DITA for Print: A DITA Open Toolkit Workbook

DITA Open Toolkit 1.8

Leigh W. White

XML PRESS
http://xmlpress.net

DITA for Print: A DITA Open Toolkit Workbook

Credits

DITA and the DITA logo are used by permission from the OASIS open standards consortium.

Disclaimer

The information in this book is provided on an "as is" basis, without warranty. While every effort has been taken by the author and XML Press in the preparation of this book, the author and XML Press shall have neither liability nor responsibility to any person or entity with respect to loss or damages arising from the information contained in this book.

This book contains links to third-party web sites that are not under the control of the author or XML Press. The author and XML Press are not responsible for the content of any linked site. Inclusion of a link in this book does not imply that the author or XML Press endorses or accepts any responsibility for the content of that third-party site.

The author and XML Press are not responsible for any damage to equipment or loss of data that might occur as a result of downloading and/or installing any software mentioned herein. Nor are the author and XML Press responsible for any penalties that might be incurred due to illegal download, installation or usage of any software mentioned herein.

Trademarks

XML Press and the XML Press logo are trademarks of XML Press.

All terms mentioned in this book that are known to be trademarks or service marks have been capitalized as appropriate. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

XML Press
Laguna Hills, CA
http://xmlpress.net

First Edition
ISBN:

Table of Contents

Introduction

Useful resources

It takes a village…

Because DITA is an open standard, thousands of people use it, and most of those people are happy to help you. If you haven't already done so, consider joining the Yahoo! DITA Users list. The list is extremely active and well-monitored. The Search feature isn't the best, but you should still try to search the archives as thoroughly as you can to make sure you don't ask a question that's already been asked and answered many times.

The SuiteShare Social Knowledgebase, sponsored by Suite Solutions, is also up and running now and growing every day. It contains articles that address many of the issues discussed on the DITA Users list, but the articles tend to be a little tighter than some of the DITA Users discussions.

The OASIS DITA Users list is an alternative to the Yahoo! DITA Users list. It's currently not as active as the Yahoo list, but there is a large, searchable archive.

If you need a brush-up on DITA itself, there are several very good books available that can get you going. There's a list at the end of this section.

These lists are by no means exhaustive. Just enter "DITA" into a search engine and you'll see that you could easily spend a lifetime exploring all the information available. Don't be distracted by Ms. Von Teese.

Websites

DITA 1.2 specification: **http://docs.oasis-open.org/dita/v1.2/spec/DITA1.2-spec.html**

Yahoo! DITA Users: **tech.groups.yahoo.com/group/dita-users**

OASIS DITA Users list: **https://lists.oasis-open.org/archives/dita-users/**

DITA Social knowledgeBase: **ditadocs.content-lifecycle.com**

Jarno Elovirta's web-based PDF plugin generator: **http://dita-generator.appspot.com/pdf-plugin/**

Patrick Quinlan's (Ditanauts.org) **mypdf** plugin: **http://sourceforge.net/p/mypdf/home/Home/**

Eliot Kimber's specialization tutorials: **http://xiruss.org/tutorials/dita-specialization/**

Guide to ANT build properties:**dita-ot.sourceforge.net/1.8/readme/dita-ot_ant_properties.html**

List of entity codes: **http://www.entitycode.com**

Apache ANT: **http://ant.apache.org/**

Apache ANT manual: **http://ant.apache.org/manual/index.html**

Books

Introduction to DITA: A User Guide to the Darwin Information Typing Architecture (Jennifer Linton, Kylene Bruski)
DITA 101: Fundamentals of DITA for Authors and Managers (Ann Rockley, Steve Manning and Charles Cooper)
Practical DITA (Julio Vazquez)
DITA Best Practices: A Roadmap for Writing, Editing, and Architecting in DITA (Laura Bellamy, Michelle Carey, Jenifer Schlotfeldt)
DITA for Practitioners, Volume I: Architecture and Technology (Eliot Kimber)
DITA Style Guide (ePub) (Tony Self)
XSL Formatting Objects Developer's Handbook (Doug Lovell)
XSL-FO (Dave Pawson)

The DITA Open Toolkit itself

After you work your way through this book, if you're eager to learn more about customizing the PDF plugin (or any other kind of plugin), you can turn to the Developer Reference documentation within the DITA Open Toolkit folder itself.

This documentation is found in `DITA-OT/doc/dev_ref` and it provides a wealth of information about the different processing stages of a DITA Open Toolkit build, as well as details on extending and further customizing plugins.

This information is delivered as a map and source topics, so you'll probably want to build a PDF to browse it easily, which you will be able to do after you finish the first few chapters of this book!

