DobuDish User Manual

Version 1.5

Revision History

22.11.2006	1.0	Torsten Uhlmann	Initial Revision
31.12.2006	1.1	Torsten Uhlmann	Added new output formats. More description
05.01.2007	1.2	Torsten Uhlmann	Changed property substitution mechanism.
10.07.2007	1.3	Torsten Uhlmann	Added a table of available build target hooks.
13.02.2008	1.4	Torsten Uhlmann	Description of new Website feature.
01.08.2008	1.5	Torsten Uhlmann	Added documentation for the defined Ant build script hooks.

Overview

This manual describes the usage of *DobuDish* from a user's point of view.

Table of Contents

1. Introduction

DobuDish is a flexible Docbook publishing tool chain. It is build upon the Java® programming language. That means you can

use it on all systems that are supported by Java®, for instance Windows®, MacOS® X and Linux™.

DobuDish itself is just a script that glues together some of the best freely available tools that are needed to process Docbook. These are for instance (in no particular order):

- Apache FOP: used to generate PDF output
- Saxon XSL Processor: Saxon is used to process the Docbook XML file and transform it into the many other output formats.
- Apache FOP: used to generate PDF output

2. Prerequisites

DobuDish has been extensively tested on Windows® XP. It should also work without problems on Windows® 2000. Others run *DobuDish* successfully on Mac OS X or Linux systems such as Ubuntu.

Please report any success or failure of running *DobuDish* on different Windows® or Unix™ versions.

DobuDish comes bundled with the Java™ JRE Version 5 for Windows® If you use it on Unix™ you need to supply your own Java™ version. Having Java™ 5 or above should be ok.

3. Features

The *DobuDish* framework supports the creation of the following document types:

- **html**: creates chunked html output
- **singlehtml**: creates one big single html file
- **xhtml:** create a single page html file
- **singlexhtml:** create a single page html file
- **manpages:** create a single page html file
- **wordml:** create a WordML file
- **htmlhelp:** create a Windows Help (CHM) file
- **eclipse:** create an Eclipse Help plugin (together with toc.xml and plugin.xml)
- **javahelp**: formats the output in JavaHelp format. Double clicking on the JAR file opens the JavaHelp browser which is included in the distribution.
- **tablehtml**(Slides): Create a Slides document with tabular html
- **framehtml**(Slides): Create a Slides html document with a frameset
- **tablexhtml**(Slides): Create a Slides document with tabular xhtml
- **framexhtml**(Slides): Create a Slides xhtml document with a frameset
- **pdf**: Create PDF files suitable for printing.

- *rtf*: Create RTF files[1]
- **ws-tab**: Create a tabular website.
- **website**: Create a non tabular website.
- **validate:** Validate the input docbook file. This is not really a document type but give's you a way to check if your documents are valid.
- **distribute:** export the Docbook input files into a ZIP archive

In addition to these, *DobuDish* features:

- Catalogs are resolved: Docbook documents can keep the official Docbook DTD URL as System ID. These are mapped to locally stored DTD files contained in the *DobuDish* installation.
- Support for XInclude: Documents can be modularized using the XInclude mechanism. Prior to transforming the document into its target format, a special stylesheet creates a docbook file that contains all modules. This files can be checked for validity.
- Additional font support: Additional fonts are available for documents. For instance, the customization layer sets „Georgia“ as the default title font.
- Included JRE: For your convinience the install archive contains the Java® JRE that is used on Windows®. That we you will not run into situations that result from old or unsupported JRE's.
- Flexible customization layer: the customization layer supports global modifications (available to all documents) and local modifications (only for selected documents)
- Adaptable build process: You may need to run extra processes that produce parts of your Docbook document. Maybe you need to parse some plain text files, extract contents and include this into your documents- you can plug these steps into the document build process. This way everything is generated using one command.
- Property resolution: You can define properties through key / value pairs in the global build.properties or in the document local localbuild.properties file. After enabling property resolution (disabled by default) in your docbook document you can write something like `$ant.get("document.version")` and it will be replaced with the corresponding value from the properties files.

4. Installation

Installation of *DobuDish* is dead simple:

From our website[2] (the freeware section of our download area[3]) you can download the latest version. Please save it to a convinient directory of your absolutely personal choice (*Programm Files* for instance). Unzip the archiv with an Unzip utility (also of your choice) into this directory. Windows XP includes an unzip utility. If you need an unzip tool just google for „7-Zip“, „PowerArchiver“ or any other unzipper.

