl:apply-imports/>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
```


Chapter 8: Integrating with oXygen

8.1 Installing and integrating the plugin with oXygen

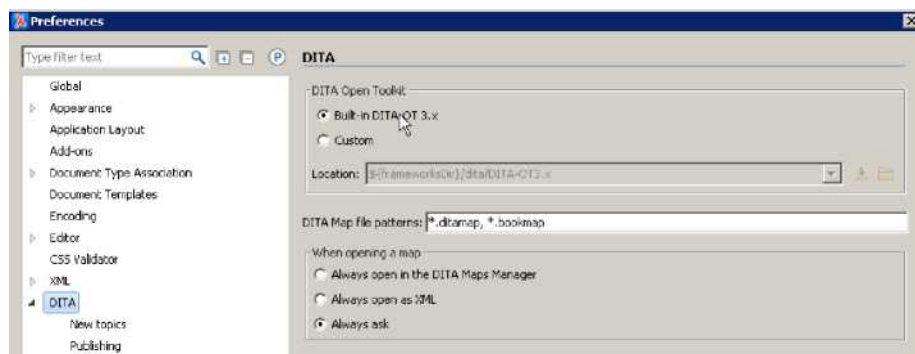
8.1.1 Installing from the setup.exe (recommended)

Double-click on the MiramoDITA-OTplugin<vs>setup.exe and navigate to the DITA-OT install root folder.

For oXygen (version 18 or above), the built-in DITA-OT folder is located in *[oxygen_install_dir]/frameworks/dita/DITA-OT3.x/plugins/com.miramo.mmpdf*, for example:

c:/program files/Oxygen XML Editor 21/frameworks/dita/DITA-OT3.x/plugins/ com.miramo.mmpdf

This corresponds to the oXygen DITA Open Toolkit 'Built-in DITA-OT 3.x' location preference shown here (options->preferences->DITA):



Note that if you choose a different folder you will have to set the dita.dir variable in the MiramoPDF transformation scenarios, or change the oXygen DITA-OT configured install folder in the preferences.

The MiramoDITA-OT plugin source (including samples, command scripts and templates) will be located in the *[dita-ot install dir]/plugins/com.miramo.mmpdf* folder, for example:

[oxygen_install_dir]/frameworks/dita/DITA-OT3.x/plugins/com.miramo.mmpdf

The MiramoDITA-OTplugin<vs>setup.exe will run the DITA-OT integrator and will append the com.miramo.mmpdf folder to the PATH environment variable.

8.1.2 Installing from the ZIP file (advanced)

Extract the MiramoDITA-OTplugin<vs>.zip file into the DITA-OT plugins folder.

For oXygen, the built-in DITA-OT plugins folder is located in *[oxygen_install_dir]/frameworks/dita/DITA-OT3.x/plugins/com.miramo.mmpdf*, for example:

C:\program files\Oxygen XML Editor 21\frameworks\dita\DITA-OT3.x\plugins

8.1.3 Running the DITA-OT integrator [oXygen]

Start your preferred oXygen product, open a DITA file or map in the editor window and run the predefined transformation scenario called **Run DITA OT Integrator**. execute it from the Docu-

ment->Transformation- >Apply Transformation Scenario(s) dialog box. If the integrator is not visible, select the Show all scenarios option that is available in the Settings drop-down menu.

8.1.4 Files and folders

See Files and folders on [page 4](#) for a list of files and folders which are installed in the *plugins\com.miramo.mmpdf* folder.

8.2 Configuring oXygen

8.2.1 Import MiramoPDF transformation scenarios [oXygen]

Once the com.miramo.mmpdf folder is installed in the DITA-OT plugins folder, set up the MiramoPDF transformation scenario: Choose options->import transformation scenarios, then navigate to the MiramoPDF.scenarios file located in the com.miramo.mmpdf/integration/oxygen subfolder (by default, *[oxygen_install_dir]/frameworks/dita/DITA-OT3.x/plugins/com.miramo.mmpdf/integration/oxygen*).

This creates the **MiramoPDF** and **MiramoPDF dev** (with PDF tooltips) transformation scenarios, which are set up to produce a PDF from the currently edited DITA file, and which can be applied and modified as required. See Ant build parameters on [page 12](#) for more information.

8.2.2 Make oXygen's content completion present the @outputclass values

To configure oXygen's content completion, it is necessary to copy and paste the match elements contained in the cc_config_miramo.xml file supplied to the DITA framework you are using.

