

# DocBook Makefile Templates

Michael Wiedmann

`mw@miwie.in-berlin.de`

Copyright © 2001 by Michael Wiedmann

## Revision History

Revision 0.1.16 2001-04-19

## Legal Notice

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no invariant sections, with no Front-Cover texts and no Back-Cover Texts.

## 1. Introduction

This document tries to describe the files and usage of the DocBook Makefile Templates which are heavily based on the makefiles for the FreeBSD Documentation Project.

I adapted the original FreeBSD Makefiles for use on Debian/GNU Linux (<http://www.debian.org>) systems. If anyone supplies the appropriate values for other Linux distributions, I will add them to the corresponding file (look at `doc.os.mk`).

These makefiles are Berkeley Makefiles so you need a Berkeley compatible version of `make`. On Debian/GNU Linux systems `pmake` can be used. On other systems watch out for `bmake`.

Feedback is always welcome!

### 1.1. Files

Following you find a short description of the main files of the distribution.

`pmake.mk`

Master makefile. Most users will only edit this file (or a copy of it). See [below](#) for a more detailed description.

`doc.local.mk`

Included if found in current working directory and holds local extensions. Can e.g. be used for additional automatic graphic file conversion. For details see [Local extensions](#).

`doc.project.mk`

Includes the rest of the Makefiles.

`doc.os.mk`

Holds all operating system specific variables and their values (pathnames, etc.). Known operating system currently are `FREEBSD` and `DEBIAN`.

`doc.images.mk`

Builds targets for automatic conversion of [graphic files](#).

`doc.docbook.mk`

Holds main part.

`doc.subdir.mk`

Contains the default targets for building subdirectories.

`doc.install.mk`

Provides variables defining the default ownership, location, and installation method.

`freebsd.dsl`

Default styleheet file. Make all your [customizations](#) in this file.

## 2. Usage

### 2.1. Putting files in place

In general there are two ways to put the files in place on your system. Use one of the listed possibilities:

- Copy all `*.mk` files to your working directory.
- Copy all `*.mk` files (except `doc.local.mk`) to a directory which is searched by your version of Berkeley Make (`/usr/share/mk/` on Debian).

### 2.2. Master Makefile

You should edit the file `pmake.mk` (or a copy of it) to hold the specific values for your documentation project. You may rename it e.g. to `Makefile`, this lets you just run `pmake` without a command line argument `-f pmake.mk` for setting the name of the Makefile.

The variables in the supplied version of `pmake.mk` are commented so you should be able to guess what they mean.

The most important operators for assigning values to variables are:

=

Assign the value to the variable, overriding any previous value.

+=

Append the value to the current value of the variable.

?=

Assign the value to the variable if it is not already defined.

Most likely you will only use the operator = in your Master Makefile. Experienced users may use ?= if they plan to override some variables from the command line.

### **2.2.1. Required variables**

The following variables need to hold correct project specific values:

#### **DOC**

Set this to the filename of your document without extension.

Default: none

#### **FORMATS**

List all output formats (space separated) which should be generated by invoking the build process without a given target.

Possible values: `html html.tar html-split html-split.tar txt rtf ps pdf tex dvi tar pdb`

Default: none

#### **IMAGES**

List all graphics files used in the document, including filename extension.

Default: empty

#### **SRCS**

List here all source files of your documentation project. This value is used for checking whether all targets are up-to-date and creating tarballs of your project.

Default: none

## OS

If you are on a Debian/GNU Linux system, set this to DEBIAN. The operating system specific values like file and pathnames are controlled by this variable.

Possible values: FREEBSD, DEBIAN

Default: FREEBSD

## 2.2.2. Optional variables

The following variables are optional and control the build process.

### DOCBOOKSUFFIX

Set the document filename extension here if your document uses a filename extension other than the default.

Default: `sgml`

### DSLHTML

Filename (including pathname if necessary) to the DSSSL stylesheet file for HTML output.

Default: `./freebsd.dsl`

### DSLPRINT

Filename (including pathname if necessary) to the DSSSL stylesheet file for print output.

Default: `./freebsd.dsl`

### INSTALL\_COMPRESSED

Set this to the extension of the compression utility you want to use for compressing output files.