You're done. Finish. No next step. Rejoice.

[1] To use this file type you need to download *xfc* from XMLMind's website [http://www.xmlmind.com/xmleditor/addons.shtml]. You are allowed to use XFC for personal use, therefore it's not included in *DobuDish* by default. Create a directory *xfc* inside the *DobuDish system* directory and copy the following jar files into it: *jaxp.jar, regexp.jar, sax.jar, xfc.jar and xp.jar* . After that you should be able to produce RTF documents.
[2] http://www.agynamix.de
[3] http://cms.agynamix.de/downloads/index.php

To test your installation, do the following:

Open up a cmd shell (DOS command window), change the directory into the *DobuDish* installation directory and issue the following commands:

```
generator.bat fluff create book
generator.bat fluff pdf
```

The first command creates the *documents* directory and the directory *fluff* inside of it. *fluff* is actually the name of the book you just created. Inside the *fluff* directory you will find a file *fluff.xml* which is a docbook template meant to get you started quickly. (I hate starting with a blank page. So I always copy some skeleton and modify this. That takes the pain of having to start with nothing. If you're like me use this template. Of course you can always throw the template away and start the book on your own. Although it would hurt my feelings :)

The second command transforms the provided book template into PDF.

The resulting PDF file can be found in *documents/fluff/output/pdf*.

5. Working with *DobuDish*

5.1. Available File Formats

For the technical minded, *DobuDish* is driven by Apache Ant under the hood. In an Ant script (which actually is much like a Makefile you might know from the C programming world) you have a set of *targets*. Each *target* executes certain steps to fullfill its duty. The pdf target for instance creates first the output folder, it will then merge all XIncluded document fragments into one big file and calls the Docbook stylesheets to process this file into a Formatting Object (.fo) file. That again is given to *Apache FOP*, which creates beautiful PDF from it. I agree, the resulting PDF is not yet as beautiful as LateX would create it, but we're working on it...

So to let *DobuDish* do some work you have to call the *generator* script. Depending on your operation system you call *generator.bat* (Windows®) or *generator.sh* (Unix™). The syntax of this command is the following:

```
generator.bat <document-name> <file-type>
```

```
for instance
```

```
generator.bat fluff pdf  [or]
generator.sh  fluff pdf
```

To issue any *DobuDish* command you have to be in the *DobuDish* root directory, for instance *C:\Program Files\dobudish*. Open a command shell (on Windows® click the „start“ button, click „Execute“ and type *cmd*, press „Enter“. This will open a command window. There type *cd C:\Program Files\dobudish* (substitute *Program Files* with the install directory you've chosen. You also might need to change the drive from *C:* to *D:* for instance. To do this type *D:* in the command window.

Please see Section 3, “Features” [2] for the available document formats or type `generator.bat help` to display a list of available targets.

5.2. Including Media objects

If you have a directory *resource* right in the *input* directory of your document, then this document with all its content is copied

over to the output directory. This is useful if you want to include images into your documents- put them into the *resource* directory and the build process will automatically take care of them.

You reference included images like this:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="resource/cover.jpg" format="JPG"/>
  </imageobject>
</mediaobject>
```

5.3. Special project directories

There are some directories in a project directory that are handled specially:

- *custom-cfg/common-files*: If this directory is present then it's content is preferred over the global common-files directories (there's one for article/book/set, slides or website).
- *input/resource*: If this directory is present then it is copied entirely to *output/resource*. This is the old style. It is preferred to use the next directory.
- *input/copy_to_output*: If this directory is present then **it's content** is copied entirely and recursively to *output*. This is the preferred way to copy data to the output directory that does not need be processed by stylesheets but should be part of the output.

5.4. Multiple documents in one document directory

The assumption *DobuDish* is making is that the directory (or project) you create with `generator.bat fluff create book` is also the name of the document, *fluff.xml*.

Now there is an extension to this assumption. Maybe you have a directory containing XInclude'd document fragments that you use in different master documents. Or maybe you are writing a revision history document along with your master source. Let's assume the revision document is called *revhistory.xml* (the *.xml* at the end is still mandatory) and let's assume, this document is part of the *fluff* project, you can now generate this document using the following command:

```
generator.bat fluff pdf revhistory
```

This will tell *DobuDish* to look at the *fluff* project (directory), generate a *pdf* document, but won't take *fluff.xml* as the input document but rather *revhistory.xml*. The resulting document will be called... ...tam tam... ...*revhistory.pdf*.

The usual way works as ever- if you omit the last parameter, then *DobuDish* will assume the source document is of the same name as the document directory (plus .xml).

One drawback though is that the output directory will be cleared each time you generate one of the documents of one directory. So you need to move a previously generated document somewhere else if you want to keep it.

I'm open for suggestions...

5.5. Creating Docbook Slides

Since version 0.9.9 of *DobuDish* you can use the **Docbook Slides** document type. You can create the following output formats:

- pdf
- html
- tablehtml
- framehtml
- xhtml
- tablexhtml
- framexhtml

To create a Slides document, create a template like this:

```
generator.bat fluffyslides create slides
```

Then, you create output as usual:

```
generator.bat fluffyslides tablehtml
```

5.6. Creating Docbook Websites

DobuDish latest addition since version 0.9.14 is the ability to create **Docbook Website** documents. Tabular chunked as well as non tabular chunked output formats are supported.

To demonstrate the use of Website documents the example which came with the website xsl distribution was incorporated as a *DobuDish* template.

To create a Website project you would type:

```
generator.bat fluffyweb create website
```

This will copy the website example as your new project. Please note that a file "fluffyweb.xml" is created. This is only needed for the generator to determine the type of the project. Other than that it is just a dummy file.

The head file for a website project is the file "layout.xml". This file is used to create "autolayout.xml" which in turn is fed to the tabular or non tabular website driver stylesheets.

To create a tabular website, type:

```
generator.bat fluffyweb ws-tab
```

To create a non tabular website, type:

```
generator.bat fluffyweb website
```

5.7. Validating your document

You might want to check if your document conforms to the Docbook DTD. You can do this with the *validate* command:

```
generator.bat <document-name> validate
```

Any validation errors that the parser finds are reported in the console windows where you issued the command.

5.8. Substituting Properties

Often you have several values that you don't want to write scattered throughout a document. You rather want to define a value in one place and reference it whereever necessary.

One way to do this is to use entities. That's fine as long as you need the value only within a document and don't want to set it from outside.

DobuDish has enhanced that mechanism. You can now set an entity's value by referencing any property variable that is known to the build process (every variable defined in the global build.properties, the document local localbuild.properties or in the build process directly.

To enable property substitution for a document or globally, set the variable `substitute.properties=yes`. If you do this in the custom-cfg directory of your document, you will enable property substitution only for the documents in this directory. If you want to enable property substitution globally you need to set it in the global property file "build.properties"

After you have enabled property substitution you can write something like this:

```
<!ENTITY docversion      "$ant.get("document.version") ">
```

The property ist retrieved by the term **$ant.get("PropertyName")** where *PropertyName* is the name of a property from the Properties files or from inside Ant.

Later in the document, you can reference it via &docversion; which will result in 1.5.

DobuDish uses VPP (Velocity PreProcessor) as substitution engine. So if you'd like to script your Docbook document a bit you can use all the mechanisms that are supported by Velocity. For a reference please see http://jakarta.apache.org/velocity/.

5.9. Distributing your document source

There are cases where you might need to send your document sources to someone. *DobuDish* provides a target called *distribute* used to zip all pieces of one document into one ZIP archive. The archive name contains the version of the document which can be set modifying *document.version* in *localbuild.properties*. To create an archive of your document use the following command:

```
generator.bat <document-name> distribute
```

This will build a file *<document-name>-<document.version>.zip* stored in your documents root directory (*documents/<document-name>*.

6. Modifiying the Customization Layer

The cutomization layer is flexible enough to function with no modifications at all, modifications on an installation wide level

and/or modifications on a per document basis.

To modify the customization layer globally, please look into the *custom-xsl* directory within the *DobuDish system* directory. The *manual* directory contains valuable information about how you can customize the Docbook layout.

To modify the customization layer on a per document basis you need to adapt the stylesheets in the *custom-cfg* directory of your document. You modify those stylesheets the same way you would do with the global ones, they are just imported into the build process before the global ones, so if you define something there, it essentially overwrites the global settings.

7. Extending the build process

Now, this is for geeks only :)

You are a geek, right?

To extend the build process you follow these steps. First look into *localbuild.xml* in the *custom-cfg* directory of your document. It is an Ant file with some empty targets included. The targets there follow the simple naming scheme *local-pre-<real target>>* and *local-post-<real target>*. You need to check where you want to plug your own extensions into. If you need to generate some sources, then the *local-post-init* target might be right for you. Please consult the Ant website[4] for any help.

8. Variables of the Build Process

Build time properties are stored in the file *localbuild.properties* in the *custom-cfg* directory of your document. Here is a snapshot of that file for the *DobuDish* manual (sourced into the manual via XInclude):