The cc_config_miramo.xml file is located here:

[oxygen_install_dir]/frameworks/dita/DITA-OT3.x/plugins/com.miramo.mmpdf/integration/oxygen/cc_config_miramo.xml

and contains entries like this:

```
<match elementName="p" attributeName="outputclass">
<items action="append">
<item value="mmpdf:span" annotation="Spans title across all columns in multi-column
document"/>
<item value="mmpdf:pageBreak" annotation="breaks page before title/p/note paragraph"/>
<item value="mmpdf:columnBreak" annotation="force title/p/note paragraph to top of next
column
(or page, for singlecolumn documents)"/>
<item value="mmpdf:section:name" annotation="start new section using SectionDef
'name'.
SectionDef is defined in the MFD template"/>
</items>
</match>
```

Inside your DITA framework there is a configuration file named cc_config.xml. If you are using the built-in DITA framework then this file is located at *[OXYGEN_INSTALL_DIR]/frameworks/dita/resources/cc_config.xml*.

If you don't know exactly what framework you are using or where it is located then you can find out like this:

- 1 Go to **Window** > **Show view** and open the Properties view.

- 2 Open a DITA topic.
- 3 Look inside the Properties view at the Document type name. Now you know the name of your framework.
- 4 Go to **Options** > **Preferences...**
- 5 On the Document Type Associations page, identify and edit the framework with the previously discovered name. The Storage field will reveal its location.

8.3 Integrating with oXygen

The <match> entries for allowing content completion for the MiramoPDF @outputclass values are located in the *[oxygen_install_dir]/frameworks/dita/DITA-OT3.x/plugins/com.miramo.mmpdf/integration/oxygen/cc_config_miramo.xml* file.

Open it and copy and paste the entries inside the <config> root element in your DITA framework cc_config.xml file.

Chapter 9: Installing and integrating the plugin with XMetaL

This section covers integration with XMetaL Author Enterprise 12 and above.

9.1 Installing from the setup.exe

Double-click on the MiramoDITA-OTplugin<vs>setup.exe and navigate to the DITA-OT install root folder.

For XMetaL Author Enterprise (version 12 or above), the built-in DITA-OT folder is located in the Windows program data directory for XMetaL for example:

C:\ProgramData\SoftQuad\XMetaL\Shared\DITA_OT2.4

The MiramoDITA-OT plugin source (including samples, command scripts and templates) will be located in:

C:\ProgramData\SoftQuad\XMetaL\Shared\DITA_OT2.4\plugins\com.miramo.mmpdf

The MiramoDITA-OTplugin<vs>setup.exe will run the DITA-OT integrator and will append the com.miramo.mmpdf folder to the PATH environment variable.

9.2 Files and folders

See Files and folders on page 4 for a list of files and folders which are installed in the *plugins\com.miramo.mmpdf* folder.

9.3 Configuring XMetaL

9.3.1 Create MiramoPDF 'deliverable types' in XMetaL

Once the com.miramo.mmpdf folder is installed in the DITA-OT plugins folder the next step is to set up two MiramoPDF deliverable types so they are available when the user selects Generate Output.

- 1 Copy the two configuration files: print_miramo_pdf.xml and print_miramo_pdf_dev.xml from:

C:\ProgramData\SoftQuad\XMetaL\Shared\DITA_OT2.4\plugins\com.miramo.mmpdf\integration\xmetal

to:

C:\Program Files\XMetaL 12.0\Author\DITA\XACs\shared\renditions

- 2 Start XMetaL.
- 3 In **Tools** > **Configure output**, click on **Add**.

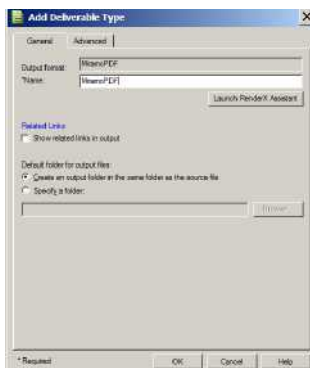


The *Add Deliverable Type* dialog appears.

- 4 From the list, select **MiramPDF**.



- 5 Click **Continue** (button)
- 6 In the **Name** field enter 'MiramoPDF' as the name you would like to appear in the list of deliverable types in XMetaL's "Generate Output for DITA Map" dialog and click **Ok**.



- 7 Switch to the **Advanced** tab. In the **Configure output options** text box, add this line:

```
run_plugin_integrator = yes
```

- 8 Repeat steps 3 to 7: Except in step 4 select 'MiramoPDF_Dev' and in step 6 enter 'MiramoPDF Development' in the **Name** field.

9.4 To test the integration

- 1 Open a DITA map in XMetaL.
- 2 Select **File** > **Generate Output for DITA Map ...**
- 3 Select 'MiramoPDF'.