Some conventions

Book organization

Each chapter addresses one particular aspect of a print publication. Every chapter has sections with names prefixed by the word **Exercise**. These sections contain the exercises you need to complete to set up a PDF plugin that meets the specifications outlined in the **Specifications used in these exercises** *(on page ?)* appendix. Some chapters have other exercises you can complete to do other cool things with your PDF plugin. These exercises can be found in the **Other things you can do** section in those chapters.

Folder names

For the sake of simplicity, in all the examples and paths throughout this book the DITA Open Toolkit folder is called `DITA-OT`. If you download the DITA Open Toolkit 1.8, you'll see that by default, the folder is named DITA-OT1.8. You can leave that folder name as-is or change it as you like. Just keep in mind that DITA-OT in file paths in this book refers to whatever you have actually named your folder, so edit the paths you use accordingly.

Paths

Most of the code samples and paths in the book use the URL syntax convention of forward slashes because DITA and XSL-FO are Web applications. Windows systems should accept the forward slashes as well in almost all cases, though you will see backslashed in some Windows-only command lines

Type styles

The following type styles are used to make certain items more obvious.

- **Bold**: file names, emphasized sections in code samples, attribute values and variable values.
- *Italic*: attribute set names, attribute names, variable names, template names, parameter names, terms defined in the glossary
- `Monospaced`: file paths, folder names, code samples, element names (DITA, XSL, and FO)
- Double quotes: marker names
- ¶: indicates a line return has been inserted in a code example for clarity (usually because without the line break, the code would not fit on the printed page); the actual code does not include a line return at that location.

Order of exercises

You can do most of the exercises in any order. You might not always get exactly the results you expect, especially with respect to formatting, but your build will work. There are a few sets of exercises that must be done in a specific order because each exercise depends on variables, attribute sets, or page layouts created in a previous exercise. These exercises are grouped together with a note recommending that you complete them in order.

Comment, comment, comment…test, test, test

Anytime you change your code, there are two things you should do that will make life a lot easier.

Comment your code

First, comment all your changes, even minor ones like changing an attribute value or a localization variable value. Even for these simple changes, it's good to have a record of what you touched and what the original value was.

For more complicated changes, it almost goes without saying that you need to explain what you did and why so anyone who comes across the code can understand why the change was made. A year from now, even you might not remember.

Ideally, start all your comments with something unique, to make it easy to find them later. A good example of a comment is:

```
<!--LWW (20130531): Switched the output order of the figure and its title-->
```

This comment includes the initials of the person making the change, the date the change was made, and an explanation of the change.

Comments can go almost anywhere in your XSLT stylesheets, attribute set files, or localization variables files. The only place you can't put a comment is within another comment.

All comments must start with `<!--` and end with `-->`.

Test your changes frequently

Second, test your changes as you go. Test each change as you make it. Don't delay testing until you've made several changes. If something goes wrong, it's going to be more trouble to find exactly where the mistake happened. Even if the changes seem simple and straightforward, test them. Then, when you encounter a problem, you will know it occurred within the last few changes you made.

From painful experience, I can tell you that even a small error, like forgetting to close an element tag or leaving out a quote character, can set off a cascade of error messages that make it look like there are dozens of mistakes, when in fact there is only one error that messed up everything after it.

Most of the exercises in this book end with, “Save your changes, run a build, and test your work.” Seriously, do that.

2

Custom PDF plugin creation

The whole point of this book is to show you how to create and use your own PDF plugin with the DITA Open Toolkit. This chapter gets you started.

First, I explain what a PDF plugin is and how the one that comes with the DITA Open Toolkit is organized.

Next, I explain how to download and install the DITA Open Toolkit (if you haven't already done that).

Finally, I explain how to create your own PDF plugin and tell the DITA Open Toolkit to use it instead of the default plugin.

> **Important:** You should perform the exercises in this section in the order they're given. Each exercise builds on the previous one, so skipping around is not a good idea.

What is a PDF plugin and why do you need one?

Out of the box, the DITA Open toolkit provides ANT build files, XSLT stylesheets, and Java executables that let you convert a collection of DITA topics into a variety of output formats, including PDF. The processing for each of these output formats comes bundled in set of plugins that you can extend and customize. In this book, we'll explore how to create a plugin that customizes PDF processing.

"Do I need my own PDF plugin," you ask. Unless you think the out-of-the-box PDFs produced by the DITA Open Toolkit are just dandy, then yes—you need a PDF plugin. No question about it.

Now that we've gotten that out of the way, here's the story with PDF plugins.

To customize PDF output, you could just edit the files in the default plugin (`org.dita.pdf2`). That would be fine, until your boss comes to you and says, "We also need to produce a Quick Reference Guide. It should be 5.5 inches by 8.5 inches. And all the fonts need to be smaller. And the margins need to be different. And the headers and footers need to be different…." What do you do? If you're using the default files, you have to keep editing them over and over. Pretty soon, you won't be able

to keep up. And maintaining more than one type of PDF output at the same time will be nearly impossible.