Possible values: `gz bz2 zip`

Default: `none`

### GEN\_INDEX

If defined, `index.sgml` will be added to the list of dependencies for source files, and `collateindex.pl` will be run to generate `index.sgml`.

Currently the generation of `index.sgml` does not work yet for PDF/PS output!

Default: empty

#### OPENJADE

If set, use `openjade` to process the document. Using the default stylesheets this allows for creating outlines (bookmarks) which make the navigation easier.

#### PREFIX

If set used as prefix to `jade`, `openjade`, `nsgmls`, and `onsgmls`.

Default: `/usr`

#### PALMDOCTITLE

Used as the title for the `prc/pdb` target.

Default: name of the main document.

## 2.3. Including graphics files

Following the guidelines which Nik Clayton explains in detail in his [posting](#) to the *DocBook ML* from 10 Feb 2001 the Makefiles assume that all used graphic files are referenced in their `<imagedata>` tags *without* a filename extension. The backend drivers choose the most acceptable format. For the necessary customization of the styleheet file see the next section.

The following lines shows how to edit your source file:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="image"> <!-- Filename, without extension -->
  </imageobject>
</mediaobject>
```

All files listed in the *IMAGES* variable are converted to the needed formats. This automatic conversion step may not fully work yet in all cases.

### 2.3.1. Stylesheet Customization for Graphic Files

tbd.

## 2.4. Stylesheet customization

You definitely want to customize the output of your DocBook document. You can do this by editing the stylesheet file `freebsd.dsl`. Look carefully over this file, most variables have meaningful comments so you should be able to figure out what they mean.

More experienced users should additionally look at the original DocBook DSSSL customization files for print and HTML output (`dbparam.dsl`). If you find a variable which isn't yet referenced in `freebsd.dsl` just copy and paste it and set its value accordingly.

On Debian systems you find these files in `/usr/lib/sgml/stylesheet/dsssl/docbook/nwalsh/html/` and `/usr/lib/sgml/stylesheet/dsssl/docbook/nwalsh/print/`.

### 2.4.1. Detailed description of `freebsd.dsl`

The default stylesheet customization file basically is divided into 5 parts: `output.html`, `output.print`, `output.html.images`, `output.print.pdf`, and a section for both HTML and print stylesheets. Each part is conditionally included by corresponding command line arguments to (open) jade (e.g. `-i output.print.pdf`).

tbd.

## 2.5. Building

After editing the master makefile building the desired output formats is simply invoking make like:

```
$ pmake -f pmake.mk
```

For building formats other than the ones listed in `FORMAT` just type e.g.:

```
$ pmake -f pmake.mk PROJECT.ps
```

To create a tarball with all sourcefiles type:

```
$ pmake -f pmake.mk PROJECT.tar.gz
```

For a list of the most important targets see [Targets](#).

## 2.6. Targets

Following a list of the main targets:

`index.html`

Creates chunked HTML output (split at `<section>` level).

`PROJECT.html`

Creates one big HTML file.

PROJECT.txt

Creates a TXT version using `w3m`.

PROJECT.rtf

Creates a RTF file.

PROJECT.pdf

Creates a PDF file.

PROJECT.ps

PostScript output.

PROJECT.prc

Output of PalmPilot `doc` format (used by AprortisDoc Reader).

PROJECT.pdb

Output of PalmPilot `iSilo` format (used by iSilo).

PROJECT.tar.gz

Creates a compressed tarball of all sourcefiles using the content of variable `SRCS`.

validate, lint

Validates the document using `nsxmls` or `onsgmls` (depending whether `OPENJADE` is set or not).

clean

Cleans all files created by the targets in `FORMATS`.

## 2.7. Local extensions

The master Makefile includes the file `doc.local.mk` if it exists in the working directory. You can extend the default functionality of the Makefiles e.g if you need additional targets.

## 3. Special enhancements

In this chapter you will find special tips and tricks for enhancing your DocBook documents. If you want to share your special tip with other users let me know, I will include it here.

### 3.1. Controlling PDF output using `jadetex.cfg`

The output of (pdf) `jadetex` can be finetuned with the file `jadetex.cfg`. Create the file and put it in your working directory. The most important directives for this file are documented here. You

should consult the documentation of the `hyperref` package if you are interested in a complete and more detailed description.

The general layout looks like:

```
\hypersetup{colorlinks=true,  
             pdfpagemode=None,  
             pdftitle={Document Title},  
             pdfsubject={Document Subject},  
             pdfauthor={Authors Name},  
             pdfkeywords={Keyword1 Keyword2}  
}
```

A short explanation of the directives shown above follows:

#### colorlinks

Hyperlinks are shown by default with a surrounding box. If you prefer to show the links coloured, set this to true. The used colours can be configured too, see the `hyperref` documentation for more information.

#### pdfpagemode

Determines how the file is opened in Acrobat Reader. Possible values are `None`, `UseThumbs` (show thumbnails), `UseOutlines` (show bookmarks), and `FullScreen`.

If you choose `UseOutlines` you may find `bookmarksopen` helpful. Default value is `false`, if `true` all the subtrees will be shown expanded. If you have many deep nested subtrees this is maybe not what you want, so you can control the level up to which the subtrees will be expanded with `bookmarksopenlevel`.

#### pdftitle

Title of the document. If set will be shown in the PDF Document Info Title field in Acrobat Reader.

#### pdfsubject

Subject of the document. If set will be shown in the PDF Document Info Subject field in Acrobat Reader.

#### pdfauthor

Author of the document. If set will be shown in the PDF Document Info Author field in Acrobat Reader.

#### pdfkeywords

If set will be shown in the PDF Document Info Keywords field in Acrobat Reader.

### 3.1.1. Notes on PDF Bookmarks (Outlines)

Especially for large documents I find PDF bookmarks (also called Outlines) very helpful because they make navigating in the document much easier.

`jade` is not capable of producing bookmarks, you have to use `openjade` instead - actually this is not a drawback because active development only happens for `openjade` so you should use it anyway.

`openjade` needs a special extension in the stylesheet files to actually create the bookmarks:

```
(declare-characteristic heading-level
  "UNREGISTERED::James Clark//Characteristic::heading-level" 2)
```

Unfortunately the default DSSSL stylesheet file `print/docbook.dsl` contains

```
(declare-characteristic heading-level
  "UNREGISTERED::James Clark//Characteristic::heading-level" 0)
```

so this won't work. You have to override this in your customization DSSSL file instead.

You still have to tell `pdfjadetex` to show the bookmarks in the PDF file with the above mentioned entry in `jadetex.cfg`:

```
pdfpagemode=UseOutlines
```

If you use `jade` to process a file with such a configuration don't get confused: you will get an *empty* window for the bookmarks!

**Note:** The current stylesheets (at least up to V1.64) is buggy which results in an empty bookmark entry for the `bookinfo/title` tag. Norman Walsh currently works on a solution for this.

### 3.1.2. Notes on PDF Thumbnails

**Note:** The following explanation is based on V1.x of `thumbpdf`. The procedure has changed a bit for V2.x. I will update this section as soon as I find some time.

Using the package `thumbpdf` (which should come with your TeX installation) by Heiko Oberdiek you can create thumbnails for every page in your PDF document. These thumbnails provide another way for navigating through your document.

The general procedure looks like follows:

Generate PDF file

There is nothing special for this step. Just run the command to produce the PDF file.

Create the thumbnail files

By running `thumbpdf.pl filename.pdf` the perl script creates small PNG files for every page in the PDF file.

Create second PDF file `thumbpdf.pdf`

In the next step the perl script creates a second PDF file which holds all created PNG files.

Create TeX file `thumbdta.tex`

The perl script creates in the third step a TeX file (`thumbdta.tex`) which holds the object representation of the thumbnails.

Include `thumbpdf.sty`

Include the package `thumbpdf.sty` in your source document and run the command to create PDF output.

I found the following problems with the above outlined procedure:

`ghostscript`

`thumbpdf.pl` uses `ghostscript` to create the PNG files for every page. Only very recent versions of `ghostscript` (V 6.X) are capable of doing this. At least for Debian/GNU Linux systems this version is not yet freely available.

A temporary solution for this is to create the PNG files for every page in a separate step using the tool of your choice.

Including `thumbpdf.sty` into your source file

Because we deal with DocBook source files it's not obvious how to include the generated `thumbpdf.sty`.

The solution is to include it through `jadetex.cfg` like:

```
\usepackage{thumbpdf}
```

Integration in Makefile templates

The described procedure is not integrated in the Makefile templates. So you have to do some steps manually.

## **3.2. Editing SGML/XML files using Emacs and PSGML**

PSGML is an Emacs major mode for editing SGML and XML files. It contains a SGML parser and can work with any DTD. The provided commands let users only insert contextually valid tags, edit attribute values, etc.

If you are already familiar with Emacs you definitely should give PSGML mode a try. It's one of the most powerful and flexible SGML/XML editor you can find.

For special tips on using PSGML mode see [Bob DuCharme's](#) pages.

## 4. Links and other documentation

### 4.1. Links

FreeBSD DocBook Makefile Templates

<http://www.freebsd.org/cgi/cvsweb.cgi/doc/share/mk/>

DocBook Apps ML

Discussion about all DocBook backend related questions. Subscribe with email to: [docbook-apps-request@lists.oasis-open.org](mailto:docbook-apps-request@lists.oasis-open.org) (mailto:docbook-apps-request@lists.oasis-open.org?body=subscribe) with `subscribe` in the body.

Norman Walsh's Homepage

You definitely should have a look at Norman Walsh's (<http://nwalsh.com>) homepage.

PSGML

Homepage: <http://sourceforge.net/projects/psgml> (<http://sourceforge.net/projects/psgml>)

Bob DuCharme's tips on using PSGML mode (<http://www.snee.com/bob/sgmlfree/emcspsgm.html>)

### 4.2. Other documentation

- Berkeley Make on your system should have a manual page which you may consult for an introduction to this flavor of Make.
- ...

## A. File hierarchy

The following picture tries to make clear how all the files work together.

```
pmake.mk +-  
          |  
          +--> include doc.local.mk  
          |
```

```
|
+--+ include doc.project.mk
|
+--> include dos.os.mk
|
+--> include dos.images.mk
|
+--> include doc.dobook.mk
|
+--> include doc.subdir.mk
|
+--> include doc.install.mk
```

## B. How it all began...

A posting of Nik Clayton in DocBook Apps ML was the reason I started adapting the FreeBSD (<http://www.freebsd.org>) Makefile templates to Debian/GNU Linux. The original email is documented [here](#). Read it carefully, it contains all the information you may need ;-)

```
Date: Sat, 10 Feb 2001 00:27:38 +0000
From: Nik Clayton <nik@nothing-going-on.demon.co.uk>
Subject: Re: DOCBOOK-APPS: Problem converting DB to PDF...
To: Dan York <dyork@e-smith.com>
Cc: docbook-apps@lists.oasis-open.org
```

Dan,

```
On Thu, Feb 08, 2001 at 05:28:38PM -0500, Dan York wrote:
> However, because we also would like to have a version available that can
> be easily printed, I have been trying to generate a PDF or PostScript
> file. So far, I have been unsuccessful. The two major problems are:
>
> 1. Graphics do not appear in the PDF file. They are implemented as
> <mediaobject> in the DocBook file.
```

I've been meaning to write this for a while. This is my guide to including images in DocBook as painlessly as possible.

Assuming that you want to create, as a minimum, HTML, PS, and PDF documents with the best quality images, you need to do the following.

First of all, you need to choose your preferred image format(s). This is not as simple as simply picking a single format. The difference between bitmap and vector based image styles means that one single

format won't suffice.

Instead, you need to pick a format that's good for bitmap images, and a format that's good for vector images.

The rest of this document assumes PNG for bitmaps, and EPS for vector.

There's another wrinkle. In my experience, PDF generation works best if you pass `pdftex` the name of a `.pdf` file, and not a `.eps` file. However, EPS files can still be the source from which the PDF is generated.

If you look at DocBook's image inclusion support, you'll see the `<mediaobject>` element, which can contain one or more `<imageobject>`s. The original idea was that, for each image, you would have one `<mediaobject>`, which would contain several different `<imageobject>`s, each pointing to a file in a different format.

The stylesheets would then select which `<imageobject>` to use. This is the approach taken in Norm Walsh's stylesheets.

However, in my experience, this doesn't work quite right, and I had difficulty getting the stylesheets to always select the correct `<imageobject>` element to use. A better approach has been to never include the filename's extension in the `<imageobject>` element's attributes, and let the stylesheets add the extension, or not, as necessary.

A useful side effect of this is that you only ever write one `<imageobject>` per `<mediaobject>`.

So, some sample markup might look like this:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="image"> <!-- Filename, without extension -->
  </imageobject>

  <textobject>
    <phrase>An image</phrase>
  </textobject>
</mediaobject>
```

