



Extending DITA Open Toolkit: How crazy can you get?

Robert D Anderson, IBM

One of the original DITA Open Toolkit crew

Co-Editor, DITA 1.1 / DITA 1.2 / DITA 1.3

Doubling my tweet count today from @robander

November 20, 2014

Overview

- Background
 - Who are we, why are we here?
 - A note on terminology

- A light touch: simple extensions can take you far
 - CSS and related parameters
 - Pitfalls of short-term overrides

- Plug-ins:
 - What and why?
 - How?
 - I was promised some crazy?

- Resources

Who am I?

- One of the original DITA Open Toolkit developers
- Authoring tools developer with IBM for 15+ years
- Working on DITA support since 2001
- Co-editor of the DITA 1.1, DITA 1.2, and DITA 1.3 specifications
- Somewhat obsessed with music
 - Enough so that I maintain the Music of DITA Project
 - For details on that ... see resources or talk to me after the presentation



Who's in the audience...?

- DITA Open Toolkit users:
 - Command line?
 - Through other tool, such as oXygen?

- Already customizing the toolkit?
 - Directly editing code?
 - Using plug-ins (yours or provided by others)?

- Toolkit version level?
 - Anybody using 1.5.3 or earlier?
 - Anybody trying out 2.0?



What are we here to learn?

- Various ways to extend DITA-OT
- What can be extended
- At least a few technical details
- How to get started with plug-ins



Terminology

- What is “programming”?
 - Using a command line?
 - Creating an Ant build?
 - Writing CSS?
 - Writing XSLT?
 - 01000100 01001001
01010100 01000001?



- Methods described in this presentation require varying levels of comfort with “programming”

Fair warning: this is a technical presentation

- A presentation about how to use XML control files to insert XSLT routines into a complex build process can be a little ZZZZZ.....



- Please interrupt at any point with questions. It will wake everybody else up!

Customization with a light touch

- Processing parameters
- CSS
- A little heavier:
 - /XSL parameters
 - Customization directories
- Editing DITA-OT code
 - Danger ... danger ... danger ...



Processing parameters



- All builds have parameters
- Some apply everywhere, some specific to one format
- See the user guide for a full list of external parameters

- Examples:
 - Draft parameter to render draft-comment and required-cleanup
 - Filter parameter to flag or exclude content
 - Header/footer: add common content to XHTML
 - Task labels: include or exclude headings for task sections
 - CSS: add your own styles

CSS: giving your content some style



- Virtually every DITA element adds an XHTML class. For example:
 - `<section>` results in `class="section"`
 - `<prereq>` (specialized from section) gives `class="section prereq"`
- Most of these are ignored by default DITA-OT CSS, but can be used to style your output
 - A few lines of CSS can set your font, heading, and text style
 - If you prefer, use pages and pages of CSS to customize every element

XSLT ... a quick dive into programming

- Most toolkit builds do final rendering with XSLT
- Parameters allow stylesheets to override default processing
 - XHTML: args.xsl runs the specified XSLT file in place of the normal conversion; typically imports core code, then overrides selectively
 - Similar parameter available for PDF
- May be useful for one-off overrides, or those that apply to only a few situations
- XSL overrides may be difficult to keep track of, easy to mix up



PDF Customization directories...

- ...are available but will not be covered here.
- Customization directories are unique to PDF (PDF2), and do not follow designs used elsewhere in DITA-OT.
- Long term, plug-ins are preferred, but we realize customization directories aren't going away soon.
- *If you can wait another hour or two, Leigh White will give you actual useful information about PDF.*

Editing DITA-OT code

- It's open source. So I should just edit source, right?
- Comes with many pitfalls:
 - Upgrades are more difficult
 - Hard to take advantage of new fixes or features
 - May duplicate work others are already doing
 - May not be possible to integrate with future releases
 - May not be possible to share with other writers
- I've encountered several companies unable to upgrade versions because of this approach



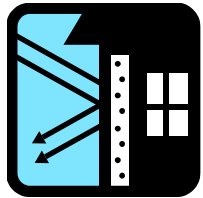
Plug-ins to the rescue

- Plugging in to the Toolkit – the best way to:
 - Brand your output
 - Modify processing
 - Share your specialization
 - Create new output formats

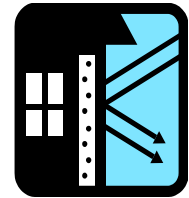


Why use a plug-in?