Instead, the right strategy is to create your own separate PDF plugin. That way, you bundle up all of your changes into one nice, neat package that you can easily maintain.

Going back to the example above, it makes a lot more sense to create one PDF plugin for User Guides and another for Quick Reference Guides. That way, you simply call the plugin you need when you create a PDF. It's like having different templates in Adobe FrameMaker or Microsoft Word for different kinds of documents.

When you have a nicely bundled-up PDF plugin, you can easily move it from one installation of the DITA Open Toolkit to another. (Or you can keep it in another location entirely; it doesn't have to be within the DITA OT folder.) You can also send the plugin to other writers on your team, to contractors, to Marketing, or to whomever. It's self-contained and portable.

So now that you're convinced you need a PDF plugin, exactly what is one?

To create PDFs from a map or bookmap, the DITA Open Toolkit uses many different files. These files are in various places in the Open Toolkit, but most of them, and almost all of the ones you'll need to work with, are found in `DITA-OT/plugins/org.dita.pdf2`. A PDF plugin contains only the files you needed to change for your customization. (In this path, and throughout the book, DITA-OT refers to whatever you've named your DITA Open Toolkit folder.)

These files fall mainly into three categories: attribute set files, XSLT stylesheets, and variables files.

- Attribute set files control the way each element looks in the PDF—things like indentation, font characteristics, line spacing, hyphenation, alignment, and so forth. Think (very generally): "attribute set file equals appearance."
- Stylesheets control the way each element is processed. For example, if you want to number a paragraph or assign a custom attribute set to an element based on an @outputclass value, you would use the appropriate stylesheet. Think (again, very generally): "stylesheet equals behavior."
- Variables files do several things, but one of the most important is to dynamically insert language-specific boilerplate text during processing. For example, if you want to add a label in front of a particular element and that label differs depending on the language, you would set that up in a variables file. Variables files also play an important role in setting up headers and footers. Think: "variables file mostly equals translation and headers and footers."

In addition to these files, a plugin can also include its own ANT build file. If you want a plugin that simply overrides attribute sets, XSLT stylesheets, and localization variables, you can create the plugin without an ANT build file. However, if you need additional functionality, you need to include an ANT build file. It is easy to create a plugin with an ANT build file, and doing so will make it easier to add functionality in the future, so we will take that approach.

Organization of the org.dita.pdf2 plugin

Before you get started creating your own plugin, it'll be helpful for you to understand a little about the `org.dita.pdf2` plugin, because your plugin will have almost the same organization.

IBM originally developed DITA and the Open Toolkit to create online output. IBM has its own print solution, but it did not include it with the Open Toolkit components it contributed to open source. After DITA became a public standard, everybody recognized the need to publish PDFs as well.

The `org.dita.pdf2` plugin was originally developed by Idiom and is still sometimes referred to as the "Idiom PDF plugin." However, the plugin is now maintained by a number of different folks. Like the rest of the DITA Open Toolkit, the PDF plugin is open source, meaning that no one really owns it. In this case, "maintains" just means that certain groups make the code changes that are included in the official plugin as it ships with the DITA Open Toolkit.

> **Important:** There are three versions of the DITA Open Toolkit: full, standard, and minimal. All of the references to folder structure in this book assume you've installed the **full** version, which I strongly recommend.

Out-of-the-box, the DITA Open Toolkit is organized like this.

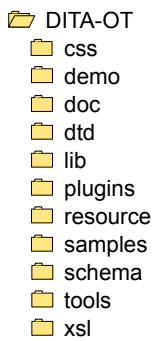

Figure 1: Default OT organization

The `org.dita.pdf2` plugin, not so shockingly, is found in the `plugins` subfolder.

> Actually, that location **is** shocking because it wasn't always that way. In DITA Open Toolkit versions before 1.6, the plugin lived in the `demo` subfolder because the capability was presented more or less as a demonstration of future possibilities. So the plugin was placed in the `demo` subfolder. Finally, it in its logical location.

Plugins folder organization

Here's what the plugins folder looks like out of the box.

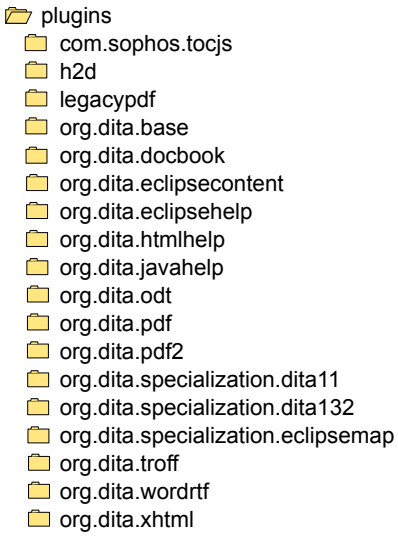

Figure 2: Plugins folder organization

It's kind of scary at first. Block out everything but the `org.dita.pdf2` subfolder, because that's where the PDF plugin lives. Here's what that subfolder looks like inside.