Of course, this assumes that you have `image.{png,eps,pdf}` in the current directory as well.

If you want to convert a document containing this to HTML, you need to use a stylesheet that customises Norm's sheets, and has

```
(define %graphic-default-extension% "png")
```

in it.

HTML is easy in this respect. PS and PDF are a little more

complicated.

Again, you need to use a customisation of Norm's stylesheets. the following two functions (these work at least up to v1.61 of Norm's sheets). These are re-writes of Norm's functions.

```
----- 8< ----- 8< ----- 8< ----- 8< ----- 8< -----

; Norm's sheets try and work out which one of the <imageobject>s
; should be used. However, we only ever have one, so just use
; the first one.
;
; XXX This can probably be made more efficient by dropping the let*
; clause. One day, I'll get around to testing that.
(define (find-displayable-object objlist notlist extlist)
  (let loop ((nl objlist))
    (if (node-list-empty? nl)
        (empty-node-list)
        (let* ((objdata (node-list-filter-by-gi
                        (children (node-list-first nl))
                        (list (normalize "videodata")
                              (normalize "audiodata")
                              (normalize "imagedata"))))
                (filename (data-filename objdata))
                (extension (file-extension filename))
                (notation (attribute-string (normalize "format") objdata)))
              (node-list-first nl))))))

; This function, given a graphic filename, looks at the filename's
; extension, and appends %graphic-default-extension% as necessary.
;
; However, given a bare filename (such as "image") TeX is perfectly
; capable of adding the .eps or the .pdf as necessary. Rather than
; try and second guess TeX, don't do anything if the tex-backend
; variable is set.
(define (graphic-file filename)
  (let ((ext (file-extension filename)))
    (if (or tex-backend ;; TeX can work this out itself
        (not filename)
        (not %graphic-default-extension%)
        (member ext %graphic-extensions%))
        filename
        (string-append filename "." %graphic-default-extension%)))

----- 8< ----- 8< ----- 8< ----- 8< ----- 8< -----
```

You can see this in the FreeBSD customisation layer, at

<http://www.freebsd.org/cgi/cvsweb.cgi/doc/share/sgml/freebsd.dsl>

We keep both HTML and Print customisation layers in one file. You can do the same thing, or use two different files if you want.

OK, so suppose you have, in one directory, the following files:

```
doc.sgml Your document
```

```
html.dsl Your customisation layer for HTML docs,  
which sets %graphic-default-extension%
```

```
print.dsl Your customisation layer for print docs,  
which contains the two functions  
listed earlier.
```

```
image.png PNG image
```

```
image.eps EPS image
```

```
image.pdf PDF image
```

what are the command lines you need to use?

As I said, HTML is easy.

```
jade -c /your/path/to/the/catalog/files \  
-d html.dsl \  
-t sgml \  
doc.sgml
```

Add things like "-Vnochunks" or whatever, depending on your preference. This should have used the PNG images.

PS is also relatively simple.

```
jade -c /your/path/to/the/catalog/files \  
-d print.dsl \  
-Vtexp-backend \  
-t tex \  
-o doc.tex \  
doc.sgml
```

Notice that you have to give the "-Vtexp-backend" option. I've also shown the use of -o to explicitly set the output file name.

You can then run

```
tex "&jadetex" doc.tex
```

a few times, to generate the .dvi file, and then convert the DVI file to PS.