```
# BEGIN Do not overwrite
document.type = book
# END Do not overwrite

# These are properties that need adaption for each individual document

# These are needed for the 'eclipse' target

# The Eclipse plugin ID for the manual plugin
eclipse.plugin.id=com.agynamix.dobudish

# A Name for your plugin
eclipse.plugin.name=DobuDish User Manual

# The company that provides the plugin
eclipse.plugin.provider=AGYNAMIX

# The version of the eclipse plugin
eclipse.plugin.version=${document.version}

# The version of the document
document.version = 1.5

# Substitute property names in docbook documents with its
# corresponding values
substitute.properties=yes

docbook-dtd.version = 4.5
```

[4] http://ant.apache.org

Global properties are found in a file called *build.properties* in the root directory of *DobuDish*. It contains the following entries. If an entry occurs in both the global and the document local property file then the local value overwrites the global one. Of course.

```
# The version of DobuDish
version = 1.1.1

# overwrite in localbuild.properties
document.version = 1.0

# The Docbook DTD version to use
docbook-dtd.version = 4.5

# Are you behind a Firewall and need to use a proxy?
http.proxyHost=
http.proxyPort=

# Do you want to enable variable substitution in your Docbook files?
# After generating the complete .docbook file, if this is set to 'yes'
# the file is scanned for variables ('{{{variable.name}}}') and if found,
# replaced with values found in your localbuild.properties or build.properties.
# You can addjust the variable separators below
substitute.properties=no
#substitute.properties=yes

# How much memory would you like to give saxon?
xslt.maxmemory.default = 256m

# How much memory would you like to give Apache FOP?
fop.maxmemory.default  = 256m

# The PDF task might need a bit more
fop.pdf.maxmemory      = 512m

# A timestamp is generated when the convertion process starts
# The property 'TODAY' will hold the current date formatted as specified here.
# Please see the documentation of SimpleDateFormat for possible patterns.
tstamp.format.pattern  = yyyy-MM-dd
tstamp.format.locale   = en,EN