> Why is there also a folder named `org.dita.pdf`? Well, because the PDF plugin has been through a lot of changes in its life. At one point, the developers replaced the original one with a snazzier new one. But because everything in the DITA Open Toolkit is designed to be backwards-compatible, the old one is still hanging around. The two folders are legacies of that history. There's nothing much in the `org.dita.pdf` folder, though, and these days, when you create a PDF, you're using the `org.dita.pdf2` plugin.

`org.dita.pdf2` folder organization

Here's what the `org.dita.pdf2` folder contains out of the box.

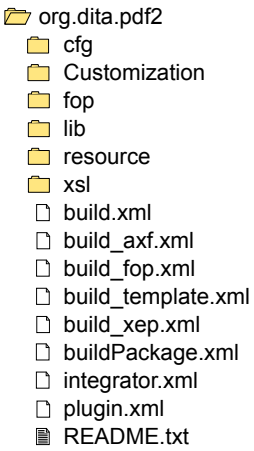

Figure 3: `org.dita.pdf2` organization

You won't need to work with any of those .xml files for a typical PDF plugin, so forget about them. You'll invoke some of them behind the scenes, but you won't directly edit them.

There are these sub folders within the `org.dita.pdf2` folder:

cfg	Contains the sub folders `common` and `fo`. More on these below.
Customization	This folder used to be a "starter set" for your own PDF customization. It became obsolete in version 1.6 of the DITA Open Toolkit
fop	Contains the Apache FOP PDF renderer.
lib	Contains fo.jar, a Java executable needed for PDF processing.
xsl	Contains the XSLT stylesheets that process DITA content into PDFs. You'll learn how to copy these XSLT stylesheets to your own PDF plugin and modify them.

The `cfg` subfolder is where the magic happens, as far as customization is concerned. Well, most of the magic, anyway. The `cfg` subfolder has this structure.

Here's a description of each of the sub folders of `cfg`.

artwork	Contains standard images and icons used by the default PDF plugin. If you have images that are specific to your plugin, such as custom icons or logos, you should store them in the artwork subfolder of your plugin folder.
index	Contains the index sorting files for various languages. These files determine the order in which the DITA Open Toolkit sorts the index for different languages.
properties	Contains the properties files for various languages. By default these properties files specify only the native character encoding for the language, but you can include additional properties in them, including ANT properties—an alternative to including those properties in each individual ANT build file.
vars	The files in this folder contain boilerplate text that can be dynamically inserted in your PDF, avoiding the need for redundant translation. This boilerplate text includes things like the labels for figure and table titles, notes, warnings, cautions, and lots more.
attrs	Contains the attribute set files that determine the appearance of DITA elements in a PDF.
i18n	Contains the character sets for various languages. These character sets specify which subset of the Unicode character set the language uses.
xsl	Empty by default (except for the **custom.xsl** file). You don't really do anything with this folder. But, within your plugin folder, you'll have an xsl subfolder just like this one. You copy any XSLT stylesheets you want to modify for your plugin into this folder.

The `fo` subfolder also contains **font-mappings.xsl** and **layout-masters.xsl**. **Font-mappings.xsl** determines the fonts that appear in your PDF. **Layout-masters.xsl** determines the master pages available in your PDF—body master pages, index master pages, cover pages, and so forth.

If this doesn't make a lot of sense right now, don't worry. As you work through the exercises in this book, you'll understand these folders better.

DITA Open Toolkit 1.5 and earlier

In case you don't know, there was a fairly significant re-architecting of the PDF plugin in the DITA Open Toolkit beginning with version 1.6. For one thing, before DITA Open Toolkit 1.6, it wasn't really a plugin at all. It was a customization that lived in `DITA-OT/demo/fo`. Although the customization folder was organized almost identically to the plugin, the method for creating and calling a PDF customization was different from the method explained in this book.

There were also many files that had a "_1.0" suffix. These files existed to supplement the original files with code specific to new functionality introduced with DITA 1.1. All of those files went away as of version 1.6 of the DITA Open Toolkit. However, if you're working with older versions of the Open Toolkit you need to be aware of them.

While a lot of folks are still working with older versions of the DITA Open Toolkit, and it would have been nice to include instructions for those versions in this book, it was impractical to do so. The good news is that almost everything in this book, except this chapter, is applicable to those older Open Toolkit versions. You might have to tweak things a little, but you should be able to follow the steps and get the results you want.

Exercise: Download and install the DITA Open Toolkit

Objective: Download the DITA Open Toolkit, install it, and run the demonstration build to verify the installation.