- Easier to share customizations
- Easier to upgrade releases
- “Just works” with many vendor products



- Insulates your extension from many toolkit changes, operating system changes, vendor changes, ... makes the extension as independent as possible



What *is* a plug-in?

- Typically distributed as a zip file, with one or more directories
- Each plug-in directory has a control file `plugin.xml`



- A plug-in can be just that control file
- Can also be 5 lines of XSLT, or 50 subdirectories with 5,000 lines of Java, or more

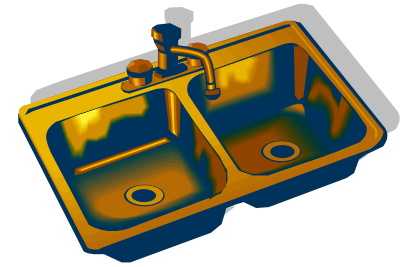
How do I install a plug-in?

- New to DITA-OT 2.0:
 `dita -install plug-in-zip`
or
 `ant -f integrator.xml install -Dplugin.file=plug-in-zip`

- Older releases, typical approach:
 - Unzip to DITA-OT\plugins\ directory
 - Run the Ant command:
 `ant -f integrator.xml`

- Advanced: use alternate install locations, automate plug-in integration as part of your build

What can I *do* with a plug-in?



- Override XSLT processing, define new XSLT parameters
- Extend the DTD/XSD catalog for new specializations or custom document type shells
- Define new Ant targets, or extend existing ones
- Add new transform types
- Define new or change existing generated text
- Add new Java libraries to the toolkit's classpath
- Define new error messages
- Define a pre-requisite plug-in
- Declare new extension points (*the possibilities are endless!*)

Before we go on...

- Samples of all extensions described here are available in zip form at <http://metadita.org/toolkit/>
 - Source versions also available at <https://github.com/robander/metadita>
- *No need to scribble down syntax during the presentation; see resources slide for links*

XSLT processing sample

- XSLT modifications may be used to change the way an element is rendered, or to create a custom brand image
- The two files below are a complete plug-in that adds a (theoretical) company logo to every XHTML page:

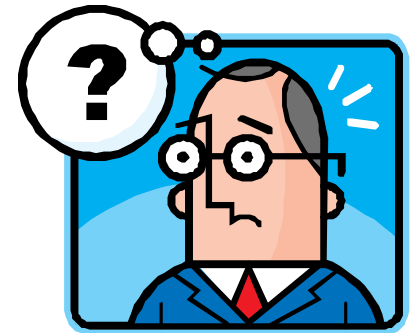
```
<?xml version="1.0" encoding="UTF-8"?>
<plugin id="org.metadita.brandheader">
  <feature extension="dita.xsl.xhtml" file="xsl/header.xsl"/>
</plugin>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template name="gen-user-header">
    <div></div>
  </xsl:template>
</xsl:stylesheet>
```

OK, what happened there?