PDF is a little more complex. As shipped, when producing PDF files, TeX will prefer to include a .png file over a .pdf file. I don't know why that is.

The way to work around this is to make sure that the line

```
\catcode\@=11\def\Gin@extensions{.pdf,.png,.jpg,.mps,.tif}\catcode\@=12
```

appears at the start of the .tex file, before you process it with pdftex. There are many ways in which you can do this.

Anyway, your command line for generating PDF should look like this;

```
jade -c /your/path/to/the/catalog/files \  
-d print.dsl \  
-Vtex-backend \  
-t tex \  
-o doc.tex \  
doc.sgml
```

As you can see, this is the same command line as for generating PS output.

Once you've run this to generate doc.tex, edit doc.tex, and insert the earlier "\catcode..." line at the start of the file. Then you can run

```
pdftex "&pdfjadetex" doc.tex
```

a few times, to generate the .pdf file.

That's that, pretty much. You can see BSD style .mk files that implement all of this, at

```
http://www.freebsd.org/cgi/cvsweb.cgi/doc/share/mk/
```

and pay particular attention to doc.docbook.mk. I'm not aware of any Linux distributions (with scripts like dbtopdf) that make this level of customisation possible. Of course, it would be trivial for you to implement your own replacement scripts which do this. The FreeBSD make(1) approach is, of course, there for the taking, and I'm happy to discuss it further on either this list, or doc@freebsd.org.

<columbo>Oh, and one more thing.</columbo>

As you might be aware, you can use w3m (a text mode browser, with support for tables) to provide very good DocBook -> Text, by first going DocBook -> HTML (as one big file), and then using w3m to convert the HTML to plain text.

Wouldn't it be neat if you could include ASCII art in your document as well, such that when you were going to produce plain text, instead of getting the ALT text on the image, you got ASCII art instead? Well, you can.

First, suppose that your markup looks like this

```
<mediaobject>  
  <imageobject>  
    <imagedata fileref="image"> <!-- Filename, without extension -->  
  </imageobject>
```

```

<textobject>
  <para>+---+
  |  A  |
  +---+</para>
</textobject>

<textobject>
  <phrase>An image</phrase>
</textobject>
</mediaobject>

```

(assume, for the moment, that your image is of a box with the letter A in it).

The HTML stylesheets will search and make sure that the file

```
image.%graphic-default-extension%
```

exists. If the image doesn't exist then the stylesheets will use the contents of the first <textobject> instead.

So if you run something like

```

jade -c /your/path/to/the/catalog/files           \
     -d /path/to/nwalsh's/html/docbook.dsl       \
     -t sgml                                     \
     -Vnochunks \
     doc.sgml > doc.html

```

```
w3m -T text/html -S -dump doc.html > doc.txt
```

Then you will have doc.txt that contains your ASCII art instead. This works because Norm's sheets, by default, do not define a value for %graphic-default-extension%. In case he ever changes this, you might want to create another HTML stylesheet which explicitly sets the variable to #f.

For an example of this in action, take a look at

[http://www.freebsd.org/cgi/cvsweb.cgi/doc/en\\_US.ISO\\_8859-1/article/vm-design](http://www.freebsd.org/cgi/cvsweb.cgi/doc/en_US.ISO_8859-1/article/vm-design)

and examine the files in there.

```

> 2. More importantly, outside of the missing graphics, the PDF file
> looks fine for the first 19 pages, until it gets to Chapter 4. In
> this chapter, I really just have the following construction:
> <sect1>
> <title></title>
> <table>
> ....
> </table>
> <table>
> ....

```

```
> </table>
> </sect1>
```

I've downloaded your document, created a bunch of test images (I'm on a 56K dialup at the moment), and can't replicate this on FreeBSD, using Jade 1.2.1, JadeTeX 2.2, and teTeX 1.0.7.

N

-

Internet connection, \$19.95 a month. Computer, \$799.95. Modem, \$149.95. Telephone line, \$24.95 a month. Software, free. USENET transmission, hundreds if not thousands of dollars. Thinking before posting, priceless. Somethings in life you can't buy. For everything else, there's MasterCard.

- Graham Reed, in the Scary Devil Monastery