The first step in creating a PDF plugin is to make sure you have a working DITA Open Toolkit installation. In case you don't already have one, here are brief instructions for installing and testing it. These instructions assume you have Java installed on your computer. If you don't, you need to install it separately because the DITA Open Toolkit requires Java to run. Installing Java is beyond the scope of this book.

1. Download the latest stable version of the DITA Open Toolkit from **http://dita-ot.sourceforge.net/**

   The design of this web page changes with some frequency, so this topic isn't going to give specific instructions for what to click to get to the latest stable release. Generally, it is well-labeled to help you determine which release you need.

> **Important:** Be sure to download the full-easy-install version, not the standard or minimal versions. The full-easy-install version includes everything you need to run the DITA Open Toolkit except Java.

2. After downloading the zip file, expand it to a folder on your computer.

   A good practice, and the practice that this book assumes, is to expand it to your C: drive on Windows or the Mac/Linux equivalent.

   You'll end up with a folder named something like DITA-OT1.8.

3. Look inside the DITA Open Toolkit folder.
4. On Windows, double-click startcmd.bat. On Mac or Linux, run startcmd.sh in a terminal window.

   On Windows, a command prompt window opens and text is displayed followed by a prompt. On Mac or Linux, text is displayed in the terminal window followed by a prompt.

5. At the prompt, type `ant -f build_demo.xml` and press **Enter**.

   This command runs a test/demonstration build that's included with the DITA Open Toolkit.

6. At the first prompt, press **Enter** to continue.

   This prompt is asking you for the map you want to use. Pressing **Enter** tells the build to use the default map, **hierarchy.map**.

7. At the second prompt, press **Enter** to continue.

   This prompt is asking you for the output folder for the PDF. Pressing **Enter** tells the build to use the default folder, `DITA-OT\out`.

8. At the third prompt, type `pdf` and press **Enter** to continue.

   This prompt is asking you for the kind of output you want to create; you are specifying a PDF.

9. At the fourth prompt, type `y` and press **Enter** to continue.

   A whole bunch of stuff scrolls across the screen. You might see some warnings. Don't worry about them for now.

   When the text stops scrolling, you should see BUILD SUCCESSFUL at the bottom of the screen. This means you have a working version of the DITA Open Toolkit. Look in the `DITA-OT\out` folder to see the PDF you just created (**hierarchy.pdf**).

   If you see BUILD FAILED, then something is wrong. Troubleshooting DITA Open Toolkit installations is beyond the scope of this book, because any number of things could be at issue. For help, refer to the documentation included with the DITA Open Toolkit (`DITA-OT\doc`) or the DITA Users list.

Don't continue until you can build **hierarchy.pdf** successfully!

Exercise: Create your own PDF plugin

Objective: Create a custom PDF plugin that you can use to build PDFs with your own look and feel.

Before you start

Complete the exercise **Download and install the DITA Open Toolkit** *(on page 15)*.

Creating your own plugin makes it easier to move your changes to a new version of the DITA Open Toolkit and also ensures the originals (in org.dita.pdf2) remain intact in case you ever need clean copies or just want to use the default PDF processing.

1. In `DITA-OT/plugins`, create a new folder named `com.company.custpdf`.

   You could name this folder anything you want, depending on what you wanted to name the plugin. For the purposes of this book, I'll use this name.

   > You can download a ready-to-use copy of this folder from **http://xmlpress.net/publications/dita/dita-for-print/plugin** to use as a comparison as you work through these exercises.

   The plugins in the plugins folder use Java-style package naming. You'll notice that most of the plugins in the plugins folder start with "org." The typical naming convention for plugins is to start the folder name with "org" for non-commercial plugins (such as those developed for the DITA Open Toolkit by various non-commercial organizations) or "com" for commercial plugins (such as those developed by companies for their own use). The plugin name normally uses the real Internet domain of the person or organization that owns the plugin. The third part of the plugin typically names the kind of plugin and should be clear enough to enable you to distinguish between different plugins that you develop.

   For example, if you work for BigCorp, Inc., (www.bigcorp.com) all of your plugin folder names would probably start with `com.bigcorp`. The third part of the folder name is what keeps them separate: `com.bigcorp.pdf`, `com.bigcorp.xhtml`, `comp.bigcorp.eclipse`. But keep in mind that you might develop several PDF plugins, for different styles or PDF. In that case, you'd want the third part of the plugin folder names to be more specific: `com.bigcorp.pdf-ug` (for User Guides), `com.bigcorp.pdf-qr` (for Quick Reference Guides), and so forth.

2. In com.company.custpdf, create a subfolder named `cfg`.
3. Within the `cfg` subfolder, create subfolders `common` and `fo`.

4. Within the `common` subfolder, create subfolders `artwork`, `index`, `properties`, and `vars`.
5. Within the `fo` subfolder, create subfolders `attrs`, `i18n`, and `xsl`.