- Start with the control file plugin.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin id="org.metadita.brandheader">
  <feature extension="dita.xsl.xhtml" file="xsl/header.xsl"/>
</plugin>
```



- This sample plugin.xml file contains three important features:
 - The @id attribute for the plug-in defines the plug-in and must be unique
 - The <feature> element defines the extension:
 - The extension attribute says what to extend. In this case, dita.xsl.xhtml corresponds to the XHTML output processor.
 - The file attribute says what file extends the processing; that file is imported into normal XHTML processing code (everybody who uses that, gets it)
- *Note: this does not count as a crazy override*

XSLT: finding the right spot to override...

- ...can be difficult. See backup slide for tips.
- Extension points are generally available for each XSLT processing step and each output format



XSLT parameters: why?

- Typically works together with XSLT or Ant extensions
- Simple example:
 - Define a new parameter called “DAY” to be passed in to the XHTML rendition step, with a value of the current day
 - An XSLT override is also provided to make use of this new parameter; the override may do anything with it, such as:
 - If DAY='WEDNESDAY', display everything in reverse
 - Alternatively, print out the current date at the bottom of each page



XSLT parameters, continued

- Complex, more realistic example:
 - A plug-in defines a new parameter called “MYTARGET”
 - This parameter is passed in to XSLT, which makes style tweaks based on the target platform
 - Users may set the value in their Ant build
 - Setting MYTARGET='ECLIPSE' may remove next/previous links
 - Setting MYTARGET='MyWidget' may add special styles to every paragraph
 - Setting MYTARGET='Review' may add revision markers or dates
- *Note: this could also be done by creating a transform type of “Review” and so on.*

New DTD or XSD entries in the catalog

- The ability to create new document types or specializations is a core part of the DITA Standard
- In order to process the grammar, DITA-OT must find it; this means adding references to your DTD / XSD into the catalog
- The extension point `dita.specialization.catalog.relative` references a file with the relevant XML Catalog extensions:

```
<feature extension="dita.specialization.catalog.relative"  
          file="my-local-catalog.xml"/>
```



Adding Ant targets

- “Ant target” = new build step
- DITA-OT declares extension points to insert new steps before many common steps
- For example:
 - Run my new step “callThisBeforeConref” before conref:

```
<feature extension="depend.preprocess.conref.pre"  
value="callThisBeforeConref"/>
```
 - Run my new step “ReadyForAnything” after the general preprocess:

```
<feature extension="depend.preprocess.post"  
value="ReadyForAnything"/>
```
- Once again, the new step can be simple or complex




Creating a new transform type

- Minimum requirements:
 - Declare the transform type (“pdf2”, “faketext”, “my-html”)
 - Define some Ant code for the new transform. To start, this could even be a redirect to existing code (“my-html” calls “xhtml”)
- More realistic minimum:
 - Ant code calls preprocess, then some new code or extensions
- Sample extension defines transform type “my-html”, causes Ant code to look for target “dita2my-html”

```
<feature extension="dita.conductor.transtype.check" value="my-html"/>
```

New transform types: samples?

- 
- Some transforms in DITA-OT started as community plug-ins:
 - TocJS: tweaks XHTML code, adds steps for JS navigation, includes framesets for tweaking to create your style
 - PDF2: Primary PDF plug-in in DITA-OT; includes a large amount of XSLT / XSL-FO processing code, Java steps, resource files, and more

 - Popular external transform types:
 - EPUB in DITA4Publishers – large set of connected plug-ins with transform types, specializations, and more
 - HTML+ – another large set of connected plug-ins with support for SVG syntax diagrams and other features
 - Many of the WebHelp options that ship with vendor tools



New generated text

- Change what is generated
 - The default “Note:” that appears for <note> becomes “Hey, look at this!”
- Add new text for use with new XSLT customizations
 - In every task, generate “Here are some steps”
 - In every footer, generate “Help contribute to DITA-OT development”
- Text is generated based on @xml:lang in the DITA content



Adding Java libraries

- Java code added to the classpath using this feature becomes available to Ant or XSLT processing:

```
<plugin id="org.metadita.jarsample">  
  <feature extension="dita.conductor.lib.import"  
    file="myJavaCode.jar" />  
</plugin>
```

- Only limitation is your skill with Java



New error or diagnostic messages

- XML file provides messages using a simple syntax with message ID, warning level, message, and optional response
- Can be used for anything. For example:
 - Ant step emits diagnostic message if property is not set
 - XSLT step warns about missing <shortdesc>



Setting up dependencies

- Several reasons to require another plug-in:
 - XSL transform depends on the presence of string files
 - Specialization depends on XSLT processing, or on another specialization (Java API Reference depends on API Reference)
 - New transform depends on existing transform type
 - New Ant code depends on externally provided transforms

- Plug-ins can declare dependencies
 - *I will not work unless you find plug-in "org.metadita.prereq"*

- *Personal note: I've seen this get ... complicated.*



Creating your own extensions

- Plug-in integration is based on `_template` files
- Each file that can be extended has a corresponding `_template` version, which is used to regenerate the main file
 - `dita2xhtml.xsl` is created when the integrator processes `dita2xhtml_template.xsl`
 - `build.xml` is created when the integrator processes `build_template.xml`
- Plug-ins can define a new `_template` file; during integration, this process will generate a new copy after searching for integration points:

```
<template file="myTemplateFile_template.xsl"/>
```





Now that everything is clear...



Resources

- DITA-OT Plug-in documentation:
 - http://dita-ot.github.io/2.0/dev_ref/plugins-overview.html

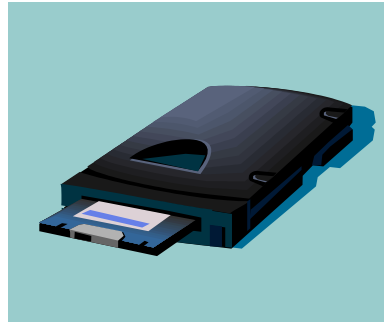
- Sample plug-ins (and a matrix!):
 - <http://metadita.org/toolkit/>

- Source repository for the samples:
 - <https://github.com/robander/metadita/>

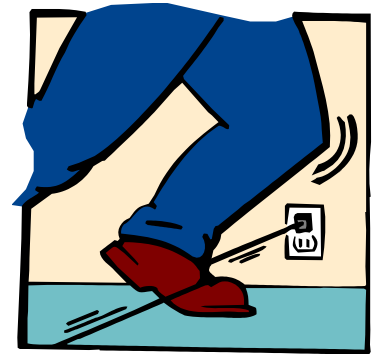
- Everything you need for PDF overrides:
 - *DITA for Print* by Leigh White – or wait for her talk!

- Music of DITA Project:
 - <http://metadita.org/music/>

Backup slides: tips and tricks, if we have time



Tips to avoid getting tripped up



- How I approach nearly any plug-in:
 - Start with a sample that extends what I want to extend
 - Change the plug-in ID, directory name, etc
 - Remove whatever it customized that you don't need
 - Integrate, run, tweak, repeat

- Example: to override XSLT for XHTML
 - Get the plug-in “org.metadita.xsl.xhtmlbrand” from samples page
 - Change directory name, edit @id in plugin.xml, maybe rename XSL
 - Delete templates “gen-user-header”, “generateDefaultCopyright”
 - Integrate, run, add a little code, integrate, run, add...

Tips for overriding XSLT

- Start with a plug-in directory, define your extension point in plugin.xml, and create an empty XSLT file:

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- Content will go here -->
</xsl:stylesheet>
```

- Integrate the plug-in – it does not do anything, but your XSLT is now part of the build
- Open the XSL file you wish to override
- Locate the template you need to modify – either a common named template as in the earlier example, or a specific element rule
- Copy the template into your no-longer-empty XSLT file and save
- Modify the template as needed; you now have a working plug-in
- Suggestion: run a simple build at each step to make sure it still runs...

Tip for plug-ins working with plug-ins

- When a plug-in is integrated, default properties are generated
 - Plug-in with ID “org.metadita.sample” generates Ant property set to the install directory:
`${dita.plugin.org.metadita.sample.dir}`
- When referencing other plug-ins from Ant, use this property to avoid dependencies on install locations or directory names.
- *This feature was added in DITA-OT 1.5.4*

Disclaimer

© Copyright IBM Corporation 2014. All rights reserved.

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

IBM customers are responsible for ensuring their own compliance with legal requirements. It is the customer's sole responsibility to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer is in compliance with any law.

IBM, the IBM logo, FileNet, Lotus, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.

Other company, product, or service names may be trademarks or service marks of others.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.