# The name of the folder whose content will be carbon copied to the output directory
special.folder.name.carbonCopy=copy_to_output
```

9. Using Build Hooks to customize the document generation

The build process is based on Apache Ant (a kind of *make* for Java). With Ant you have *targets* which you call in order for Ant to perform several steps that would lead to the result of this target. *PDF* for instance, is an Ant target. So when you call `generator.bat fluff pdf` it will internally call a target called *pdf*.

To give you some flexibility with the build process I have incorporated some hooks into the chain of executed commands. A hook is basically an Ant target. If it's there it will be executed. If it is missing it's simply skipped. In your document's folder there is a folder called *custom-cfg*, within that you find a file *localbuild.xml* which is the place to add local hooks. Now if there are local hooks then there are also global ones (we want to be consistent, right?). Global hooks need to be added to *system/*

etc/build-user-augments.xml.

When hooks are executed the script will first search for a local hook. If the local hook is there it will be executed. If it can't be found the systems looks for a defined global hook. Again if it's found it will be executed, otherwise the build process simply proceeds. If you want to call global **and** local hooks for one specific hook you have to insert a call to the global hook into the local one like so: `<antcall target="local-pre-init" />`

Following is a table showing you all the available hook targets together with a small explanation when they're called. If you're curious you can look into the build file *system/etc/build-output-formats.xml* where almost all of them are referenced.

I will omit the prefix "global" or "local" except for those occasions where there is only one of the two available.

Hook Target	**Descripton**
global-pre-create	before creation of a new document starts
-post-cp-custom	after copying the template's cfg directory into the documents directory
-post-cp-template	after copying relevant template files except the cfg directory and the template doc-book files
-post-cp-example	after copying the template docbook files into the documents directory (not called for *website* documents)
-post-create	after the whole document creation process is done
-pre-init	before initialization starts
-post-init	after initialization finished
-pre-resolve-xinclude	before the *resolve-xinclude* target. This target is called before the actual targets that generate things in order to slurp all document snippets together into one.
-post-resolve-xinclude	after the *resolve-xinclude* target. This target is called before the actual targets that generate things in order to slurp all document snippets together into one.
-pre-singlehtml	before the *singlehtml* target
-pre-cleanup-singlehtml	before the *cleanup* target for *singlehtml*
-post-singlehtml	after the *singlehtml* target
-pre-singlexhtml	before the *singlexhtml* target
-pre-cleanup-singlexhtml	before the *cleanup* target for *singlexhtml*
-post-singlexhtml	after the *singlexhtml* target
-pre-javahelp	before the *javahelp* target
-pre-jar-javahelp	before the *jar* target for *javahelp*. Called before the generated files ar packaged as JAR.
-pre-cleanup-javahelp	before the *cleanup* target for *javahelp*
-post-javahelp	after the *javahelp* target
-pre-eclipse	before the *eclipse* target
-pre-jar-eclipse	before the *jar* target for *eclipse*. Called before the generated files ar packaged as JAR.
-pre-cleanup-eclipse	before the *cleanup* target for *eclipse*
-post-eclipse	after the *eclipse* target
-pre-html	before the *html* target
-pre-cleanup-html	before the *cleanup* target for *html*
-post-html	after the *html* target

Hook Target	**Descripton**
-pre-tablehtml	before the *tablehtml* target
-pre-cleanup-tablehtml	before the *cleanup* target for *tablehtml*
-post-tablehtml	after the *tablehtml* target
-pre-framehtml	before the *framehtml* target
-pre-cleanup-framehtml	before the *cleanup* target for *framehtml*
-post-framehtml	after the *framehtml* target
-pre-xhtml	before the *xhtml* target
-pre-cleanup-xhtml	before the *cleanup* target for *xhtml*
-post-xhtml	after the *xhtml* target
-pre-tablexhtml	before the *tablexhtml* target
-pre-cleanup-tablexhtml	before the *cleanup* target for *tablexhtml*
-post-tablexhtml	after the *tablexhtml* target
-pre-framexhtml	before the *framexhtml* target
-pre-cleanup-framexhtml	before the *cleanup* target for *framexhtml*
-post-framexhtml	after the *framexhtml* target
-pre-htmlhelp	before the *htmlhelp* target
-pre-hhc-htmlhelp	before the *hhc* target for *htmlhelp*
-pre-cleanup-htmlhelp	before the *cleanup* target for *htmlhelp*
-post-htmlhelp	after the *htmlhelp* target
-pre-pdf	before the *pdf* target
-pre-cleanup-pdf	before the *cleanup* target for *pdf*
-post-pdf	after the *pdf* target
-pre-rtf	before the *rtf* target
-pre-cleanup-rtf	before the *cleanup* target for *rtf*
-post-rtf	after the *rtf* target
-pre-wordml	before the *wordml* target
-pre-cleanup-wordml	before the *cleanup* target for *wordml*
-post-wordml	after the *wordml* target
-pre-manpages	before the *manpages* target
-pre-cleanup-manpages	before the *cleanup* target for *manpages*
-post-manpages	after the *manpages* target
-pre-fo	before the *fo* target. FO is generated from Docbook source prior to generating pdf with Apache FOP.
-post-fo	after the *fo* target. FO is generated from Docbook source prior to generating pdf with Apache FOP.
-pre-svg	before the *svg* target
-pre-cleanup-svg	before the *cleanup* target for *svg*
-post-svg	after the *svg* target
-pre-ws-tab	before the *ws-tab* target

Hook Target	**Descripton**
-pre-cleanup-ws-tab	before the *cleanup* target for *ws-tab*
-post-ws-tab	after the *ws-tab* target
-pre-website	before the *website* target
-pre-cleanup-website	before the *cleanup* target for *website*
-post-website	after the *website* target

10. Feedback

We'd love to hear from you! Tell us about your experiences with *DobuDish* and about the ways you use it and wish it extended. For that purpose we've set up a discussion group on our website.[5] Feel free to post your comments and suggestions! We'll do our best to help you have a great Docbook experience. In case you really found a bug (just kidding, I'm sure there are some hiding), please use our cool bug tracker to report it to us[6].

[5] http://helpdesk.agynamix.de/index.php?pg=forums.topics&id=1
[6] http://helpdesk.agynamix.de/index.php