   When you're finished creating the folder structure, it should look like this:

Figure 4: Plugin folder structure

6. Within the `cfg` subfolder, create a file named **catalog.xml**.
7. Add this content to **catalog.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog prefer="system" xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <uri name="cfg:fo/attrs/custom.xsl" uri="fo/attrs/custom.xsl"/>
  <uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"/>
  <uri name="cfg:fo/font-mappings.xml" uri="fo/font-mappings.xml"/>
</catalog>
```

   The **catalog.xml** file in your PDF plugin is like a tourist information booth; it directs the Open Toolkit where to look for attribute set files, XSLT stylesheets, index files, i18n files, and more within your plugin. The three lines in this file tell the Open Toolkit to look first for attribute files, XSLT stylesheets, and font mappings within `org.dita.pdf2` and use those rather than the corresponding ones within `com.company.custpdf`.

8. Within the `fo\attrs` subfolder, create a file named **custom.xsl**.
9. Add this content to **custom.xsl**.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    version="2.0">

</xsl:stylesheet>
```

10. Copy **custom.xsl** to the `fo\xsl` subfolder.

Even though the **custom.xsl** files in the `fo/xsl` and `fo/attrs` subfolders have the same name and structure, they serve different (though similar) purposes. In this book, I'll refer to them as **custom.xsl** (attrs) and **custom.xsl** (xsl) to distinguish them.

11. In the `com.company.custpdf` folder, create a file named **plugin.xml**.
12. Add the following content to the file.

```
<?xml version="1.0" encoding="UTF-8"?>

<plugin id="com.company.custpdf">
    <require plugin="org.dita.pdf2"/>
    <feature extension="dita.conductor.transtype.check" value="custpdf"/>
    <feature extension="dita.transtype.print" value="custpdf"/>
    <feature extension="dita.conductor.target.relative" file="build.xml"/>
</plugin>
```

The plugin id (**com.company.custpdf**) corresponds to the name of the plugin folder. When you create additional plugin folders in the future, be sure to change the plugin id to match.

The value of the *dita.conductor.transtype.check* and *dita.transtype.print* feature extensions (**custpdf**) is the name of the transform type that you need to invoke in your ANT build file to call this particular plugin. Each additional plugin should specify a unique transform type.

Finally, The value of the *dita.conductor.target.relative feature* feature extension (**build.xml**) is the file you're going to use to call the whole plugin. Now you need to create that file.

13. In the `com.company.custpdf` folder, create a file named **build.xml**.
14. Add the following content to the file.

```
<project>
    <import file="build_custpdf_template.xml"/>
</project>
```

Build.xml hooks your plugin into the Open Toolkit. It's a pointer to your custom ANT build file.

15. In the `com.company.custpdf` folder, create a file named **build_custpdf_template.xml**.
16. Add the following content to the file.

```
<project name="com.company.custpdf" default="dita2custpdf">
   <property name="transtype" value="custpdf"/>
   <target name="dita2custpdf"
           xmlns:dita="http://dita-ot.sourceforge.net"
           dita:extension="depends org.dita.dost.platform.InsertDependsAction">
      <property name="customization.dir"
                location="${dita.plugin.com.company.custpdf.dir}/cfg"/>
      <antcall target="dita2pdf2"/>
   </target>
</project>
```

This is your custom ANT build file. The custom ANT build file is a template that the DITA Open Toolkit will use to create a file named **build_custpdf.xml** in your plugin folder each time you run the integrator.xml script. Throughout the DITA Open Toolkit, every time you see xxxxxxxx_build.xml, you'll probably see a corresponding xxxxxxxx_build_template.xml file.

Notice that the project name (in this case "com.company.custpdf") corresponds to the name of the plugin folder. When you create additional plugin folders in the future, be sure to change the project name to match.

This ANT build file specifies *dita2custpdf* as the default target; however, the *dita2custpdf* target depends on the *dita2pdf2* target, so the DITA Open Toolkit will run *dita2pdf2* first and then run *dita2custpdf*. (Target *dita2pdf2* is found in the **build_template.xml** file of the default `org.dita.pdf2` plugin). What this process means is that the Open Toolkit runs the default PDF plugin and overrides aspects of it as appropriate with corresponding targets and properties in your plugin.

17. Copy **layout-masters.xsl** and **font-mappings.xml** from `DITA-OT/plugins/org.dita.pdf2/cfg/fo/` to `DITA-OT/plugins/com.company.custpdf/cfg/fo`.

Reality check

Reality check: Your plugin folder should look exactly like this fully expanded example.

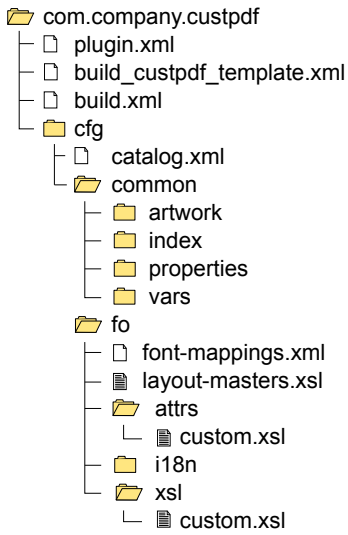

Figure 5: Plugin folder structure - expanded

If it doesn't, go back and review these steps and make changes until it does. If you get sideways here, nothing else will work correctly!

You now have the essential parts of an PDF plugin folder set up. The next step is to integrate your plugin into the DITA Open Toolkit.

Exercise: Integrate your plugin into the DITA-OT

Objective: Make the DITA Open Toolkit recognize your custom PDF plugin.

Before you start

Complete the exercise **Create your own PDF plugin** *(on page 17)*.

You've created a plugin, but now you need to tell the DITA Open Toolkit about your plugin, because the Open Toolkit can't detect it automatically.

1. In the DITA-OT folder, double-click **startcmd.bat** (Windows) or run **startcmd.sh** from the command line (Mac/Linux).

In addition to kicking off builds, the batch or shell file sets up a lot of environment variables on your machine. What are environment variables? There are other applications the DITA Open Toolkit needs to run, and these applications are located in various places on your computer. Environment variables tell the DITA Open Toolkit where everything is and also define some properties needed to run a build.

A command window appears and a lot of text scrolls by quickly. You are left at a command prompt:

```
C:\DITA-OT>_
```

2. At the command prompt, type `ant -f integrator.xml` and press **Enter**.

   The integrator scampers through the plugins subfolder, finds all the installed plugins, detects their transtypes and other information, and marks them present and accounted for to the DITA Open Toolkit. You can (and probably will) run the integrator multiple times; it doesn't hurt anything to re-integrate a plugin.

   You won't see much in the command prompt window. Most likely, just

```
strict:

integrate:
```

   And then (hopefully) BUILD SUCCESSFUL.

   When you see that, your plugin is ready to go.

   If you see BUILD FAILED, then something is wrong in your plugin. Review the previous exercises and verify that you have set everything up exactly right.

   Don't continue until your plugin is successfully integrated into the DITA Open Toolkit.

Exercise: Add an attribute set file to your plugin

Objective: Copy an attribute set from the default PDF plugin to your custom plugin so you can make changes to it without affecting the default attribute sets.

Each time you need to change the appearance of an element in your PDF, you will need to find the attribute set file that controls that element and copy that file to your plugin. You don't need to do this right now, but you'll do it a lot as you add files to your plugin. Here are instructions you can follow each time.

> **Important:** If you include **commons.xsl** in your plugin, then you add an attribute set file to your plugin, you must also copy the corresponding XSLT stylesheet, even if you don't need to change anything in it. If you don't add the XSLT stylesheet, changes to the attribute sets in the included attribute set file are not reflected in your PDF. This seems a little confusing, but there is a reason. To understand why, you need to understand DITA element classes. You can wait and find out about those a little later, or you can jump ahead a little and read about them in **DITA element classes** *(on page ?)*.
>
> The corresponding XSLT stylesheets have the same name as the attribute set files, just without the `-attr` part. You can find them in `DITA-OT/plugins/org.dita.pdf2/xsl/fo/`, and you copy them into `DITA-OT/plugins/com.company.custpdf/cfg/fo/xsl`. Note that the relative path from the plugin to the subfolder is different in the plugin and the original. This is another anomaly that you need to be aware of.

1. After you determine the attribute set file you need to customize, copy it from `DITA-OT/plugins/org.dita.pdf2/cfg/fo/attrs/` to `DITA-OT/plugins/com.company.custpdf/cfg/fo/attrs`.
2. Add an `xsl:import` statement to `DITA-OT/plugins/com.company.custpdf/cfg/fo/attrs/`**custom.xsl**.

   For example, if you copy **commons-attr.xsl**, add the following `xsl:import` statement:

```
<xsl:import href="commons-attr.xsl"/>
```

3. Save **custom.xsl** (attrs).

Exercise: Add an XSLT stylesheet to your plugin

Objective: Copy an XSLT stylesheet from the default PDF plugin to your custom plugin so you can make changes to the processing without affecting the default processing.

Each time you need to change the behavior of an element in your PDF, you will need to find the stylesheet that processes the element and then copy that stylesheet to your plugin. Again, you don't need to do this right now, but you'll do it a lot as you add files to your plugin. Here are the instructions you can follow each time.

1. After you determine which stylesheet you need to customize, copy it from `DITA-OT/plugins/org.dita.pdf2/xsl/fo/` to `DITA-OT/plugins/com.company.custpdf/cfg/fo`.

2. Add an `xsl:import` statement to `DITA-OT/plugins/com.company.custpdf/cfg/fo/`**custom.xsl**.

   For example, if you copy **commons.xsl**, add the following `xsl:import` statement:

```
<xsl:import href="commons.xsl"/>
```

> **Important:** When adding `xsl:import` statements to **custom.xsl** (xsl), be aware that the higher in the list a file is called, the lower its priority. Generally this is not critical, but it is important with respect to **commons.xsl**. Be sure to include **commons.xsl** high in the list—even first. Why is this? The explanation might be a bit much to absorb right now, but if you want to read ahead, see **DITA element classes** *(on page ?)*

3. Save **custom.xsl** (xsl).

Why not use a single custom file for my changes?

You just learned how to copy an attribute set file or a stylesheet to your plugin folder. Copying the files you want to change, including them using an import statement in **custom.xsl** (attrs) or **custom.xsl** (xsl), and making your changes in the copied files is almost always the best way to customize your plugin.

But there is an alternative. Say you want to customize a particular attribute set from **commons-attr.xsl**. Rather than copy **commons-attr.xsl** to your plugin, you can copy just that attribute set to **custom.xsl** (attrs). What this approach says is, "I don't want my own copy of **commons-attr.xsl**. Instead, I'm going to include directly within **custom.xsl** only the attribute sets from **commons-attr.xsl** that I plan to change in my plugin."

For example, here's **custom.xsl** (attrs) with an attribute set copied from **commons-attr.xsl**:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    version="2.0">

    <xsl:attribute-set name="fig.title" use-attribute-sets="base-font common.title">
        <xsl:attribute name="font-weight">bold</xsl:attribute>
        <xsl:attribute name="space-before">5pt</xsl:attribute>
        <xsl:attribute name="space-after">10pt</xsl:attribute>
        <xsl:attribute name="keep-with-previous.within-page">always</xsl:attribute>
    </xsl:attribute-set>

</xsl:stylesheet>
```

Generally, don't do this. Why? On the surface, this might seem easier than copying entire files. And, if you know for a fact you're only going to edit a handful of templates or a handful of attribute

sets, it might make sense to do it this way. But it's pretty rare that a full PDF plugin is limited to so few changes. Odds are, you're going to change dozens of attribute sets and templates.

Eventually, it's going to be hard to keep track of your templates and attribute sets. When you want to make a change, you'll have to remember whether you already copied that template/attribute set to **custom.xsl** or not. For a given set of elements, some of the processing and formatting might take place in the default XSLT stylesheets and some via your custom template/attribute sets in **custom.xsl**. That can become confusing when you need to track errors or find where to make a specific edit.

With one notable exception, it's better to copy the entire stylesheet or attribute set file to your plugin folder. That way, you know for sure where the processing is being done and where you need to make your edits or do your troubleshooting.

What's the exception? If you create a template from scratch or use one that some generous person has given you, it makes perfect sense to add the new template to **custom.xsl** (xsl) unless it fits neatly into one of the existing XSLT stylesheets. The same is theoretically true of attribute sets, but most are closely related to an existing attribute set, so it usually makes more sense to add them near their kinfolk, in the appropriate attribute set file.

That said, Patrick Quinlan has developed a **custom.xsl** file with some of the most common edits to title pages, headers and footers, notes, and so forth. In turn, he gives a shout-out to Jarno Elovirta, who has an excellent online stylesheet generator you can use to generate your own **custom.xsl** file. You can drop either of these files into your plugin. For a small number of customizations, these tools are great. If you plan to do a serious overhaul of the default PDF plugin, you'll probably be better off using them as starting points or learning aids. For links to these tools, refer to **Useful resources** *(on page 3)*.

Exercise: Add a localization variable file to your plugin

Objective: Copy a localization variables file from the default PDF plugin to your custom plugin so you can make changes to variables without affecting the default values.

Localization variable files are named using ISO language codes—for example, **en.xml**. These names usually reflect the name of the language in that language. For example, **de.xml** is the localization variables file for German, because the name for the German language in German is *Deutsch*.

- Copy the appropriate localization variables files from `DITA-OT/plugins/org.dita.pdf2/cfg/common/vars/` to `DITA-OT/plugins/com.company.custpdf/cfg/common/vars/`.

You should copy the files for all the languages you plan to publish in, even if you don't make any changes to them right away. Odds are you will, and when you do, they will be nestled snugly in your plugin folder, waiting for you.

Reality check

That's all you need to do. Your plugin automatically will use your localization variables files instead of the default ones. You don't need to do anything to the catalog.

Wrap-up

If you worked your way through the preceding pages, you now have your very own plugin folder, `com.company.custpdf`. You also understand how to add more files to your plugin for all the formatting and processing changes you'll want to make. And you have just enough understanding of how the plugin is organized to be dangerous … er, to sound knowledgeable.

As you work through this book, you'll need to test your changes. One easy way to do that is to set up simple ANT build files to create a PDF. We'll look at that in the next chapter.